En el código QR tendrás acceso a los videotutoriales.

CURSO DE PROGRAMACIÓN EN C

SEGÚN TUTORIALES DE DIEGO MOISSET DE ESPANES

PERE MANEL VERDUGO ZAMORA

pereverdugo@gmail.com
Página web: http://www.peremanelv.com

Contenido

Capítulo 1.- Instalación del compilador y entorno de desarrollo para el lenguaje C Code :: Blocks

En el siguiente enlace podremos acceder a su descarga:

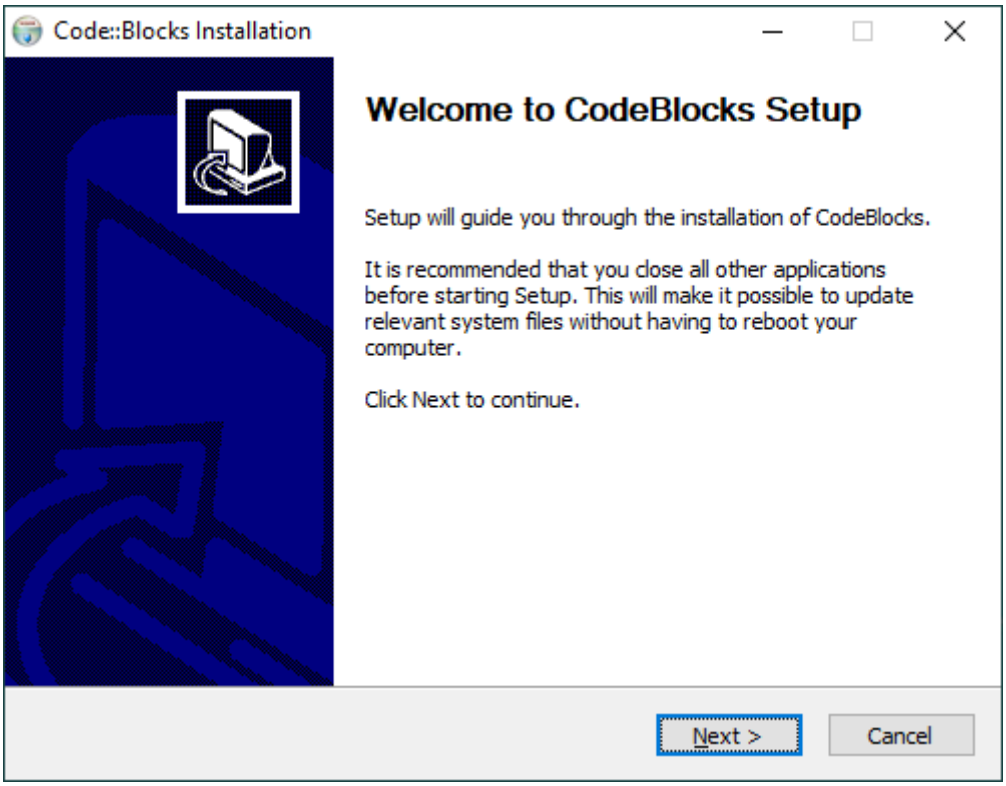
http://www.codeblocks.org/downloads/binaries/#imagesoswindows48pnglogo-microsoft-windows

Microsoft Windows

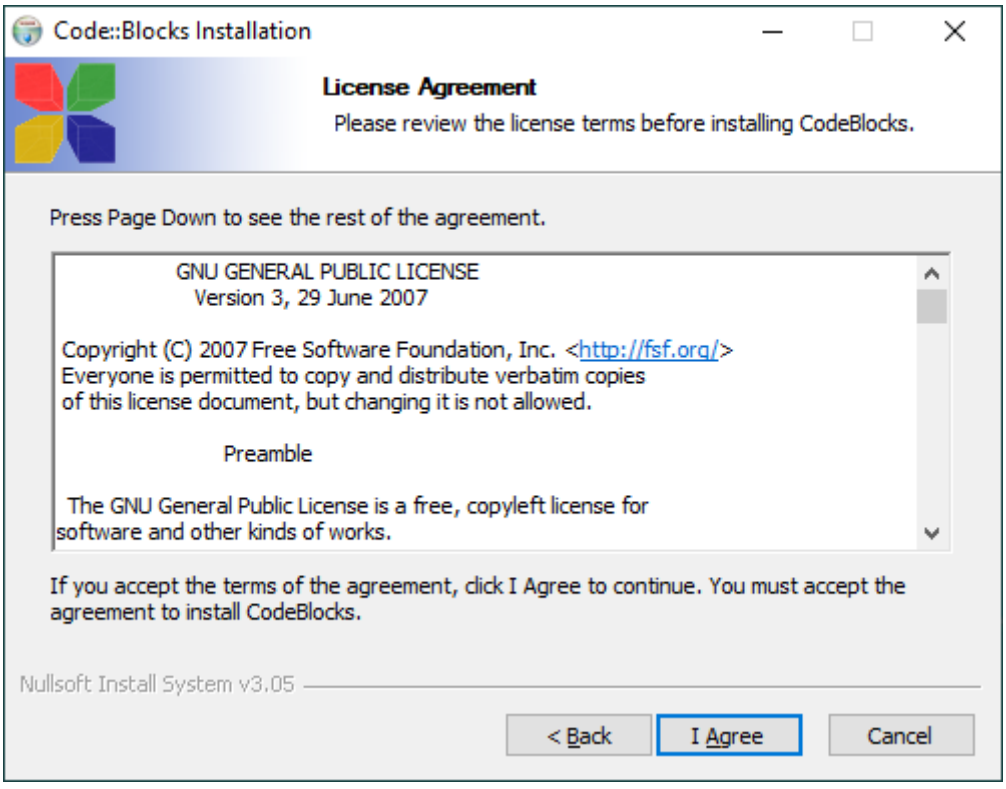
File	Download from
codeblocks-20.03-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03-setup-nonadmin.exe	FossHUB or Sourceforge.net
codeblocks-20.03-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03mingw-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03mingw-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-setup-nonadmin.exe	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03mingw-32bit-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03mingw-32bit-nosetup.zip	FossHUB or Sourceforge.net

Lo descargamos:

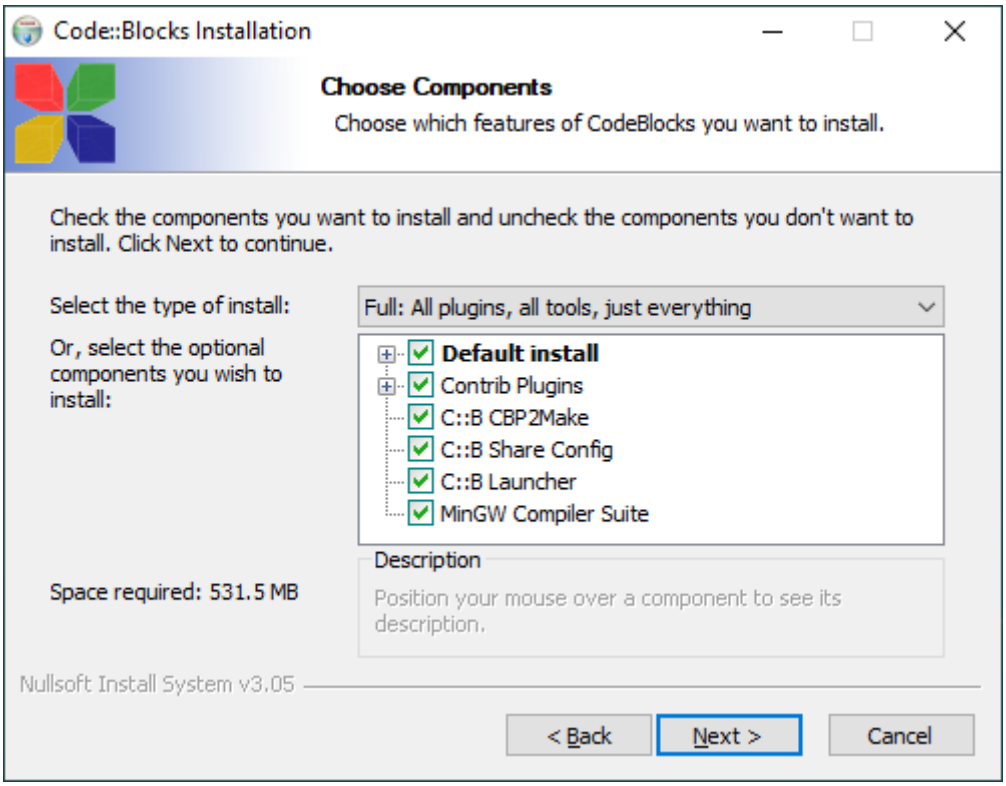
Vamos a instalarlo haciendo doble clic sobre el:



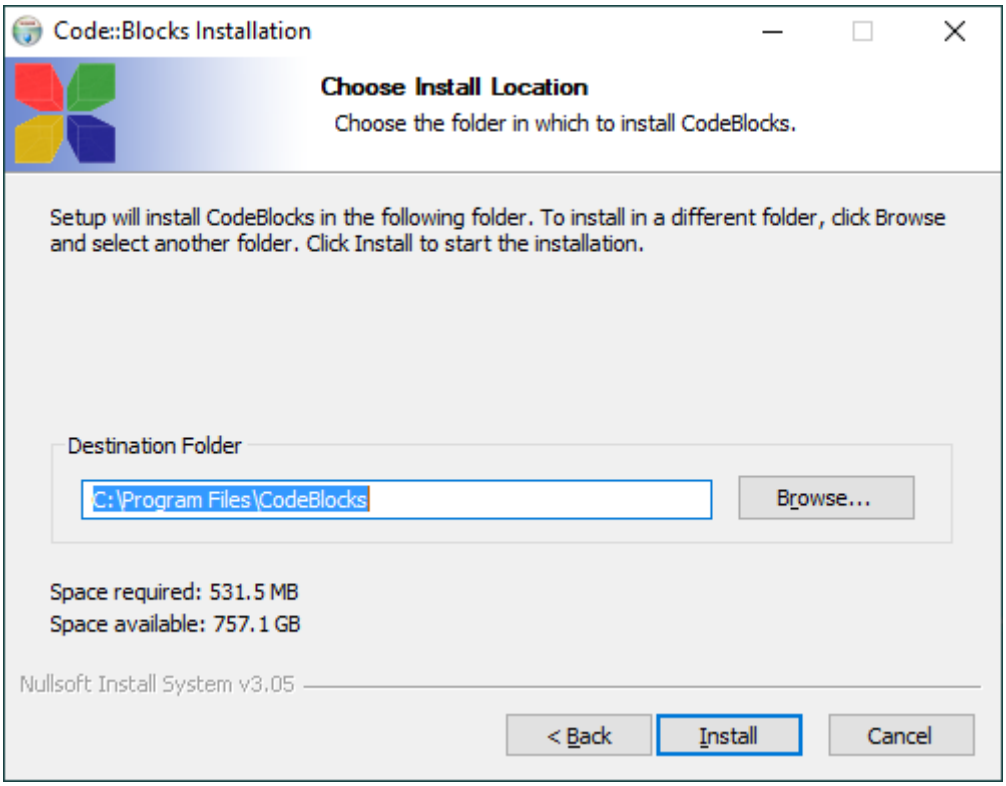
Le damos a Next.



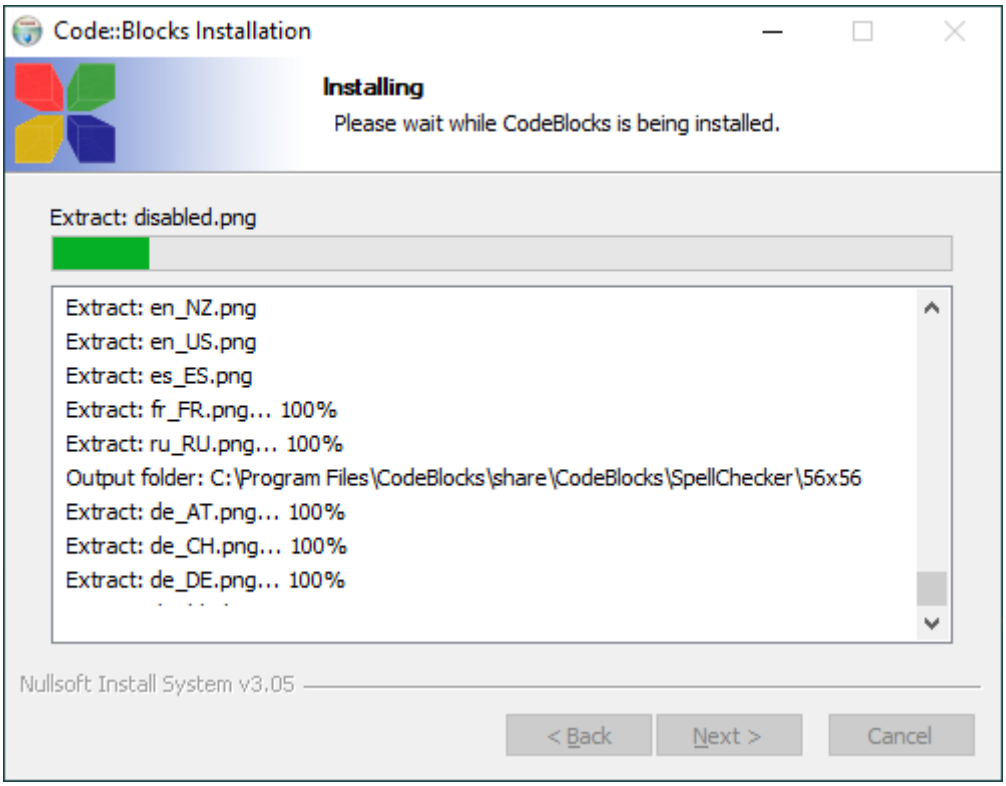
Le damos a I agree.



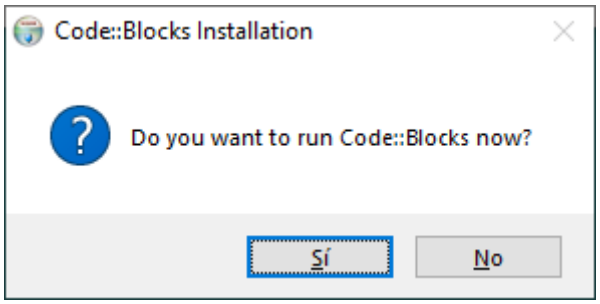
Se damos a Next.



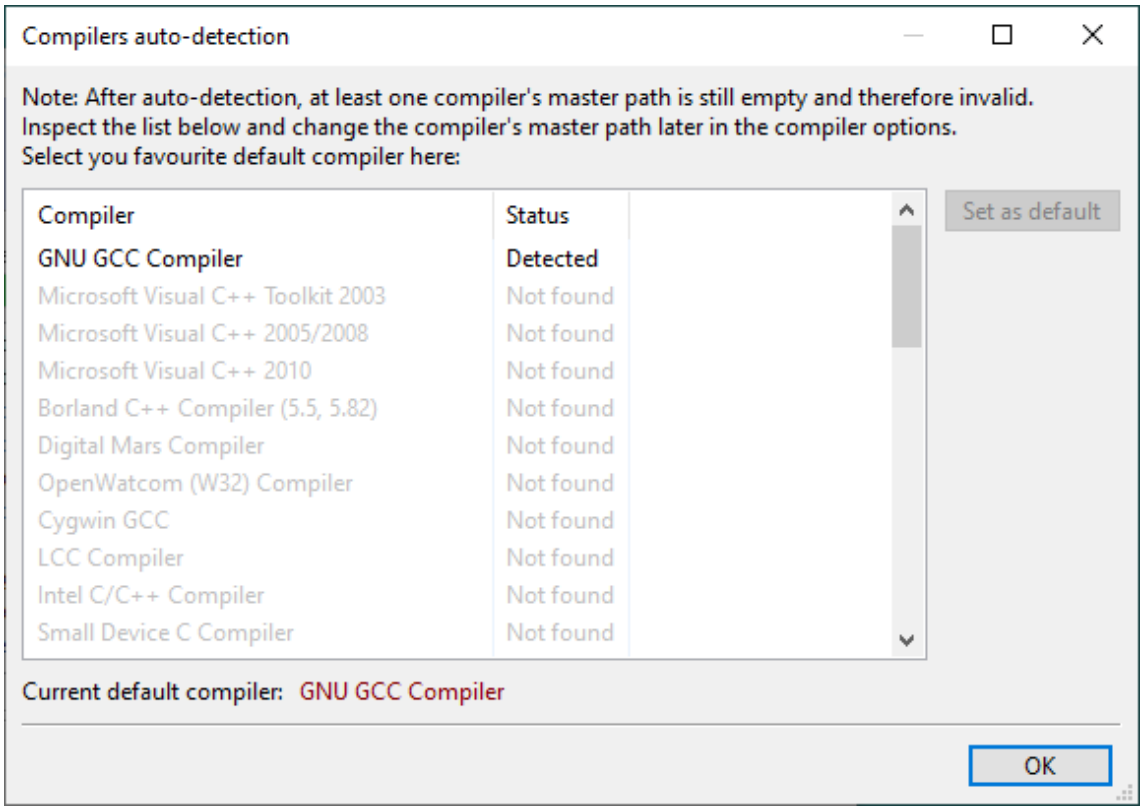
Le damos a Install.



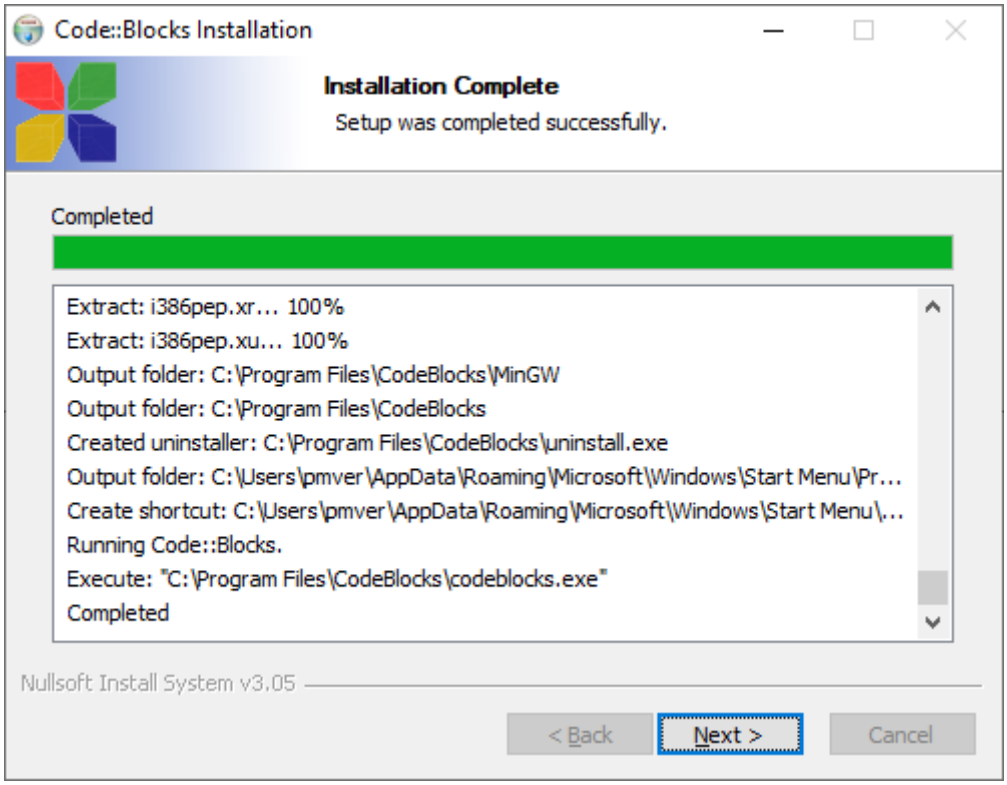
Empieza la instalación.



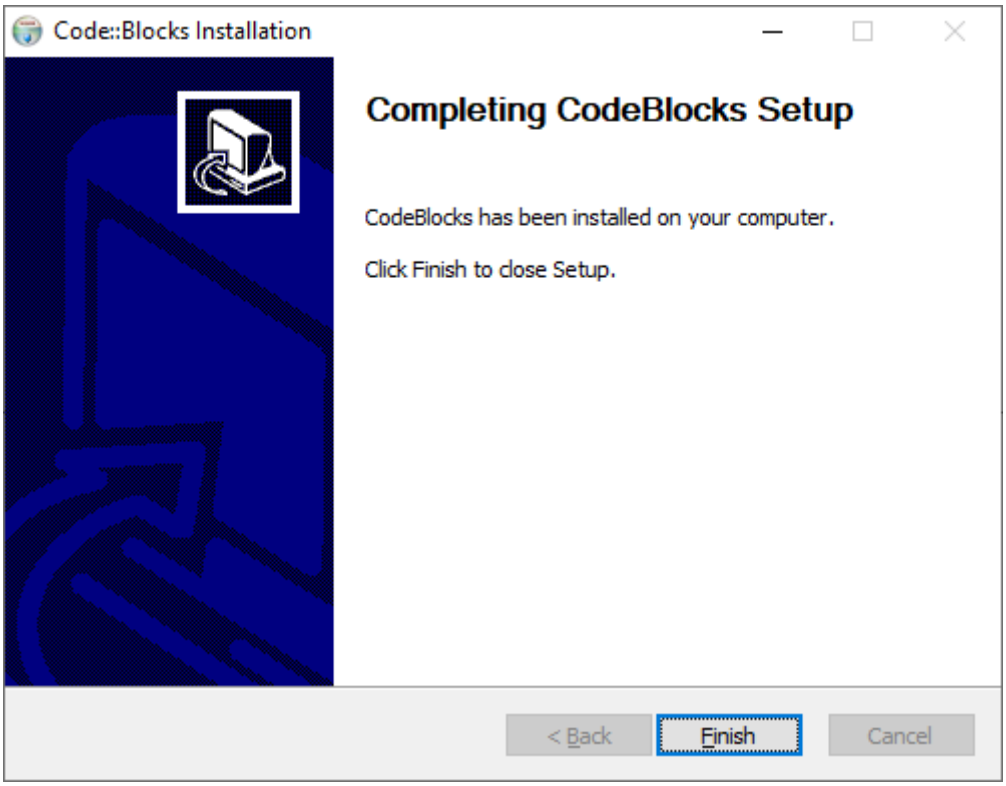
Presionaremos el botón Sí.



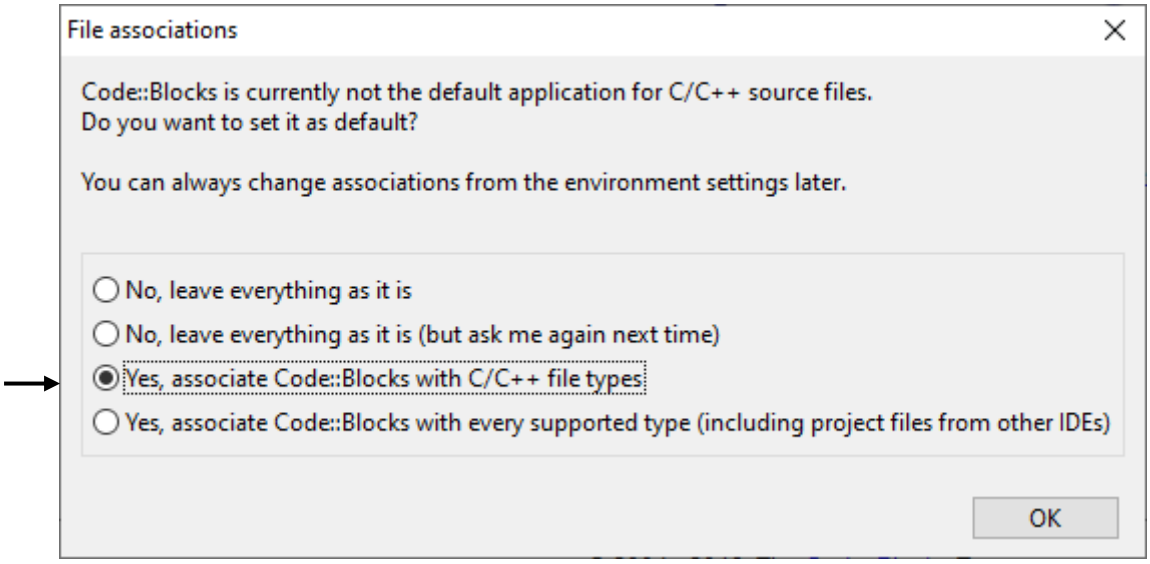
Seleccionamos OK.



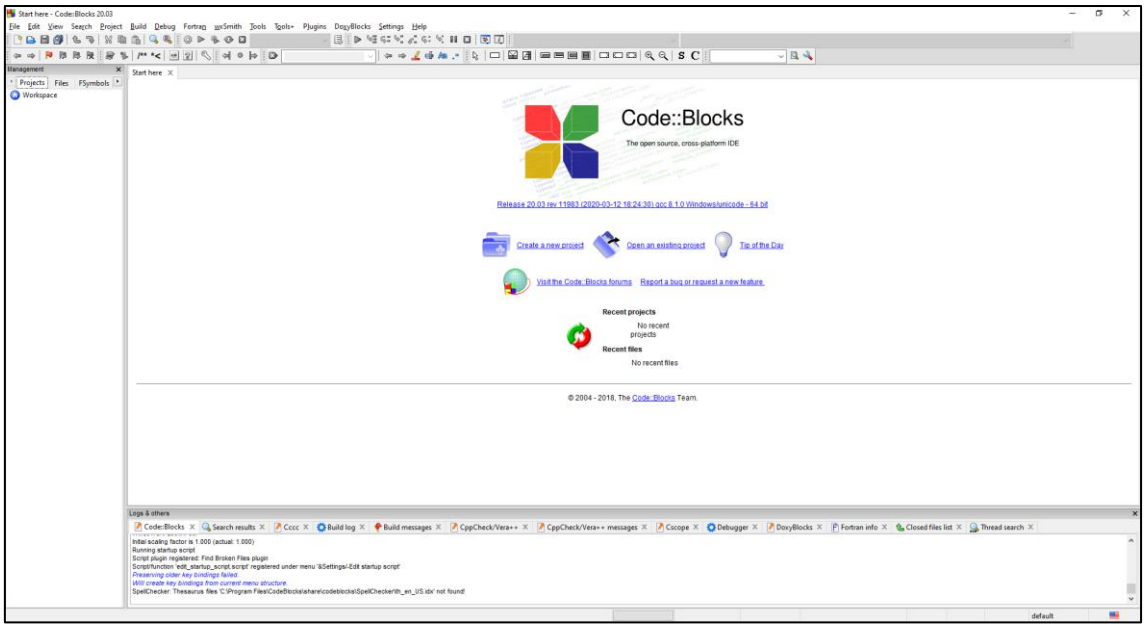
Le damos a Next.

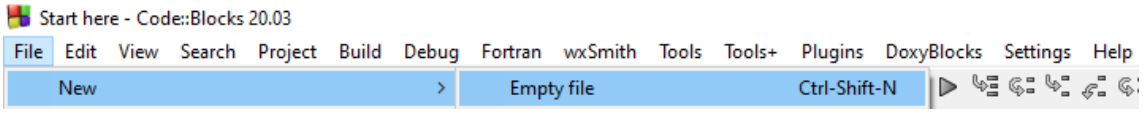


Le damos a Finish.

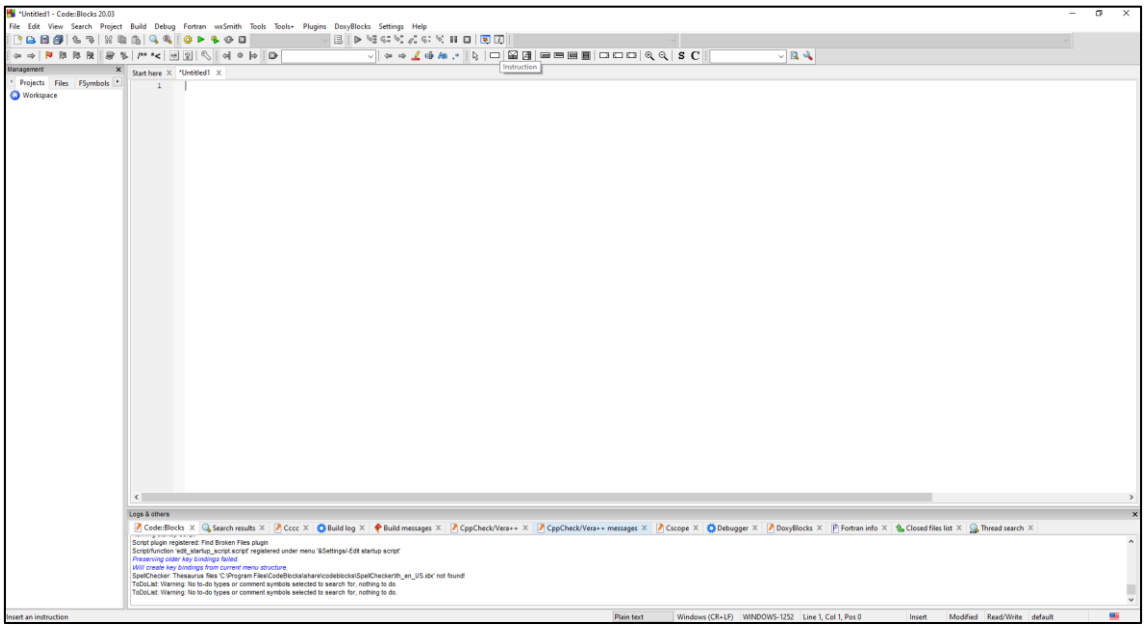


Le damos a OK.





Del menú File seleccionaremos New y de este Empty file.

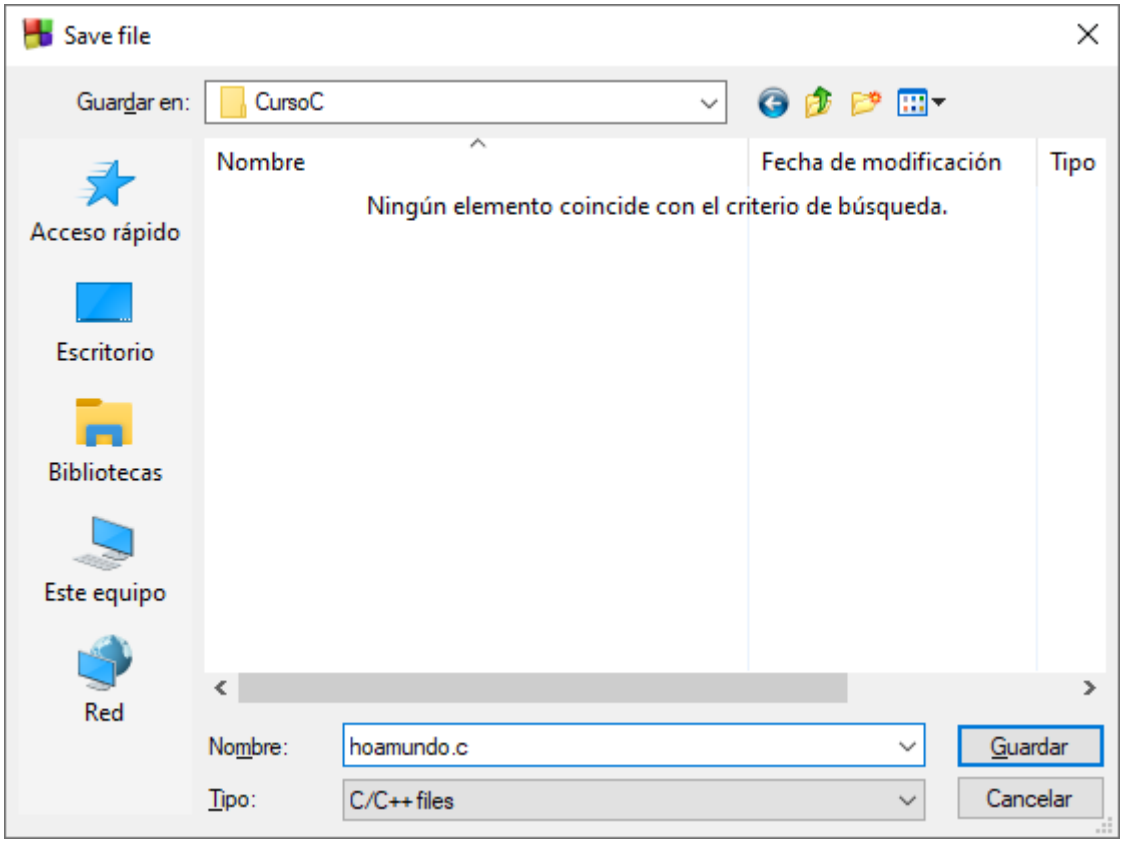


Vamos a escribir un pequeño programa:

```
1   #include<stdio.h>
2
3   int main()
4   {
5       printf("Hola mundo");
6       getchar();
7       return 0;
8   }
```

Vamos a crear una carpeta para guardar nuestros proyectos:

Del menú File seleccionaremos Save file…

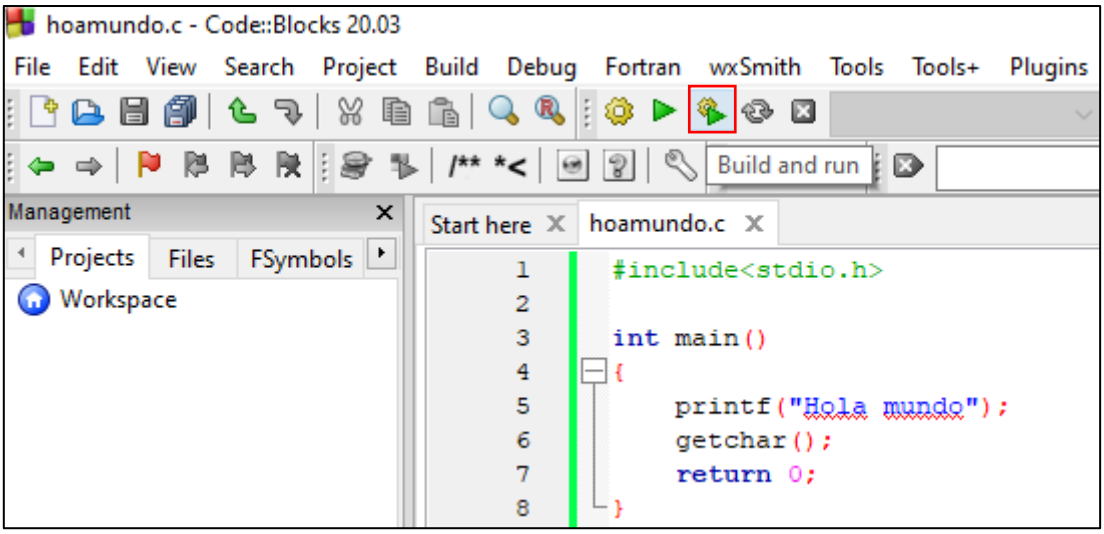


Lo vamos a llamar holamundo.c, seguido del botón Guardar.

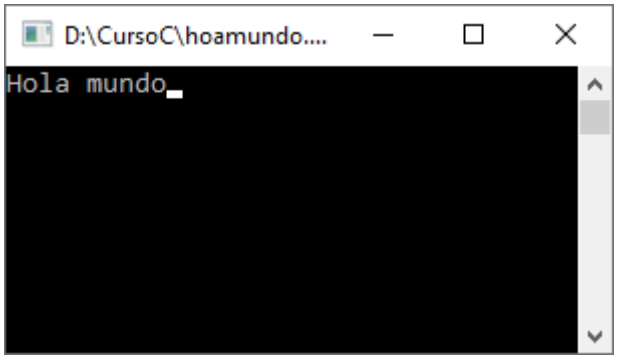
```c
#include<stdio.h>

int main()
{
    printf("Hola mundo");
    getchar();
    return 0;
}
```

Muestra las palabras claves con distintos colores.

Para ejecutar:

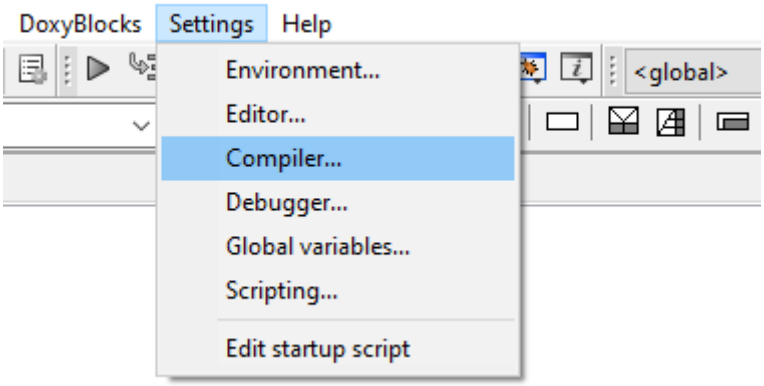


Seleccionamos el botón Buid and run.

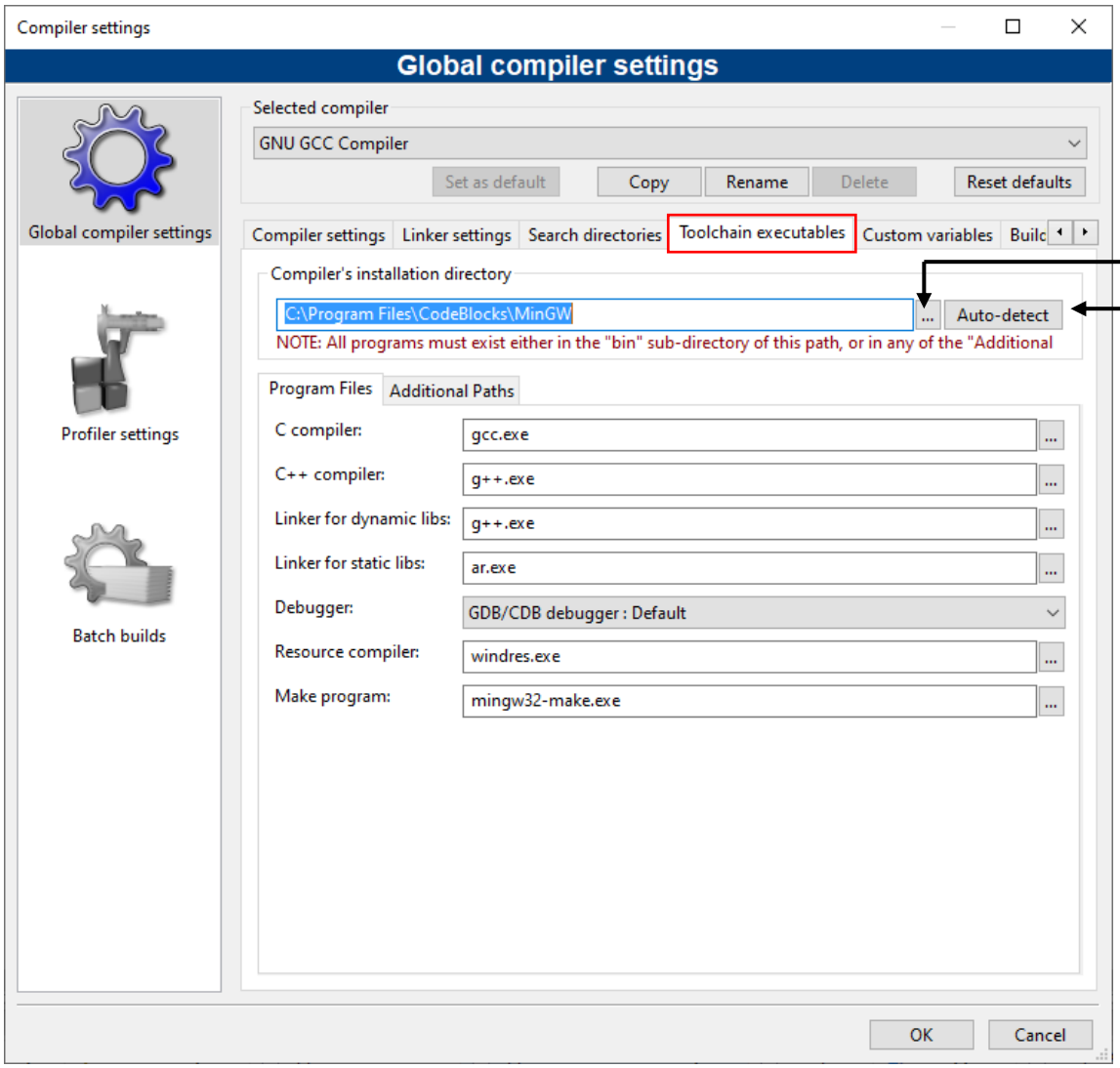


Este será el resultado.

Un posible error del Code::Bloks es que esté mal configurado el compilador.



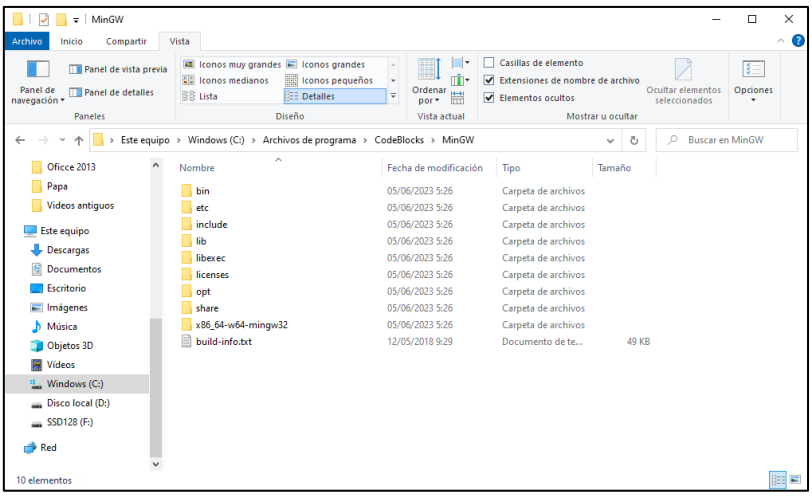
Del menú Settings seleccionaremos Compiler…



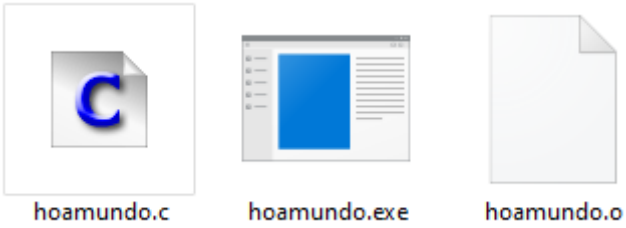
En la pestaña Toolchain executables tiene que estar la ruta del compilador si no es así le damos al botón Auto-detect y si aun así no funciona lo buscamos manualmente con el botón con los 3 puntos.

Normalmente se instala en la carpeta:

C:\Program Files\CodeBlocks\MinGW



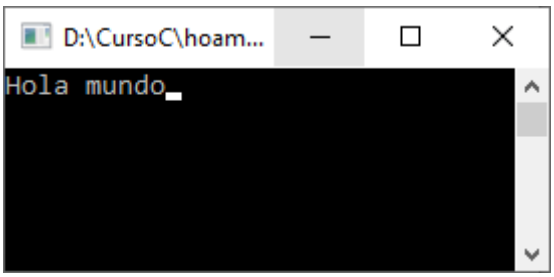
Una vez realizado el primer programa observaremos los siguientes archivos:



holamundo.c es el programa fuente.

holamundo.o en una compilación intermedia, pero sin el enlace de las librerías.

holamundo.exe es el ejecutable, si hacemos doble clic sobre este archivo.



Se muestra el resultado.

Capítulo 2.- Algoritmo y Diagrama de flujo

Codificación del diagrama de flujo en el lenguaje C

¿Qué es un programa?

Programa: Conjunto de instrucciones que entiende una computadora para realizar un actividad. Todo programa tiene un objetivo bien definido. un procesador de textos es un programa que permite cargar, modificar e imprimir textos, un programa de ajedrez permite jugar al ajedrez contra el ordenador u otro contrincante humano.

La actividad fundamental del programador es resolver problemas empleando el ordenador como herramienta fundamental.

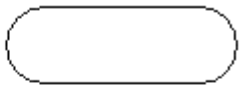
Para la resolución de un problema hay que plantear un algoritmo.

Algoritmo: Son los pasos a seguir para resolver un problema.

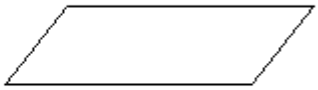
Diagrama de flujo

Un diagrama de flujo es la representación gráfica de un ALGORITMO.

Los símbolos gráficos a utilizar para el planteo de diagramas de flujo son:

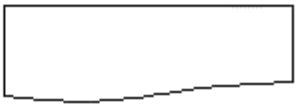


Inicio y fin del diagrama.



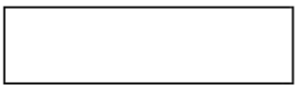
Entrada de Datos

(Vamos a considerar que las entradas de datos se realizan siempre por el teclado de la computadora).

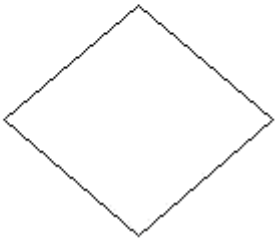


Salida de Datos

(Vamos a considerar que las salidas de datos se realizan siempre por la pantalla de la computadora).



Operación



Condición

Planteo de un problema utilizando diagramas de flujo

Para plantear un diagrama de flujo debemos tener muy en claro el problema a resolver.

Ejemplo: Calcular el sueldo mensual de un operario conociendo la cantidad de horas trabajadas y el precio por hora.

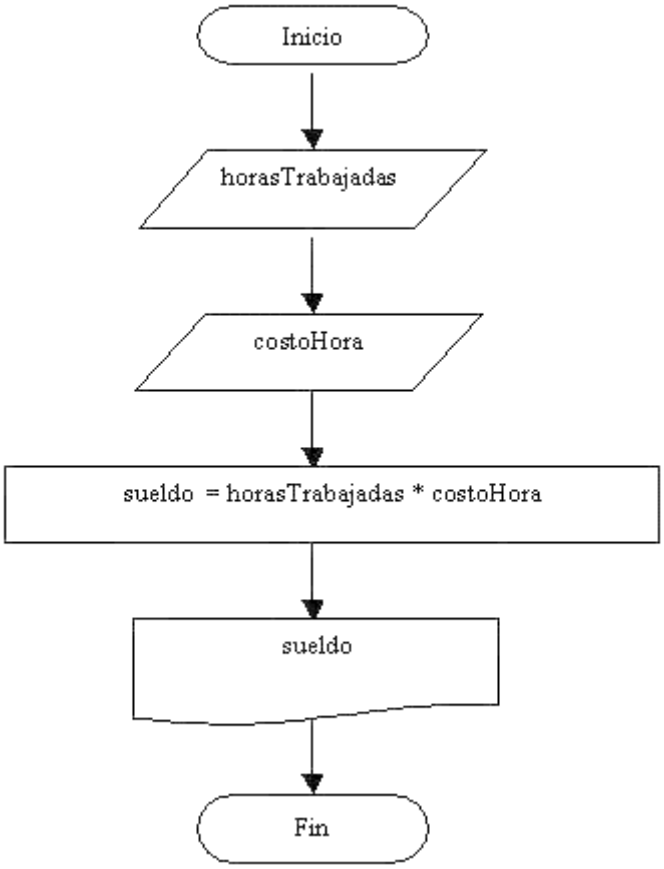
Datos conocidos:
Horas trabajadas en el mes.
Precio hora.

Proceso:
Cálculo del sueldo multiplicando la cantidad de horas por el precio hora.
Información resultante:
Sueldo mensual.

Si hacemos un análisis todo el problema está constituido por:

- Datos conocidos: Datos con los que se cuenta al plantear el problema.
- Proceso: Operaciones a realizar con los datos conocidos.
- Información resultante: Es la información que resuelve el problema.

Esta forma de expresar un problema identificando sus datos conocidos, procesos e información resultante puede llegar a ser engorrosa para problemas complejos donde hay muchos datos conocidos y procesos. Es por eso que resulta mucho más efectivo representar los pasos para que la resolución del problema mediante un diagrama de flujo.



Resulta mucho más fácil entender un gráfico que un texto.

El diagrama de flujo nos identifica claramente los datos de entrada, operaciones y datos de salida.

En el ejemplo tenemos dos datos de entrada: horasTrabajadas y PrecioHora, a las entradas las representamos con un paralelogramo y hacemos un paralelogramo por cada dato de entrada.

La operación se representa con un rectángulo, debemos hacer un rectángulo por cada operación. A la salida la representamos con la hora rota.

El diagrama de flujo nos da una idea del orden de ejecución de las actividades en el tiempo. Primero cargamos los datos de entrada, luego hacemos las operaciones necesarias y por último mostramos los resultados.

Capítulo 3.- Codificación del diagrama de flujo en lenguaje C

Codificación del problema con el lenguaje C

No debemos perder de vista que el fin último es realizar un programa de computación que permita automatizar una actividad para que muchos procesos sean desarrollados por la computadora.

El diagrama de flujo es un paso intermedio para facilitarnos la resolución del problema.

El paso siguiente es la codificación del diagrama de flujo en un lenguaje de computación, en nuestro caso emplearemos el lenguaje C.

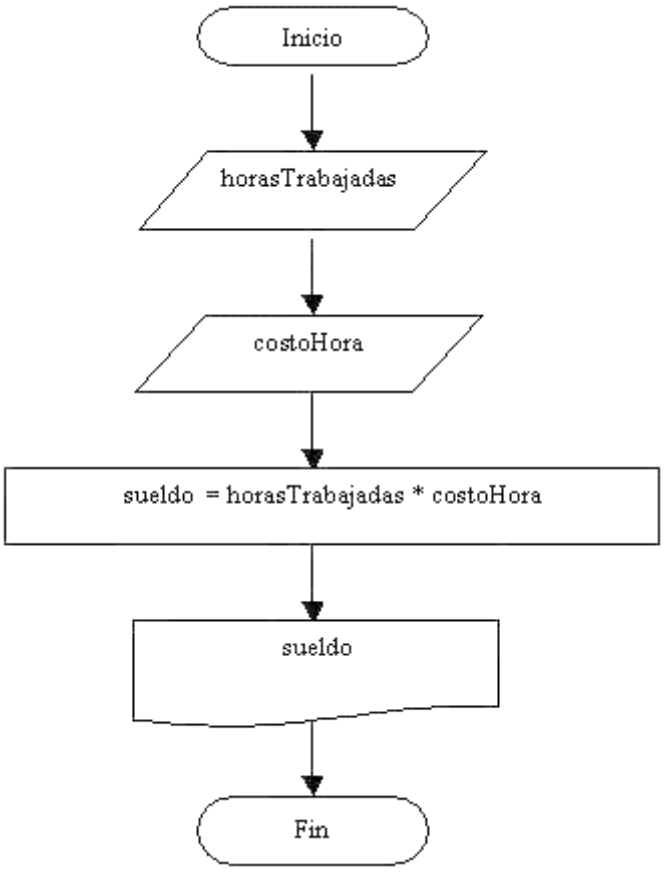
Lenguaje de computación: Conjunto de instrucciones que son interpretadas por una computadora para realizar operaciones, mostrar datos por pantalla sacar listados por impresora, entrar datos por teclado, etc.

Conceptos básicos para codificar un programa

Variable: Es un depósito donde hay un valor. Consta de un nombre y pertenece a un tipo.

Para el ejemplo planteado en el video anterior la variable horasTrabajadas almacena la cantidad de horas trabajadas por el operario. La variable valorHora almacena el precio de una hora de trabajo.

En el ejemplo tenemos tres variables.



Tipos de variable:

Una variable puede almacenar:

- Valores enteros (100, 260, etc.)
- Valores reales (1.24, 2.90, 5.00, etc)
- Cadena de caracteres ("Juan", "Compras", "Listado", etc.)

Elección del nombre de una variable:

Debemos elegir un nombre de variable representativo. En el ejemplo el nombre horasTrabajadas es lo suficientemente claro para darnos una idea acabada sobre su contenido. Podemos darle otros buenos nombres. Otros no son tan representativos, por ejemplo hTr. Posiblemente cuando estemos resolviendo un problema dicho nombre nos recuerde que almacenamos las horas trabajadas por el operario pero pase el tiempo y leamos el diagrama probablemente no recordemos ni entendamos que significa hTr.

Consideraciones a tener en cuenta en cada programa que codifiquemos

Hay que tener en cuenta que el lenguaje C no se ha creado teniendo como objetivo el aprendizaje de la programación. Debido a esto veremos que a medida que avanzamos con el tutorial muchos conceptos que iremos dejando pendientes se irán aclarando.

Veremos los pasos para la creación de un programa en C,

Pasos

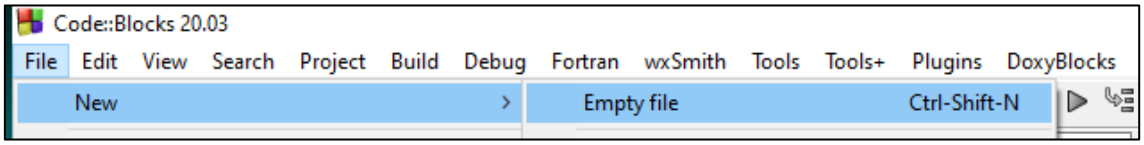
1.- Ingresamos al "Code::Blocks":

2.- Creación de un programa en C.

Lo primero que haremos es darle un nombre al archivo y grabarlo en el disco dura de nuestro ordenador.

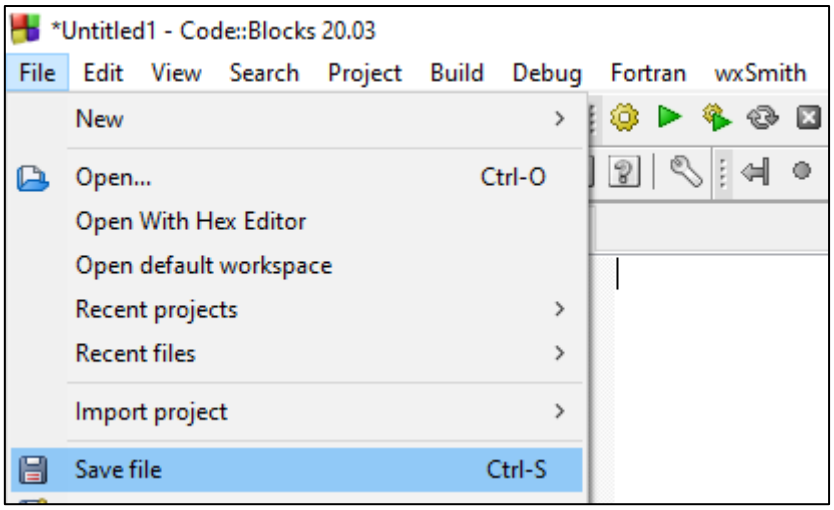
Podemos crear una carpeta donde almacenar todos nuestros programas.

Al programa que codificamos lo llamaremos "programa1.c" la extensión del archivo es importante que sea "c".

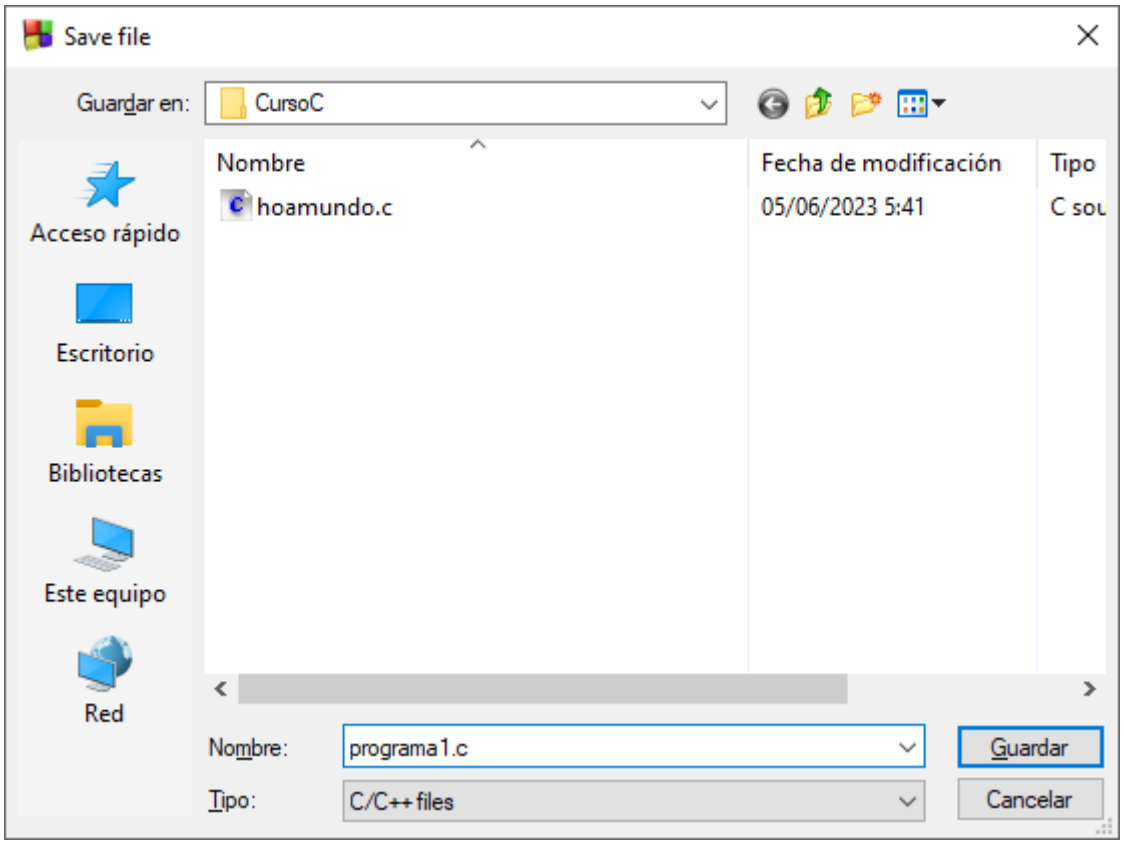


Creamos un archivo nuevo desde el menú File / new / Empty file.

El siguiente paso será guardar el archivo.



Del menú File seleccionaremos Save file.



Nos iremos a una carpeta que hemos preado previamente para todos los programas que vayamos a realizar y como nombre de nuestro primer proyecto programa1.c, seguido del botón Guardar.

Es importarse de no olvidarse de la extensión que tiene que ser .c.

Ahora debemos codificar el diagrama de flujo utilizando las instrucciones del lenguaje C.

A medida que avancemos en el curso veremos que significa las palabras clave include, return, etc.

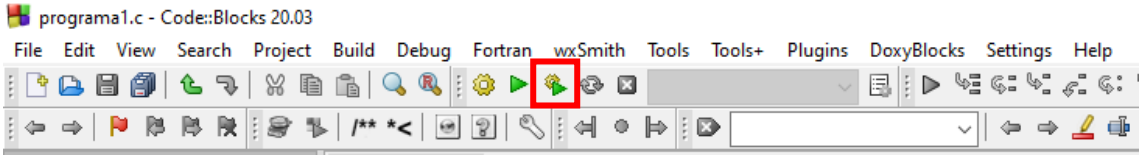
Por el momento nos centraremos donde codificaremos nuestro diagrama de flujo.

La codificación del diagrama del flujo lo haremos dentro de la función main (la función main es la primera que se ejecuta al iniciarse un programa en lenguaje C).

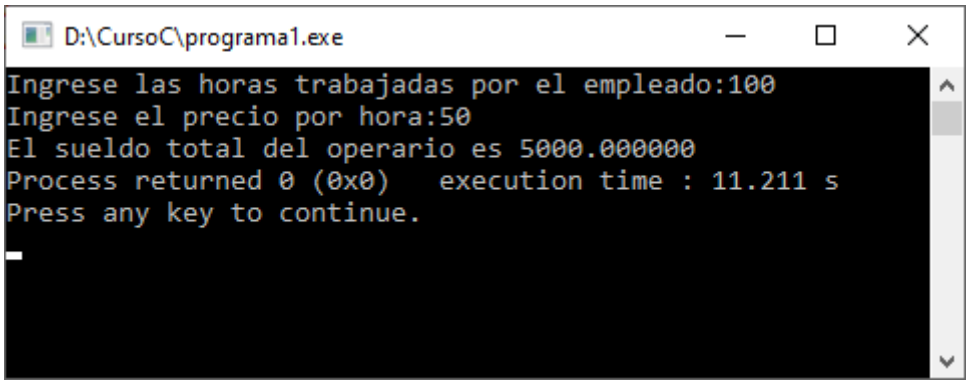
El programa completo para el cálculo del sueldo de un operario conociendo la cantidad de horas trabajadas y el precio hora es (copiemos estas líneas en el editor del Code::Blocks dentro de la ventana programa1.c):

```c
#include<stdio.h>

int main()
{
    int horasTrabajadas;
    float precioHora;
    float sueldo;
    printf("Ingrese las horas trabajadas por el empleado:");
    scanf("%i",&horasTrabajadas);
    printf("Ingrese el precio por hora:");
    scanf("%f",&precioHora);
    sueldo=horasTrabajadas*precioHora;
    printf("El sueldo total del operario es ");
    printf("%f",sueldo);
    getchar();
    return 0;
}
```

Para probar el funcionamiento del programa debemos presionar el icono con un triángulo verde y la rueda dentada (o la tecla F9).

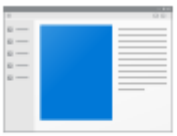


Como podemos ver se abre una ventana con la ejecución del programa que acabamos de compilar donde debemos ingresar las horas trabajadas por un operario y el precio hora, dando como resultado el sueldo que debe cobrar.



Tener en cuenta que debemos ingresar el carácter "." en lugar del carácter ',' si ingresamos un precio por hora con decimales.

Cuando lo ejecutamos se crea el programa ejecutable "programa1.exe" que es el resultado de la compilación del código fuente que escribimos en el editor en el archivo "programa1.c".



programa1.exe

Conceptos que deben quedar claros:

1.- Por el momento haremos todo el algoritmo dentro de la función main.

2.- Si observamos el diagrama de flujo vemos que debemos definir tres variables:

(horasTrabajadas, precioHora, sueldo), aquí es donde debemos definir que tipos de datos se almacenarán en la misma. La cantidad de horas normalmente será un valor entero (ej. 100, -150, -230 etc.), pero el precio hora es muy común que sea un valor real (ej. 5.35, -7.50, etc.) y como el sueldo resulta de multiplicar las horas trabajadas por el precio hora el mismo deberá ser real.

La definición de las variables la hacemos en la función main:

```
int horasTrabajadas;
float costoHora;
float sueldo;
```

Utilizaremos la palabra clave int para definir una variables enteras (en C las palabras claves deben ir obligatoriamente en minúsculas, sino se produce un error sintáctico). Luego de la palabra clave debemos indicar el nombre de la variable, por ejemplo: horasTrabajadas. SE propone que el nombre de la variable comience con minúsculas y en caso de estar construida por dos palabras o más deben ir en mayúsculas el primer carácter (un nombre de variable no puede tener espacios en blanco, empezar con número, si tampoco utilizar caracteres especiales).

Debemos buscar siempre nombres de variables que nos indiquen que almacenan (no es conveniente llamar a nombres de variables que nos indiquen lo que almacenan (no es conveniente llamar a nombres de variables con letras individuales salvo en casos puntuales que veremos más adelante).

3.- Para mostrar mensajes en la pantalla utilizamos la función "printf":

```
printf("Ingrese las horas trabajadas por el empleado:");
```

Con esta sintaxis todo lo que se encuentra contenido entre comillas aparecerá exactamente en pantalla.

Si necesitamos mostrar el contenido de una variable debemos disponer del carácter %f si mostramos una variable float o %i si mostramos una variable int (luego de la cadena a mostrar disponemos separado por coma la variable a mostrar):

```
printf("%f",sueldo);
```

4.- Para hacer la entrada de datos por teclado en C debemos utilizar la función "scanf".

```
scanf("%i",&horasTrabajadas);
```

La función scanf tiene un primer parámetro entre comillas dobles y dentro del carácter % y la letra i se carga un entero y la letra f se se carga un float.

El segundo parámetro es el nombre de la variable a cargar antecedida por el carácter &.

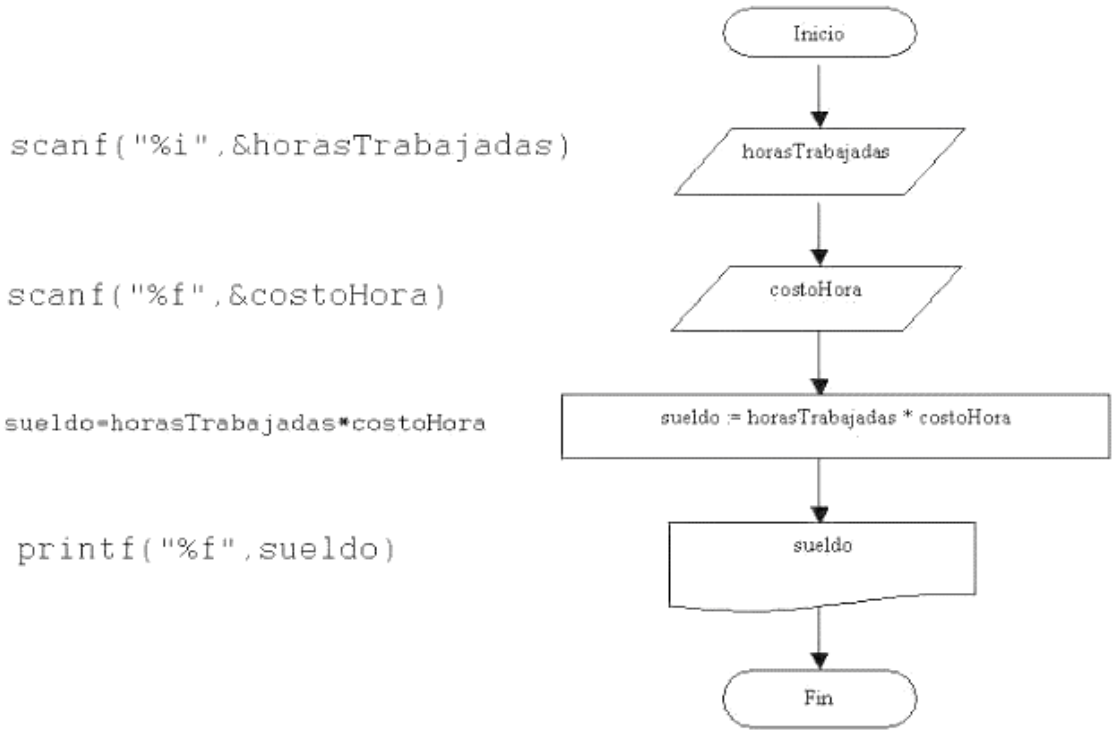
5.- Las operaciones que indicamos en el diagrama de flujo mediante la figura rectángulo la codificamos tal cual:

```
sueldo=horasTrabajadas*costoHora;
```

6.- Por el momento veremos que siempre nuestro programa finaliza llamando a la función fetchar() (detiene la ejecución del programa hasta que se presiona una tecla) y la instrucción return 0 (le avisa al sistema operativo que el programa finalizó correctamente):

```
getchar();
return 0;
```

Podemos ver una relación entre las instrucciones que debemos utilizar para cada símbolo del diagrama de flujo:



En el diagrama de flujo no indicamos la definición de variables:

```
int horasTrabajadas;
float costoHora;
float sueldo;
```

No representamos son símbolos los mensajes a mostrar previo a la carga de datos por teclado:

```
printf("Ingrese las horas trabajadas por el empleado:");
```

Como hemos visto hasta ahora has varias partes del nuestro código que no entendemos:

```
#include<stdio.h>
```

Pero son indispensable para la implementación de nuestros programas, a medida que avancemos con el curso muchos de estos conceptos se irán aclarando (el include permite importar librerías de funciones contenidos en otros archivos).

La línea:

```
getchar();
```

No permite detener la ejecución del programa para ver el sueldo del operario y esperar a que se presione una tecla.

Capítulo 4.- Errores sintácticos y lógicos

Confeccionaremos un problema y agregaremos adrede una serie de errores tipográficos. Este tipo de errores siempre son detectados por el COMPILADOR, antes de ejecutar el programa.

A los errores tipográficos, como por ejemplo la falta de puntos y comas, nombres de variables incorrectas, falta de paréntesis, palabras claves mal escritas, etc. los llamamos errores SINTACTICOS.

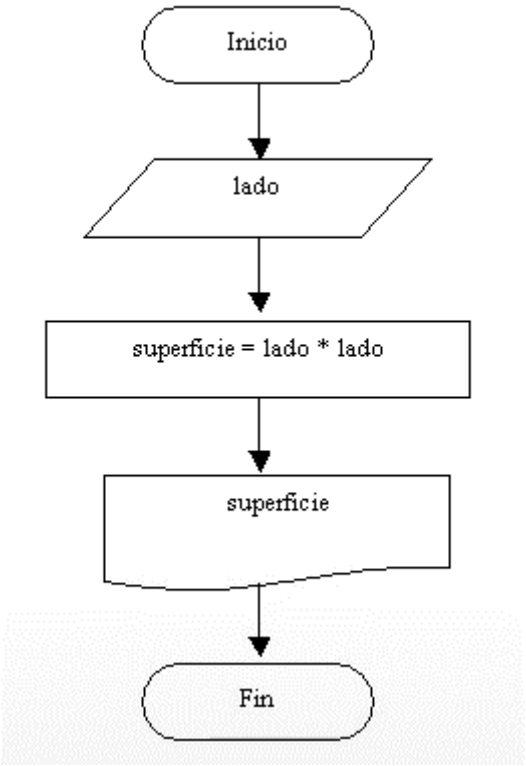
Un programa no se puede ejecutar sin corregir absolutamente todos los errores sintácticos.

Existen otro tipo de errores llamados ERRORES LOGICOS. Este tipo de errores en programas grandes (miles de líneas) son más difíciles de localizar. Por ejemplo un programa que permite hacer la facturación pero la salida de datos por impresora es incorrecta.

Problema:

Hallar la superficie de un cuadrado conociendo el valor de un lado.

Diagrama de flujo:



Creemos un archivo con el editor Code::Blocks llamado: programa2.c (Recordemos que lo hacemos desde el menú de opciones File -> New => Empty file y luego para dar nombre nuevamente desde el menú de opciones elegimos File -> Save file).

Codificamos el algoritmo en C e introducimos un error sintáctico:

1.- En el momento que calculamos la superficie hacemos referencia a la variable con el primer carácter en mayúsculas, es decir Superficie.

```c
#include<stdio.h>

int main()
{
    int lado;
    int superficie;
    printf("Ingrese el valor del lado del cuadrado:");
    scanf("%i",&lado);
    Superficie=lado*lado;
    printf("La superficie es %i", superficie);
}
```

Como podemos observar aparece en una ventana en la parte inferior un mensaje de error sintáctico detectado.

```
D:\CursoC\pro...   9    error: 'Superficie' undeclared (first use in this function); did you mean 'superficie'?
D:\CursoC\pro...   9    note: each undeclared identifier is reported only once for each function it appears in
                        === Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===
```

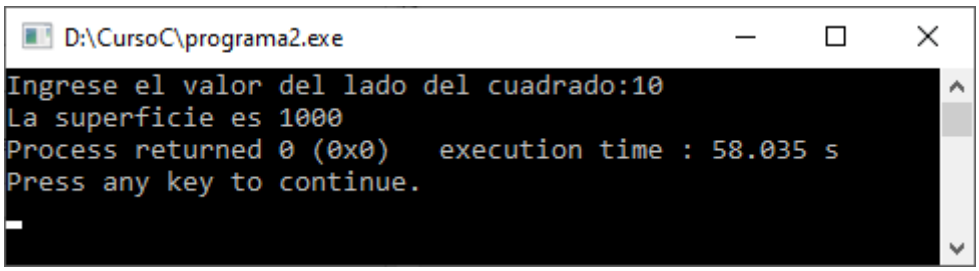
Hay muchos tipos de errores sintácticos que nos pueden suceder como que nos falte punto y coma al final de cada instrucción, definir variables en forma incorrecta, nombres de funciones con caracteres equivocados, etc.

Programa con un error lógico

```c
superficie = lado * lado * lado;
```

```c
#include<stdio.h>

int main()
{
    int lado;
    int superficie;
    printf("Ingrese el valor del lado del cuadrado:");
    scanf("%i",&lado);
    superficie=lado*lado*lado;
    printf("La superficie es %i", superficie);
}
```

Si ejecutamos este será el resultado:



Los errores lógicos no los detecta el compilador.

Capítulo 5.- Estructura de programación secuencial – 1

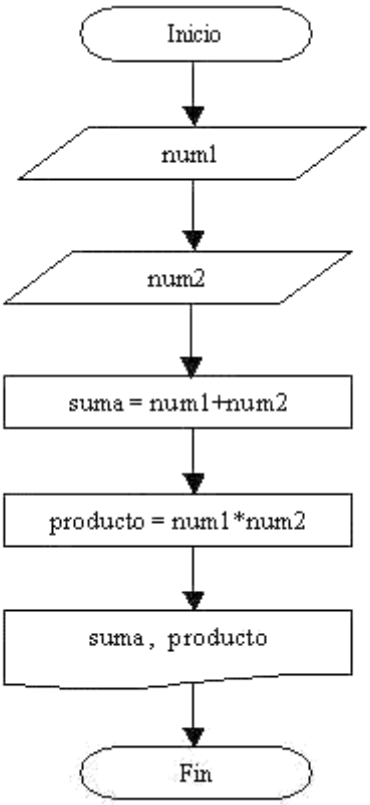
Cuando en un problema sólo participan operaciones, entradas y salidas se las denomina una estructura secuencial.

Los problemas diagramados y codificados previamente emplean solo estructuras secuenciales.

Problema:

Realiza la carga de dos números enteros por teclado e imprimir su suma y su producto.

Diagrama de flujo:



Tenemos dos entradas num1 y num2, dos operaciones: calculas la suma y el producto de los valores ingresados y dos salidas, que son los resultados de la suma y el producto de los valores ingresados.

En el símbolo de impresión podemos indicar una o más salidas, eso queda a criterio del programador, lo mismo para indicar las entradas por teclado.

Para poder utilizar las funciones printf y scanf debemos importar el archivo donde se las declaran:

```
#include<stdio.h>
```

Para poder utilizar la función getch debemos importar:

```
#include<conio.h>
```

Cuando instalamos el Code::Blocks se instaló entre otras cosas un archivo llamado stdio.h que contiene la declaración de las funciones y que son indispensables para efectuar las salidas por pantallas (printf) y las entradas por teclado (scanf).

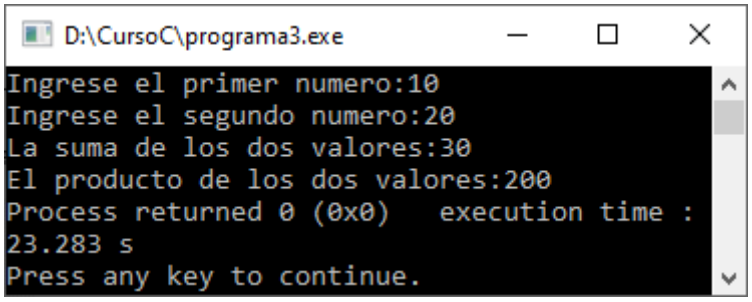
Tener en cuenta que el lenguaje C es sensible a mayúsculas y minúsculas por lo que no podemos escribir:

```
#INCLUDE<stdio.h>
```

Vamos a programar:

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   int main()
5   {
6       int num1, num2, suma, producto;
7       printf("Ingrese el primer numero:");
8       scanf("%i", &num1);
9       printf("Ingrese el segundo numero:");
10      scanf("%i", &num2);
11      suma=num1+num2;
12      producto=num1*num2;
13      printf("La suma de los dos valores:");
14      printf("%i", suma);
15      printf("\n");
16      printf("El producto de los dos valores:");
17      printf("%i", producto);
18      getch();
19      return 0;
20  }
21
```

Si ejecutamos este será el resultado:



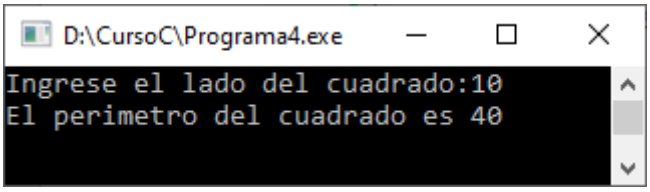
Capítulo 6.- Estructura de programación secuencial – 2

Problema propuesto

Realizar la carga del lado de un cuadrado, mostrar por pantalla el perímetro del mismo (El perímetro de un cuadrado se calcula multiplicando el valor del lado por cuatro).

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   int main()
5   {
6       int lado, perimetro;
7       printf("Ingrese el lado del cuadrado:");
8       scanf("%i",&lado);
9       perimetro=lado*4;
10      printf("El perimetro del cuadrado es ");
11      printf("%i",perimetro);
12      getch();
13  }
14
```

Si ejecutamos este será el resultado:



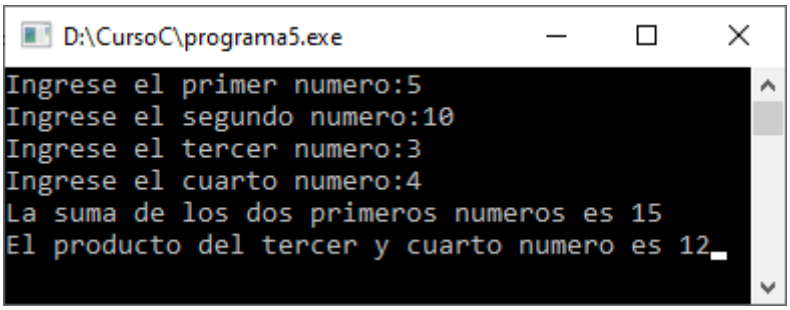
Capítulo 7.- Estructura de programación secuencial – 3

Problema propuesto

Escribir un programa en el cual se ingresen cuatro números, calcular e informar la suma de los dos primeros y el producto del tercero y cuarto.

```
1  #include<stdio.h>
2  #include<conio.h>
3
4  int main()
5  {
6      int num1, num2, num3, num4, suma, producto;
7      printf("Ingrese el primer numero:");
8      scanf("%i", &num1);
9      printf("Ingrese el segundo numero:");
10     scanf("%i", &num2);
11     printf("Ingrese el tercer numero:");
12     scanf("%i", &num3);
13     printf("Ingrese el cuarto numero:");
14     scanf("%i", &num4);
15     suma=num1+num2;
16     producto=num3*num4;
17     printf("La suma de los dos primeros numeros es ");
18     printf("%i", suma);
19     printf("\n");
20     printf("El producto del tercer y cuarto numero es ");
21     printf("%i", producto);
22     getch();
23     return 0;
24 }
25
```

Si ejecutamos este será el resultado:



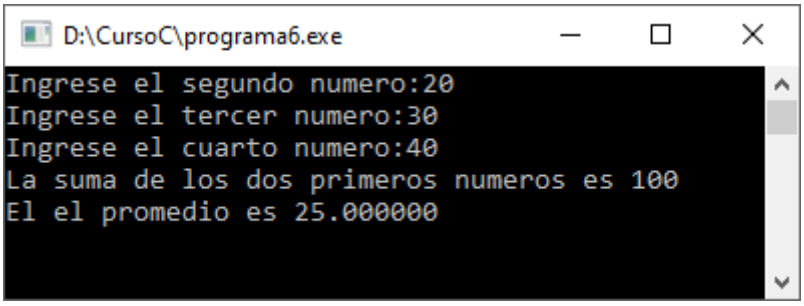
Capítulo 8.- Estructura de programación secuencial – 4

Problema propuesto

Realizar un programa que lea cuatro valores numéricos e informar de su suma y promedio.

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int num1, num2, num3, num4, suma;
    float promedio;
    printf("Ingrese el primer numero:");
    scanf("%i", &num1);
    printf("Ingrese el segundo numero:");
    scanf("%i", &num2);
    printf("Ingrese el tercer numero:");
    scanf("%i", &num3);
    printf("Ingrese el cuarto numero:");
    scanf("%i", &num4);
    suma=num1+num2+num3+num4;
    promedio=suma/4;
    printf("La suma de los dos primeros numeros es ");
    printf("%i", suma);
    printf("\n");
    printf("El el promedio es ");
    printf("%f", promedio);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 9.- Estructura de programación secuencial – 5

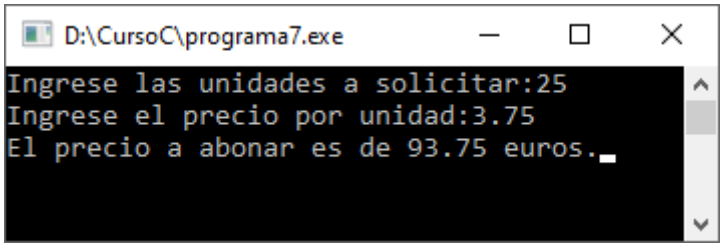
Problema propuesto

Se debe desarrollar un programa que pida el ingreso del precio de un artículo y la cantidad que lleva el cliente. Mostrar lo que debe abonar el comprador.

Definir una variable float para el precio del artículo.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   int main()
5   {
6       int unidades;
7       float precio, total;
8       printf("Ingrese las unidades a solicitar:");
9       scanf("%i", &unidades);
10      printf("Ingrese el precio por unidad:");
11      scanf("%f", &precio);
12      total=precio*unidades;
13      printf("El precio a abonar es de %.2f euros.", total);
14      getch();
15      return 0;
16  }
```

Si ejecutamos este será el resultado:



Capítulo 10.- Estructuras condicionales simples y compuestas – 1

Estructuras condicionales simples

No todos los problemas pueden resolverse empleando estructuras secuenciales.

Cuando hay que tomar una decisión aparecen las estructuras condicionales.

En nuestra vida diaria se nos presentan situaciones donde debemos decidir.

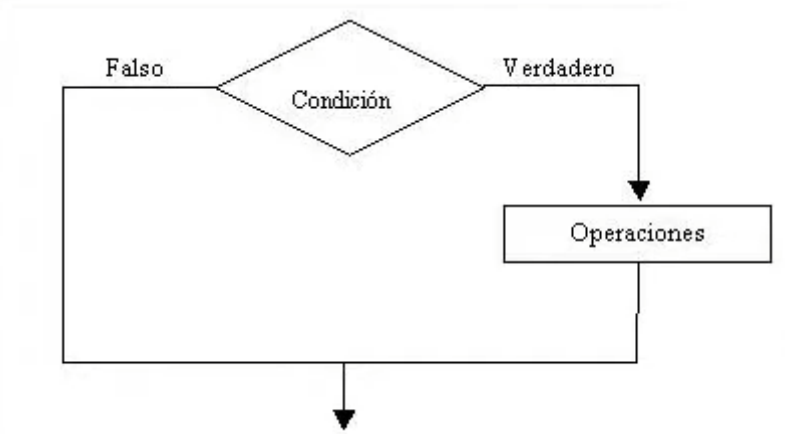
¿Elijo la dirección A o la dirección B?
¿Estudio el lenguaje C o el lenguaje Java?
¿Me pongo este pantalón?
Para ir al trabajo ¿elijo la ruta A o la ruta B?
Al cursar una carrera, ¿elijo el turno mañana, tarde o noche?

Por supuesto que en un problema se combinan estructuras secuenciales y condicionales.

Estructura condicional simple

Cuando se presenta la elección tenemos la opción de realizar una actividad o no realizar ninguna.

Representación gráfica:



Podemos observar: El rombo representa la condición. Hay dos opciones que se pueden tomar. Si la condición da verdadera se sigue el camino del verdadero, o sea el de la derecha, si la condición da falsa se sigue el camino de la izquierda.

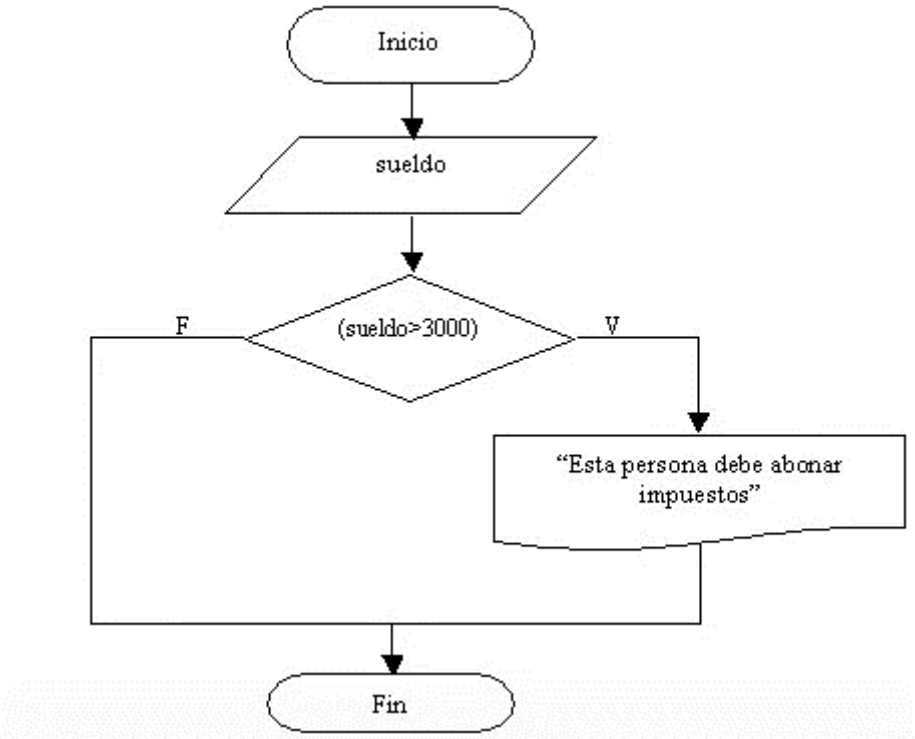
Se trata de una estructura CONDICIONAL SIMPLE porque por el camino de verdadero hay actividades y por el camino del falso no hay actividades.

Por el camino del verdadero pueden existir varias operaciones, entradas y salidas, inclusive ya veremos que puede haber otras estructuras condicionales.

Problema

Ingresar el sueldo de una persona, si supera los 3000 pesos mostrar un mensaje en pantalla indicando que debe abonar impuestos.

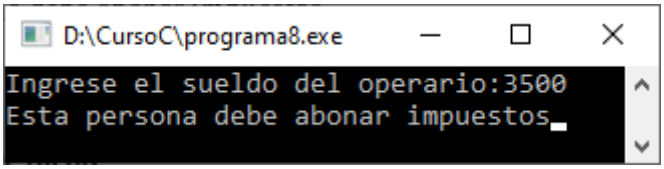
Diagrama de flujo:



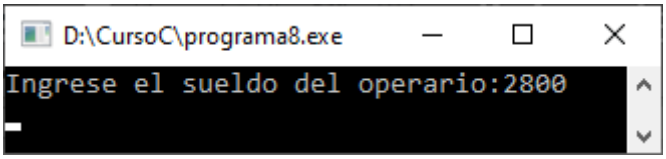
Podemos observar lo siguiente: Siempre se hace la carga del sueldo, pero si el sueldo que ingresamos supera los 3000 pesos se mostrará por pantalla el mensaje "Esta persona debe abonar impuestos", en caso que la persona cobre 3000 o menos no aparece nada en pantalla.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   int main()
5   {
6       float sueldo;
7       printf("Ingrese el sueldo del operario:");
8       scanf("%f", &sueldo);
9       if (sueldo>3000)
10      {
11          printf("Esta persona debe abonar impuestos");
12      }
13      getch();
14      return 0;
15  }
```

Vamos a ejecutar introduciendo un importe de 3500.



Ahora un importe de 2800.

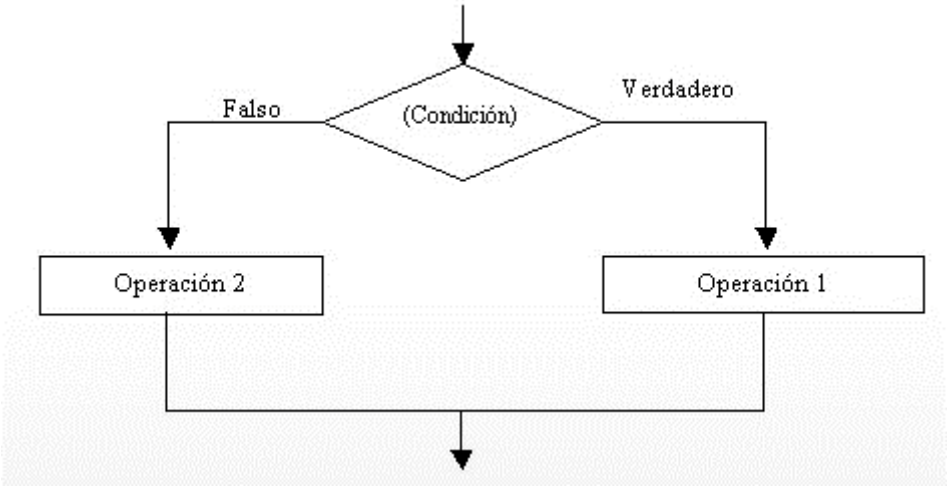


Capítulo 11.- Estructuras condicionales simples y compuestas – 2

Estructura condicional compuesta

Cuando se presenta la elección tenemos la opción de realizar una actividad u otra. Es decir tenemos actividades por el verdadero y el falso de la condición. Lo más importante que hay que tener en cuenta que se realizan las actividades de la rama del verdadero o las del falso, NUNCA se realizan las actividades de las dos ramas.

Representación gráfica:

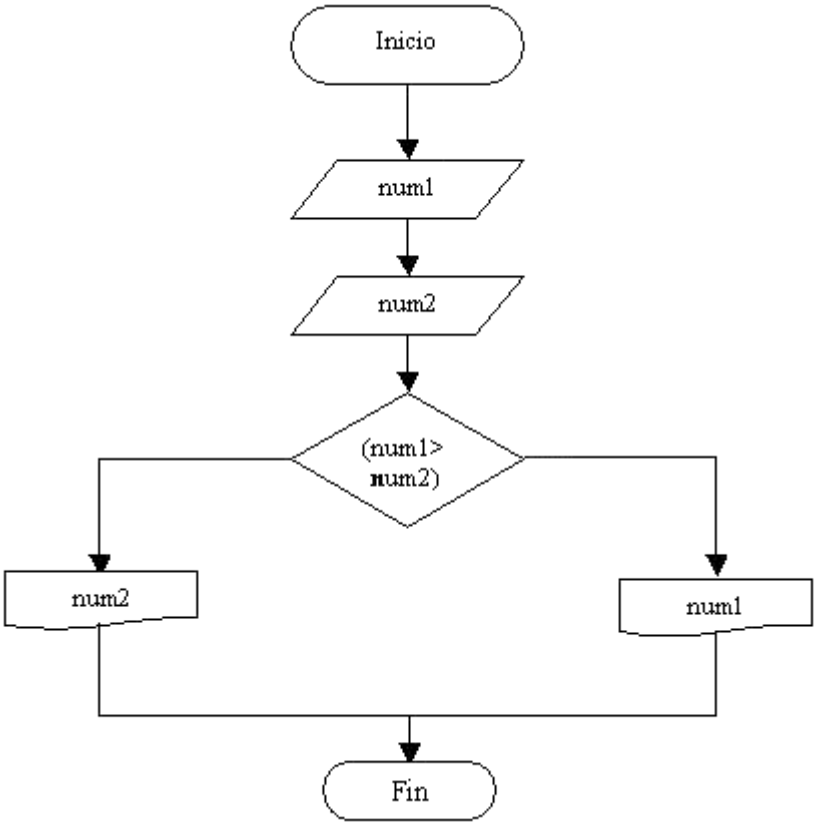


En una estructura condicional compuesta tenemos entradas, salidas, operaciones, tanto por la rama del verdadero como por la rama del falso.

Problema:

Realizar un programa que solicite al operador ingresar dos números y muestre en pantalla el mayor de ellos.

Diagrama de flujo:

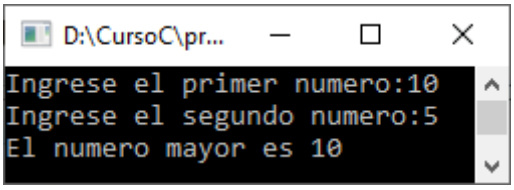


Se hace una entrada del num1 y el num2 por teclado. Para saber que variable tiene un valor mayor preguntamos si el contenido de num1 es mayor (>) que el contenido del num2, si la respuesta es verdadera vamos por la rama de la derecha e imprimimos num1, en caso que la condición será falsa vamos por la rama de la izquierda (Falsa) e imprimimos num2.

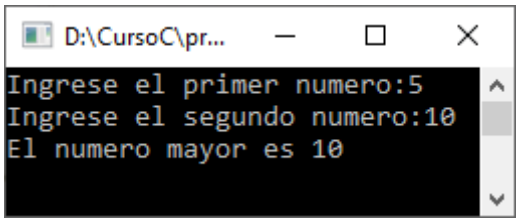
Estamos en la presencia de una ESTRUCTURA CONDICIONAL COMPUESTA ya que tenemos actividades por la rama del verdadero y del falso.

```
#include<stdio.h>
#include<conio.h>

int main()
{
    int num1, num2;
    printf("Ingrese el primer numero:");
    scanf("%i", &num1);
    printf("Ingrese el segundo numero:");
    scanf("%i", &num2);
    if (num1>num2)
    {
        printf("El numero mayor es %i", num1);
    }else
    {
        printf("El numero mayor es %i", num2);
    }
    getch();
    return 0;
}
```

Vamos a ejecutar introduciendo los valores de 10 y 5.



Ahora vamos a ejecutar introduciendo los valores 5 y 10.



Operadores

En al condición debe disponerse únicamente variables, valores contantes y operadores relacionales.

Operadores relacionales

>	(mayor)
<	(menor)
>=	(mayor o igual)
<=	(Menor o igual)
==	(igual)
!=	(distinto)

Operadores Matemáticos

+	(más)
-	(menos)
*	(producto)
/	(división)
%	(resto de la división) Ej.: x=13%5 {se guarda 3}

Hay que tener en cuenta que al disponer una condición debemos seleccionar que operador relacional se adapta a la pregunta.

Ejemplos:

Se ingresa un número multiplicarlo por 10 si es distinto a 0. (!=)
Se ingresan dos números mostrar una advertencia si son iguales (==)

Los problemas que se pueden presentar son infinitos y la correcta elección del operador sólo se alcanza con la práctica intensiva en la resolución de problemas.

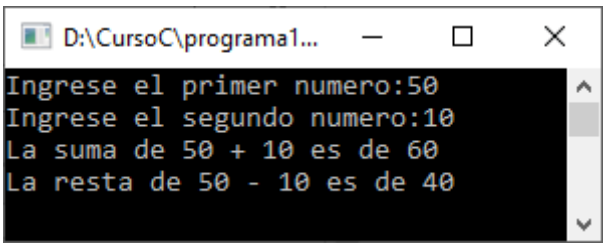
Capítulo 12.- Estructuras condicionales simples y compuestas – 3

Problema propuesto

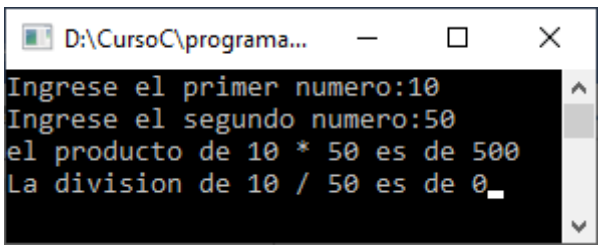
Realizar un programa que solicite la carga por teclado de dos números, si el primero es mayor al segundo informar su suma y diferencia, en caso contrario informar el producto y la división del primero con respecto al segundo.

```
#include<stdio.h>
#include<conio.h>

int main()
{
    int num1, num2,suma, resta, producto, division;;
    printf("Ingrese el primer numero:");
    scanf("%i", &num1);
     printf("Ingrese el segundo numero:");
    scanf("%i", &num2);
    if(num1>num2)
    {
        suma=num1+num2;
        resta=num1-num2;
        printf("La suma de %i + %i es de %i", num1, num2, suma);
        printf("\n");
        printf("La resta de %i - %i es de %i", num1, num2, resta);
    }
    else
    {
        producto=num1*num2;
        division=num1/num2;
        printf("el producto de %i * %i es de %i", num1, num2, producto);
        printf("\n");
        printf("La division de %i / %i es de %i", num1, num2, division);
    }
    getch();
    return 0;
}
```

Vamos a introducir los valores 50 y 10:



Ahora vamos a introducir los valores 10 y 50.



En la división muestra un 0 porque tiene un valor decimal y la variable int no los muestra.

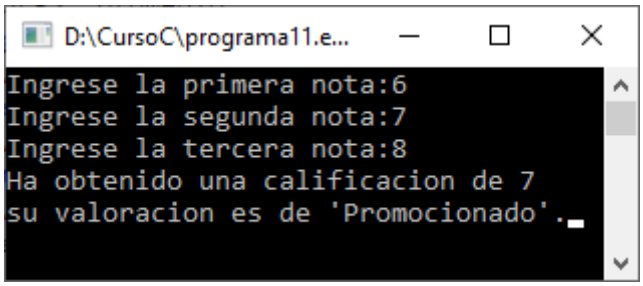
Capítulo 13.- Estructuras condicionales simples y compuestas – 4

Problema propuesto

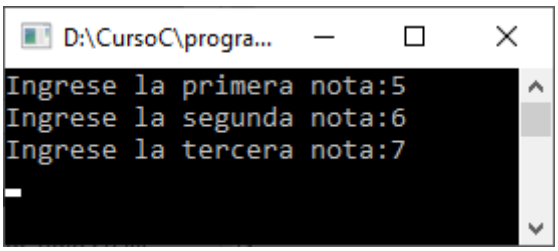
Se ingresan tres notas de un alumno, si el promedio es mayor o igual a siete mostrar un mensaje "Promocionado".

```
#include<stdio.h>
#include<conio.h>

int main()
{
    int nota1, nota2, nota3, promedio;
    printf("Ingrese la primera nota:");
    scanf("%i", &nota1);
    printf("Ingrese la segunda nota:");
    scanf("%i", &nota2);
    printf("Ingrese la tercera nota:");
    scanf("%i", &nota3);
    promedio=(nota1+nota2+nota3)/3;
    if (promedio>=7)
    {
        printf("Ha obtenido una calificacion de %i", promedio);
        printf("\nsu valoracion es de 'Promocionado'.");
    }
    getch();
    return 0;
}

```

Vamos a ejecutar introduciendo las notas 6, 7 y 8.



Ejecutamos de nuevo introduciendo las notas 5, 6 y 7.



Capítulo 14.- Estructuras condicionales simples y compuestas – 5

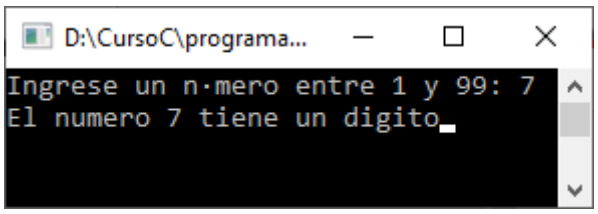
Problema propuesto

Se ingresa por teclado un número positivo de uno o dos dígitos (1..99) mostrar un mensaje indicando si el número tiene uno o dos dígitos.

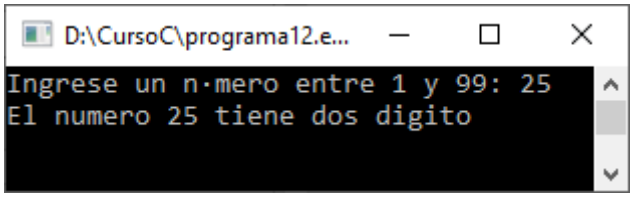
(Tener en cuenta que condición debe cumplirse para tener dos dígitos un número entero)

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   int main()
5   {
6       int numero;
7       printf("Ingrese un número entre 1 y 99: ");
8       scanf("%i", &numero);
9       if (numero<10)
10      {
11          printf("El numero %i tiene un digito", numero);
12      }
13      else
14      {
15          printf("El numero %i tiene dos digito", numero);
16      }
17      getch();
18      return 0;
19  }
```

Vamos a ejecutar introduciendo el valor 7.



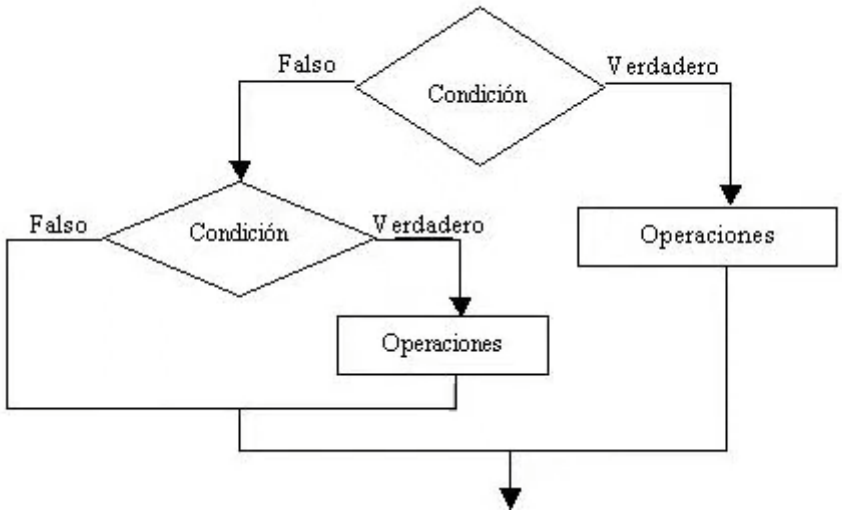
Vamos a ejecutar de nuevo introduciendo el valor 25.



Capítulo 15.- Estructuras condicionales anidadas – 1

Decimos que una estructura condicional es anidada cuando por la rama del verdadero o falso de una estructura condicional hay otra estructura condicional.

Representación gráfica:



El diagrama de flujo que se presenta contiene dos estructuras condicionales. La principal se trata de una estructura condicional compuesta y la segunda es una estructura condicional simple y está contenida por la rama del falso de la primera estructura.

Es común que se presenten estructuras condicionales anidadas aún más complejas.

Problema

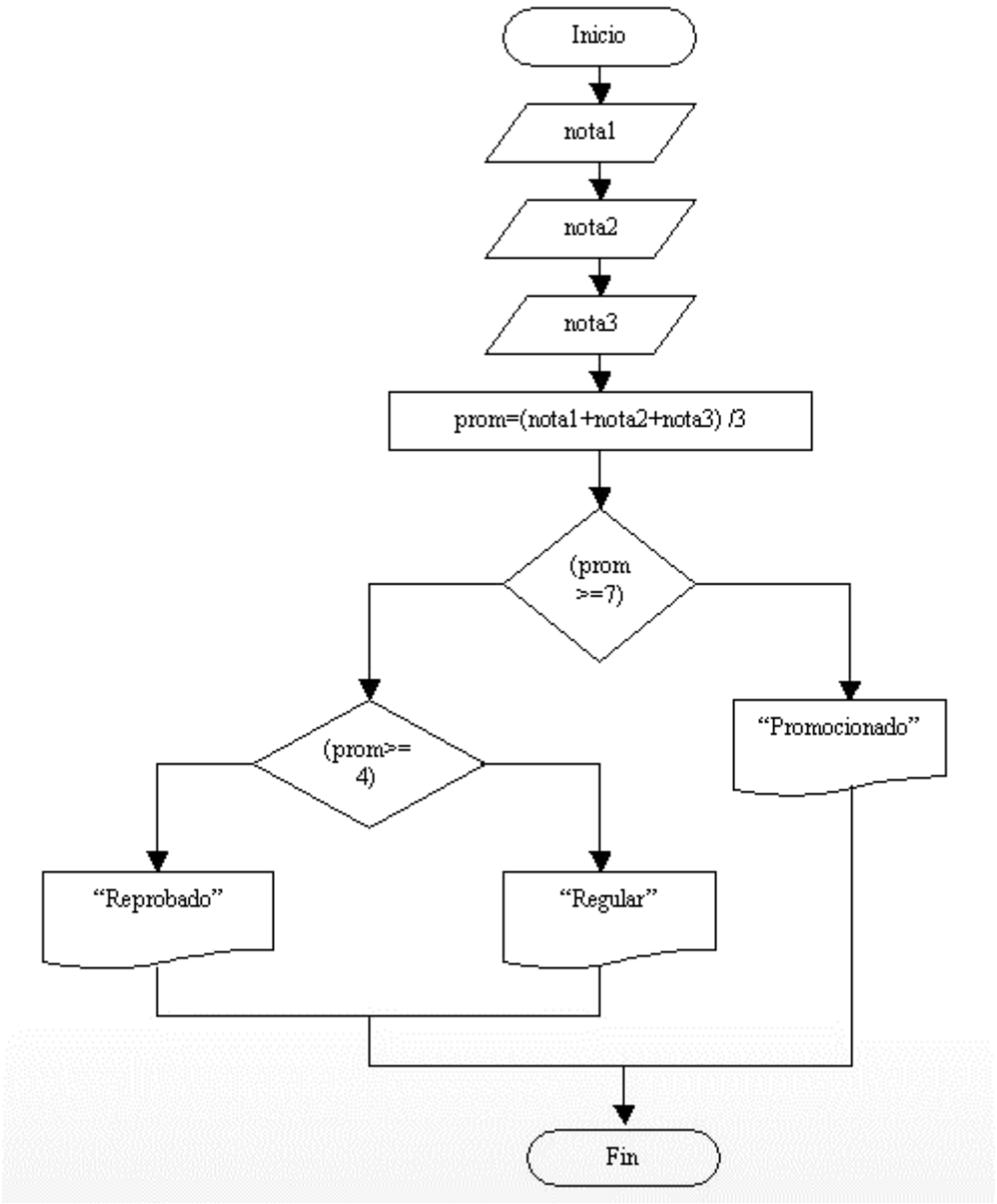
Confeccionar un programa que pida por teclado tres notas de un alumno, calcule el promedio e imprima alguno de estos mensajes:

Si el promedio es >=7 mostrar "Promocionado".

Si el promedio es >=4 y <7 mostrar "Regular".

Si el promedio es <4 mostrar "Reprobado".

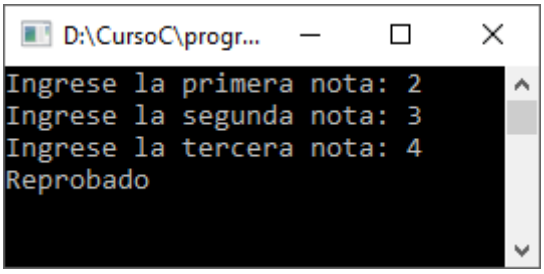
Diagrama de flujo:



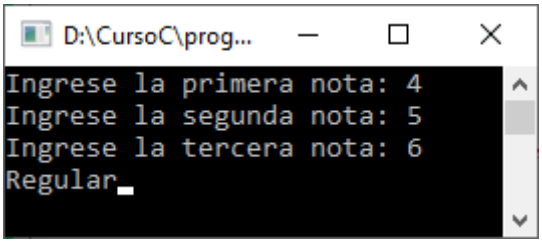

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int nota1, nota2, nota3, prom;
    printf("Ingrese la primera nota: ");
    scanf("%i", &nota1);
    printf("Ingrese la segunda nota: ");
    scanf("%i", &nota2);
    printf("Ingrese la tercera nota: ");
    scanf("%i", &nota3);
    prom=(nota1+nota2+nota3)/3;
```

```
14      if(prom>=7)
15      {
16          printf("Promocionado");
17      }
18      else
19      {
20          if(prom>=4){
21              printf("Regular");
22          }
23          else
24          {
25              printf("Reprobado");
26          }
27      }
28      getch();
29      return 0;
30  }
```

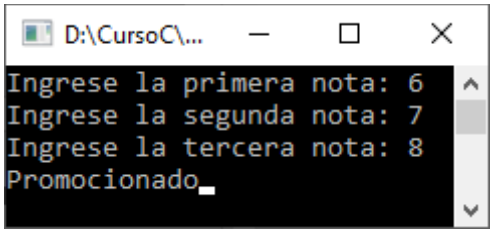
Vamos a ejecutar ingresando las notas 2, 3 y 4.



Vamos a ejecutar de nuevo ingresando las notas 4, 5 y 6.



Vamos a ejecutar de nuevo ingresando las notas 6, 7 y 8.



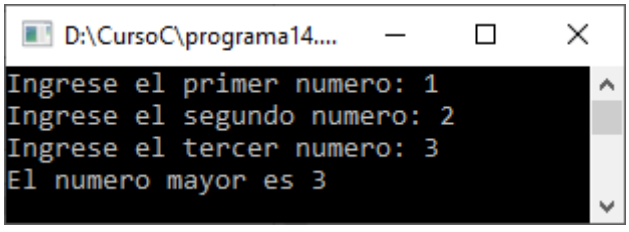
Capítulo 16.- Estructuras condicionales anidadas – 2

Problema propuesto

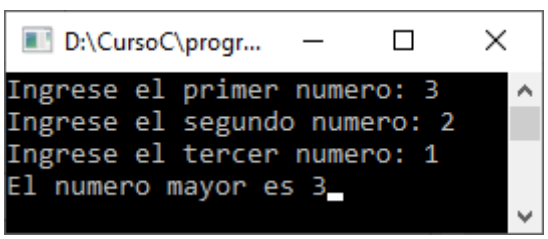
Se cargan por teclado tres números distintos. Mostrar por pantalla el mayor de ellos.

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int num1, num2, num3;
    printf("Ingrese el primer numero: ");
    scanf("%i", &num1);
    printf("Ingrese el segundo numero: ");
    scanf("%i", &num2);
    printf("Ingrese el tercer numero: ");
    scanf("%i", &num3);
    if(num1>num2)
    {
        if(num1>num3){
            printf("El numero mayor es %i", num1);
        }
    }
    else{
        if (num2>num3)
        {
            printf("El numero mayor es %i", num2);
        }
        else
        {
            printf("El numero mayor es %i", num3);
        }
    }
    getch();
    return 0;
}
```

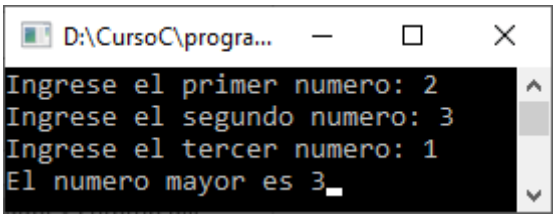
Vamos a ejecutar introduciendo los valores 1, 2 y 3.



Ejecutamos de nuevo introduciendo los valore 3, 2 y 1.



Ejecutamos de nuevo introduciendo los valores 2, 3 y 1.



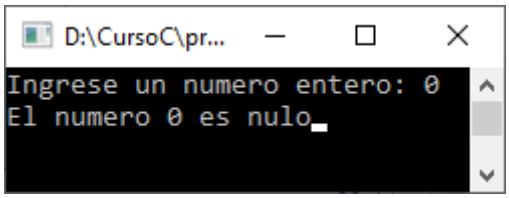
Capítulo 17.- Estructuras condicionales anidadas – 3

Problema propuesto

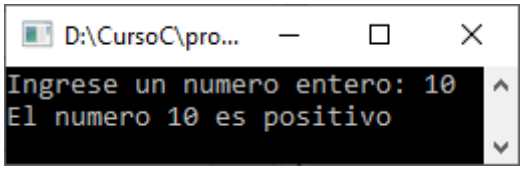
Se ingresan por teclado un valor entero, mostrar una leyenda que indique si el número es positivo, negativo o nulo (es decir cero)

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   int main()
5   {
6       int numero;
7       printf("Ingrese un numero entero: ");
8       scanf("%i", &numero);
9       if(numero==0)
10      {
11          printf("El numero %i es nulo", numero);
12      }
13      else
14      {
15          if (numero>0)
16          {
17              printf("El numero %i es positivo", numero);
18          }
19          else
20          {
21              printf("El numero %i es negativo", numero);
22          }
23      }
24      getch();
25      return 0;
26  }
```

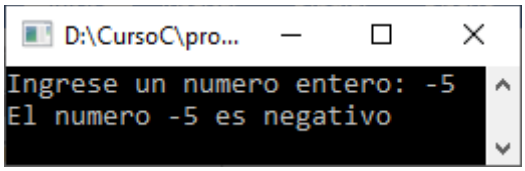
Vamos a ejecutar introduciendo el valor 0.



Ejecutamos de nuevo introduciendo el valor 10.



Ejecutamos de nuevo introduciendo el valor -5.



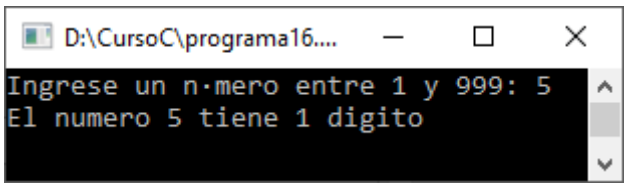
Capítulo 18.- Estructuras condicionales anidadas – 4

Problema propuesto

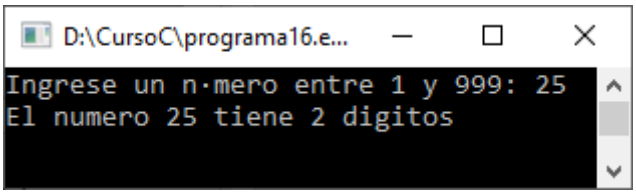
Confeccionar un programa que permita cargar un número entero positivo de hasta tres cifras y muestre un mensaje indicando si tiene 1, 2 o 3 cifras. Mostrar un error si el número de cifras es mayor.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   int main()
5   {
6       int numero;
7       printf("Ingrese un número entre 1 y 999: ");
8       scanf("%i", &numero);
9       if(numero<10){
10          printf("El numero %i tiene 1 digito", numero);
11      }
12      else
13      {
14          if(numero<100)
15          {
16              printf("El numero %i tiene 2 digitos", numero);
17          }
18          else{
19              if(numero<1000)
20              {
21                  printf("El numero %i tiene 3 digitos", numero);
22              }
23              else{
24                  printf("Error en la entrada de datos");
25              }
26          }
27      }
28      getch();
29      return 0;
30  }
```

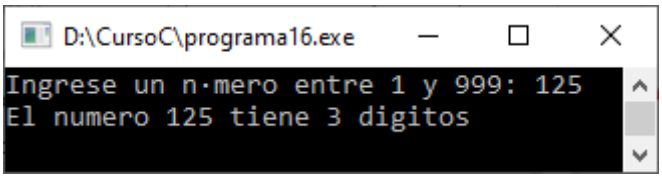
Ejecutamos introduciendo el valor 5:



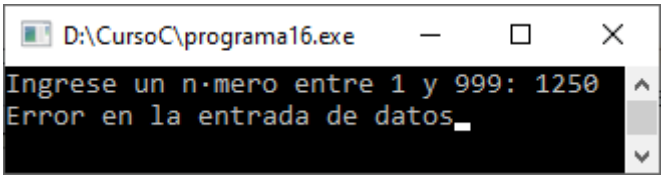
Ejecutamos introduciendo el valor 25:



Ejecutamos introduciendo el valor 125:



Ejecutamos por última vez introduciendo el valor 1250:



Capítulo 19.- Estructuras condicionales anidadas – 5

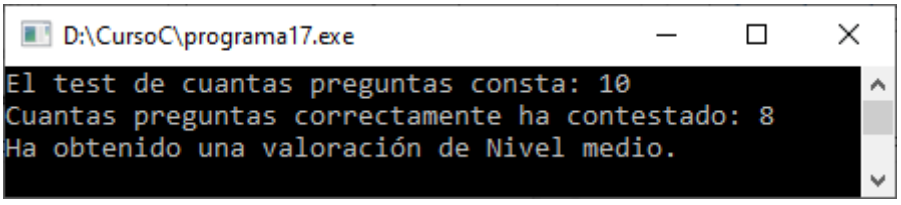
Un postulante a un empleo, realiza un test de capacitación, se obtuvo la siguiente información:

Cantidad total de preguntas que se le realizaron y la cantidad de preguntas que contestó correctamente. Se pide confeccionar un programa que ingrese los dos datos por teclado e informe el nivel del mismo según el porcentaje de respuestas correctas que ha obtenido y sabiendo que:

Nivel máximo:	Porcentaje>=90%.
Nivel medio:	Porcentaje>=75% y <90%.
Nivel regular:	Porcentaje>=50% y <75%.
Fuera de nivel:	Porcentaje <50%.

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int totPreguntas, totPregunContestadas, porcentaje;
    printf("El test de cuantas preguntas consta: ");
    scanf("%i", &totPreguntas);
    printf("Cuantas preguntas correctamente ha contestado: ");
    scanf("%i", &totPregunContestadas);
    porcentaje=totPregunContestadas*100/totPreguntas;
    if (porcentaje>=90){
        printf("Ha obtenido una valoración de Nivel maximo.");
    }
    else
    {
        if (porcentaje>=75)
        {
            printf("Ha obtenido una valoración de Nivel medio.");
        }
        else
        {
            if (porcentaje>=50)
            {
                printf("Ha obtenido una valoración de Nivel regular.");
            }
            else
            {
                printf("Ha obtenido una valoración de Fuera de nivel.");
            }
        }
    }
    getch();
    return 0;
}
```

Vamos a ejecutar:



Capítulo 20.- Condiciones compuestas con operadores lógicos – 1

Hasta ahora hemos visto los operadores:

relacionales (>, <, >0, <=, ==, !=)

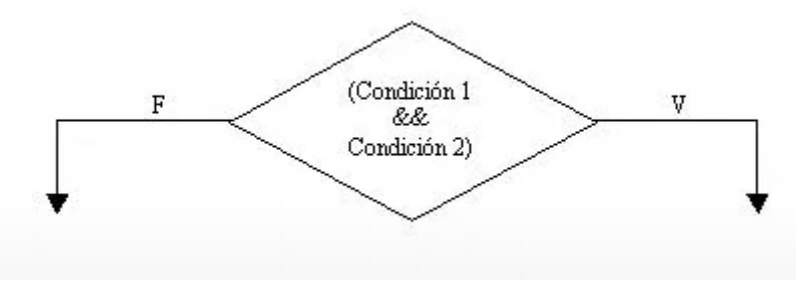
matemáticos (+, -, *, /, %)

Pero nos están faltando otros operadores imprescindible:

lógicos (&&, ||).

Estos dos operadores se emplean fundamentalmente en las estructuras condicionales para agrupar varias condiciones simples.

Operador &&



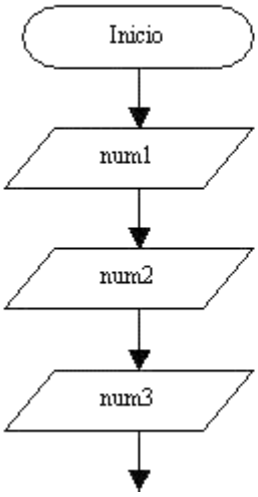
Traducido se lo lee como "Y". Si la condición 1 es verdadera Y la condición 2 es verdadera luego ejecutar la rama del verdadero.

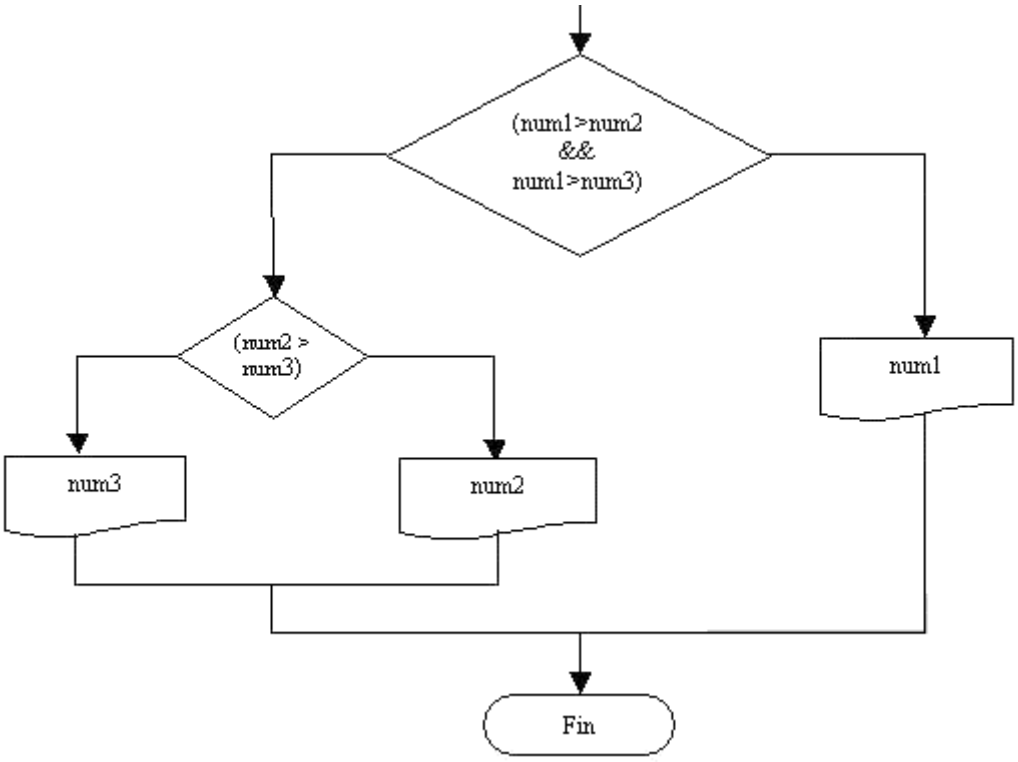
Cuando vinculamos dos condiciones con el operador "&&", las dos condiciones deben ser verdaderas para que el resultado de la condición puesta sea Verdadero y continúe por la rama del verdadero de la estructura condicional.

La utilización de operadores lógicos permiten en muchos casos plantear algoritmos más cortos y comprensibles.

Problema:

Confeccionar un programa que lea por teclado tres números distintos y nos muestre el mayor.





Este ejercicio está resuelto sin emplear operadores lógicos en un concepto anterior del tutorial.

La primera estructura condicional es una ESTRUCTURA CONDICIONAL COMPUESTA con una CONDICIÓN COMPUESTA.

Podemos leerla de la siguiente forma:

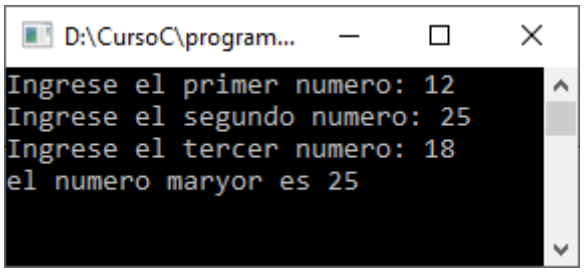
Si el contenido de la variable num1 es mayor al contenido de la variable num2 y si el contenido de la variable num1 es mayor al contenido de la variable num3 entontes la CONDICIÓN COMPUESTA resulta Verdadera.

Si una de las condiciones simples da falso la CONDICIÓN COMPUESTA da Falso y continua por la rama del falso.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   int main()
5   {
6       int num1, num2, num3;
7       printf("Ingrese el primer numero: ");
8       scanf("%i", &num1);
9       printf("Ingrese el segundo numero: ");
10      scanf("%i", &num2);
11      printf("Ingrese el tercer numero: ");
12      scanf("%i", &num3);
13      if (num1>num2 && num1>num3)
14      {
15          printf("el numero mayor es %i", num1);
16      }
17      else{
```

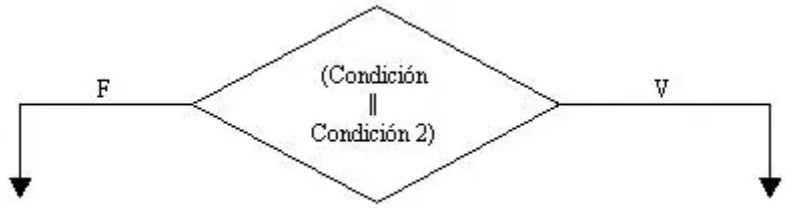
```
18          if (num2>num3)
19          {
20              printf("el numero maryor es %i", num2);
21          }
22          else{
23              printf("el numero maryor es %i", num3);
24          }
25      }
26      getch();
27      return 0;
28  }
```

Si ejecutamos este será el resultado:



Capítulo 21.- Condiciones compuestas con operadores lógicos – 2

Operador ||



Traducido se lo lee como "O". Si la condición 1 es Verdadera o la condición 2 es Verdadera, luego ejecutar la rama del Verdadero.

Cuando vinculamos dos o más condiciones con el operador "Or" que en C se escribe con los dos caracteres ||, con que una de las dos condiciones sea Verdadera alcanza para que el resultado de la condición compuesta sea Verdadero.

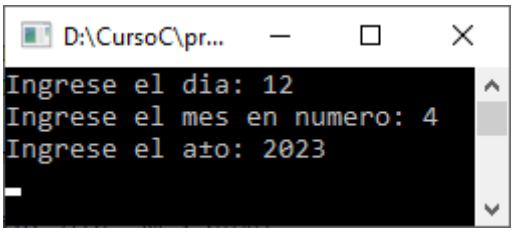
Problema:

Se carga una fecha (día, mes y año) por teclado. Mostrar un mensaje si corresponde al primes trimestre del año (enero, febrero o marzo) Cargar por teclado el valor numérico del día, mes y año.

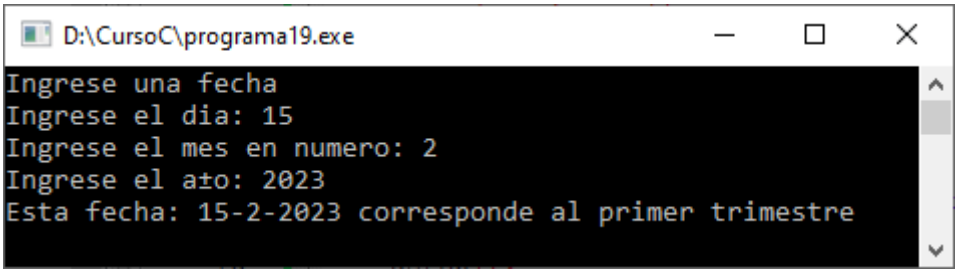
Ejemplo: dia:10 mes:2 año:2020.

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int dia, mes,anyo;
    printf("Ingrese una fecha \n");
    printf("Ingrese el dia: ");
    scanf("%i", &dia);
    printf("Ingrese el mes en numero: ");
    scanf("%i", &mes);
    printf("Ingrese el año: ");
    scanf("%i", &anyo);
    if(mes==1 || mes==2 || mes==3)
    {
        printf("Esta fecha: %i-%i-%i ", dia, mes, anyo);
        printf("corresponde al primer trimestre");
    }
    getch();
    return 0;
}
```

Vamos a ejecutar introduciendo la fecha 12 de Marzo (4) de 2023:



Ahora vamos a introducir la fecha 15 de Febrero (2) de 2023.



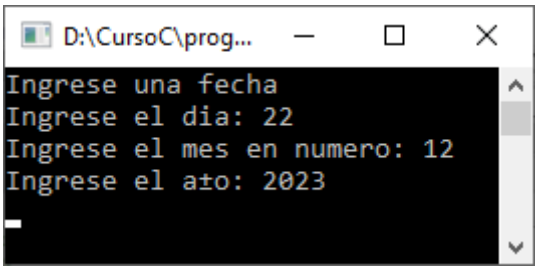
Capítulo 22.- Condiciones compuestas con operadores lógicos – 3

Problema propuesto

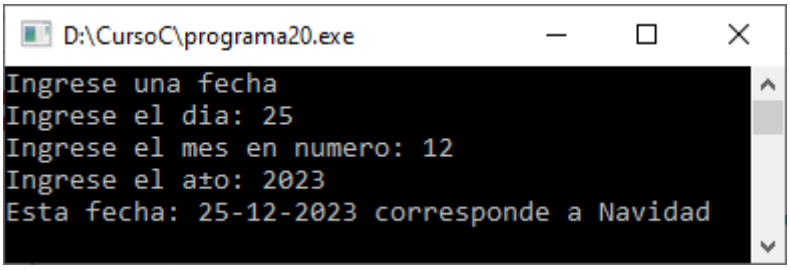
Realiza un programa que pida cargar una fecha cualquiera, luego verificar si dicha fecha corresponde a Navidad.

```
#include<stdio.h>
#include<conio.h>

int main()
{
    int dia, mes,anyo;
    printf("Ingrese una fecha \n");
    printf("Ingrese el dia: ");
    scanf("%i", &dia);
    printf("Ingrese el mes en numero: ");
    scanf("%i", &mes);
    printf("Ingrese el año: ");
    scanf("%i", &anyo);
    if(dia==25 && mes==12)
    {
        printf("Esta fecha: %i-%i-%i ", dia, mes, anyo);
        printf("corresponde a Navidad");
    }
    getch();
    return 0;
}
```

Vamos a ingresar una fecha que no corresponde con Navidad:



Ejecutamos de nuevo ahora si ingresaremos una fecha que corresponde a Navidad:



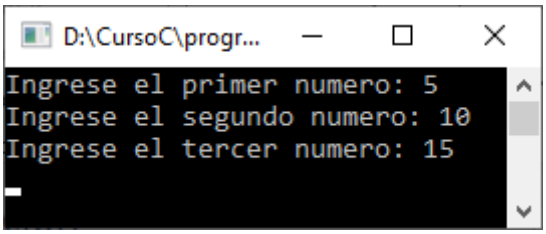
Capítulo 23.- Condiciones compuestas con operadores lógicos – 4

Problema propuesto

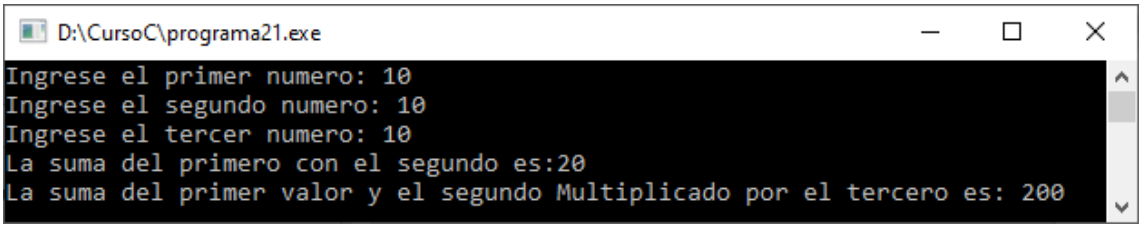
Se ingresan tres valores por teclado, si todos son iguales se imprime la suma del primero con el segundo y a este resultado se lo multiplica por el tercero.

```
#include<stdio.h>
#include<conio.h>

int main()
{
    int num1, num2, num3, suma, producto;
    printf("Ingrese el primer numero: ");
    scanf("%i", &num1);
    printf("Ingrese el segundo numero: ");
    scanf("%i", &num2);
    printf("Ingrese el tercer numero: ");
    scanf("%i", &num3);
    if (num1==num2 && num1==num3)
    {
        suma=num1+num2;
        printf("La suma del primero con el segundo es:");
        printf("%i", suma);
        printf("\n");
        producto=suma*num3;
        printf("La suma del primer valor y el segundo ");
        printf("Multiplicado por el tercero es: ");
        printf("%i", producto);
    }
    getch();
    return 0;
}
```

Ejecutamos introduciendo los valores 5, 10 y 15.



Ejecutamos de nuevo introduciendo los valores 10, 10 y 10.



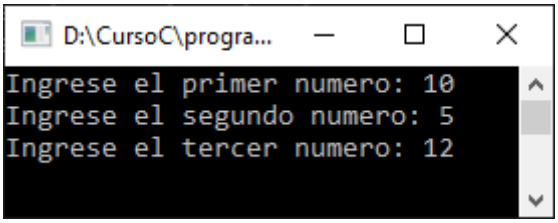
Capítulo 24.- Condiciones compuestas con operadores lógicos – 5

Problema propuesto

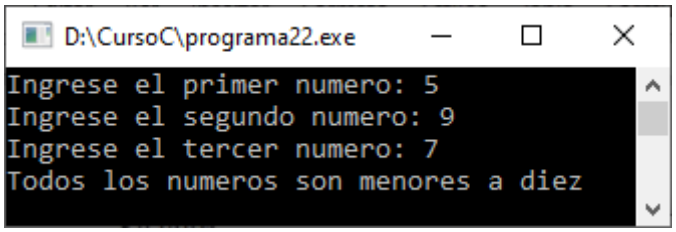
Se ingresan por teclado tres números, si todos los valores ingresados son menores a 10, imprimir en pantalla la leyenda "Todos los números son menores a diez".

```
#include<stdio.h>
#include<conio.h>

int main()
{
    int num1, num2, num3, suma, producto;
    printf("Ingrese el primer numero: ");
    scanf("%i", &num1);
    printf("Ingrese el segundo numero: ");
    scanf("%i", &num2);
    printf("Ingrese el tercer numero: ");
    scanf("%i", &num3);
    if (num1<10 && num2<10 && num3<10)
    {
        printf("Todos los numeros son menores a diez");
    }
    getch();
    return 0;
}
```

Vamos a ejecutar e introducir los valores 10, 5, 12.



Vamos a ejecutar de nuevo con todos los valores menores de 10.



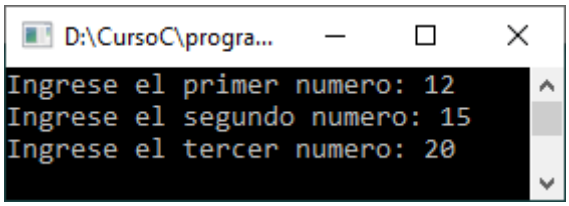
Capítulo 25.- Condiciones compuestas con operadores lógicos – 6

Problema propuesto

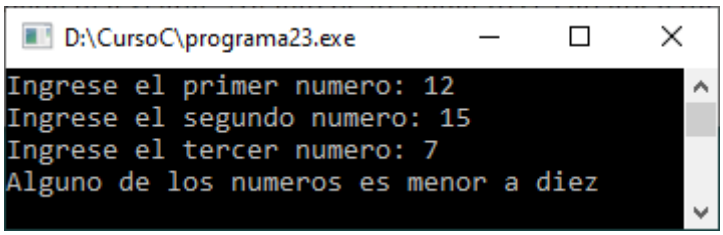
Se ingresan por teclado tres números, si al menor uno de los valores ingresados es menor a 10, imprimir en pantalla la leyenda "Alguno de los números es menor a diez".

```
#include<stdio.h>
#include<conio.h>

int main()
{
    int num1, num2, num3, suma, producto;
    printf("Ingrese el primer numero: ");
    scanf("%i", &num1);
    printf("Ingrese el segundo numero: ");
    scanf("%i", &num2);
    printf("Ingrese el tercer numero: ");
    scanf("%i", &num3);
    if (num1<10 || num2<10 || num3<10)
    {
        printf("Alguno de los numeros es menor a diez");
    }
    getch();
    return 0;
}
```

Vamos a ejecutar introduciendo los valores 12, 15 y 20.



Ejecutamos de nuevo introduciendo los valores 12, 15 y 7.

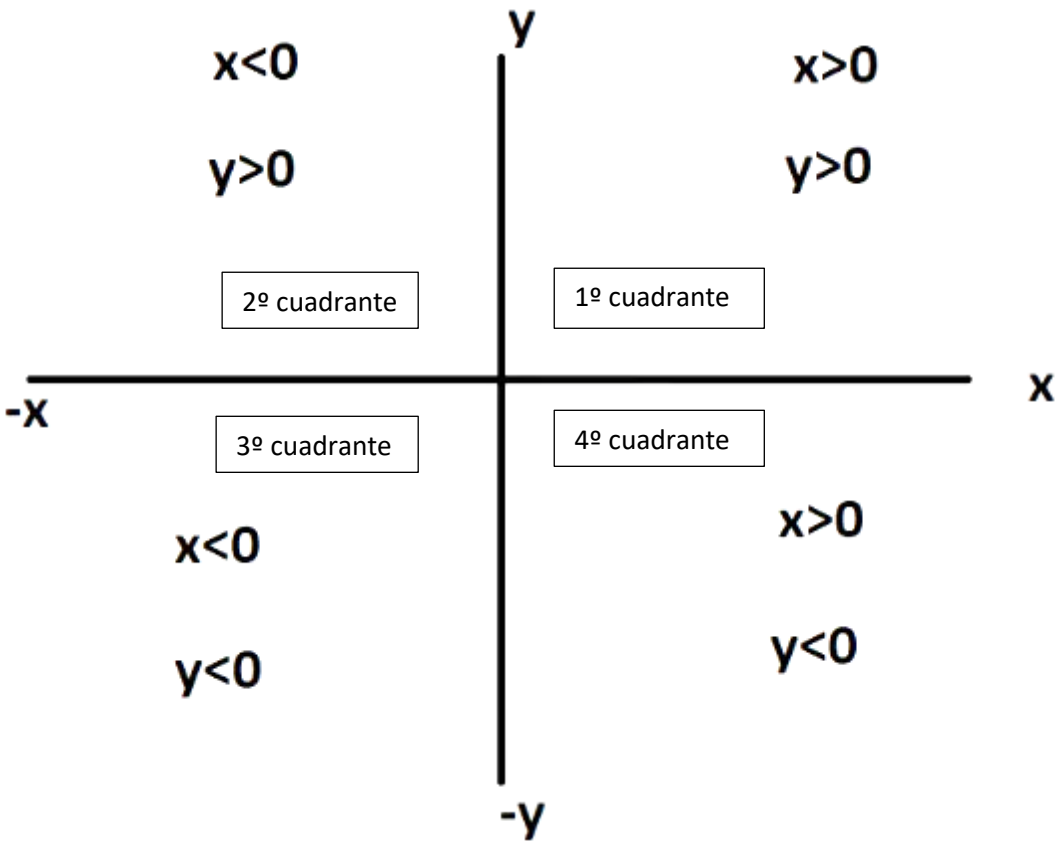


Capítulo 26.- Condiciones compuestas con operadores lógicos – 7

Problema propuesto

escribir un programa que pida ingresar la coordenada de un punto en el plano, es decir dos valores enteros x e y (distintos a cero).

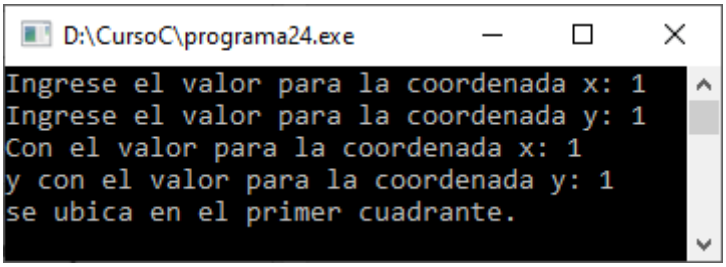
Posteriormente imprimir en pantalla en que cuadrante se ubica dicho punto (1º Cuadrante si x>0 Y y>0, 2º Cuadrante: x<0 Y y>0, etc.)



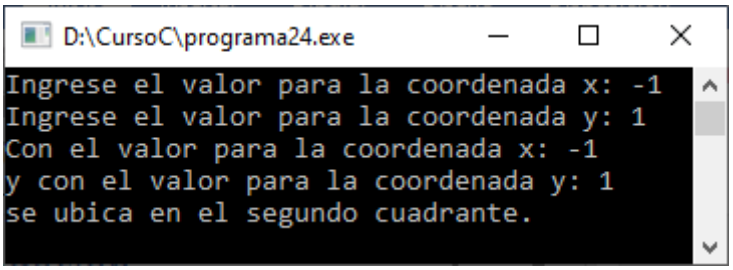

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int x, y;
    printf("Ingrese el valor para la coordenada x: ");
    scanf("%i", &x);
    printf("Ingrese el valor para la coordenada y: ");
    scanf("%i", &y);
    if(x>0 && y>0)
    {
        printf("Con el valor para la coordenada x: %i", x);
        printf("\ny con el valor para la coordenada y: %i", y);
        printf("\nse ubica en el primer cuadrante.");
    }else
    {
        if(x<0 && y>0)
        {
            printf("Con el valor para la coordenada x: %i", x);
            printf("\ny con el valor para la coordenada y: %i", y);
            printf("\nse ubica en el segundo cuadrante.");
```

```
23          }
24          else
25          {
26              if (x<0 && y<0)
27              {
28                  printf("Con el valor para la coordenada x: %i", x);
29                  printf("\ny con el valor para la coordenada y: %i", y);
30                  printf("\nse ubica en el tercer cuadrante.");
31              }
32              else
33              {
34                  if(x>0 && y<0)
35                  {
36                      printf("Con el valor para la coordenada x: %i", x);
37                      printf("\ny con el valor para la coordenada y: %i", y);
38                      printf("\nse ubica en el cuarto cuadrante.");
39                  }
40                  else
41                  {
42                      printf("Se encuentra sobre un eje.");
43                  }
44              }
45          }
46      }
47      getch();
48      return 0;
49  }
```

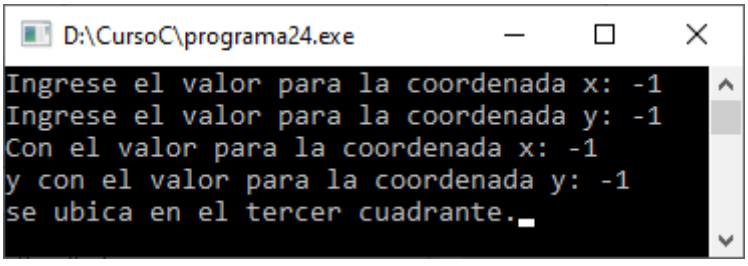
Vamos a ejecutar con las coordenadas x=1 Y y=1.



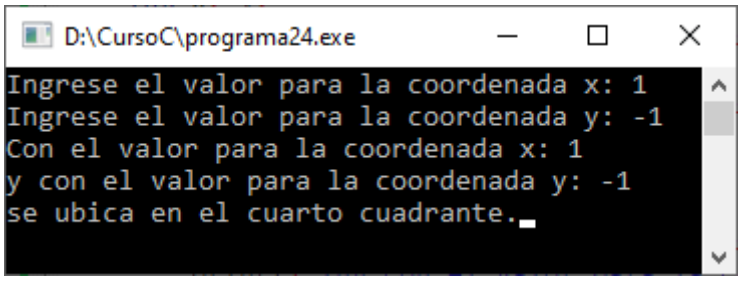
Vamos a ejecutar con las coordenadas x=-1 Y y=1.



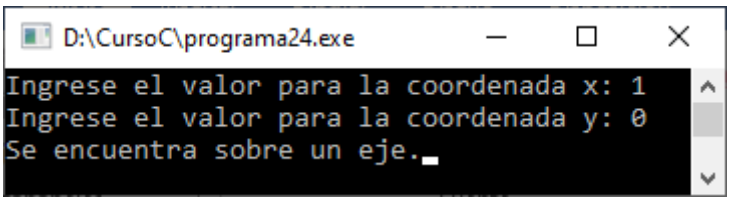
Vamos a ejecutar con las coordenadas x=-1 Y y=-1.



Vamos a ejecutar con las coordenadas x=1 Y y=-1.



Ahora en la x o en la y pondremos el valor 0.



Capítulo 27.- Condiciones compuestas con operadores lógicos – 8

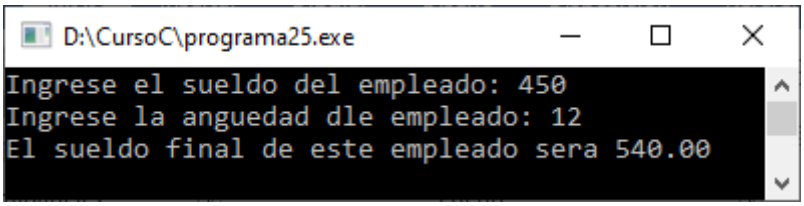
Problema propuesto

De un operario se conoce su sueldo y los años de antigüedad. Se pide confeccionar un programa que lea los datos de entrada e informe:

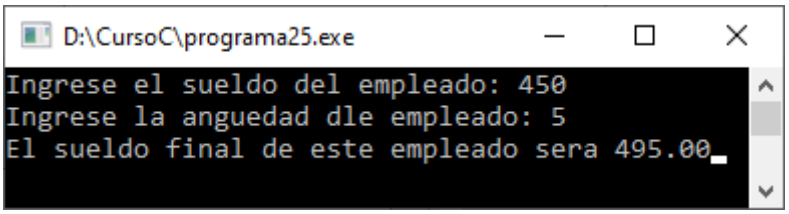
a) Si el sueldo es inferior a 500 y su antigüedad es igual o superior a 10 años, otorgarle un aumento del 20%, mostrar el sueldo a pagar.
b) Si el sueldo es inferior a 500 pero su antigüedad es menor a 10 años, otorgarle un aumento del 5%.
c) Si el sueldo es mayor o igual a 500 mostrar el sueldo en pantalla sin cambios.

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    float sueldo, sueldoFinal;
    int antiguedad;

    printf("Ingrese el sueldo del empleado: ");
    scanf("%f", &sueldo);
    printf("Ingrese la anguedad dle empleado: ");
    scanf("%i", &antiguedad);
    if (sueldo<500 && antiguedad>=10)
    {
        sueldoFinal=sueldo+(sueldo*20/100);
        printf("El sueldo final de este empleado sera %.2f", sueldoFinal);
    }
    else
    {
        if (sueldo<500)
        {
            sueldoFinal=sueldo+(sueldo*10/100);
            printf("El sueldo final de este empleado sera %.2f", sueldoFinal);
        }
        else
        {
            printf("El sueldo final de este empleado sera %.2f", sueldo);
        }
    }
    getch();
    return 0;
}
```

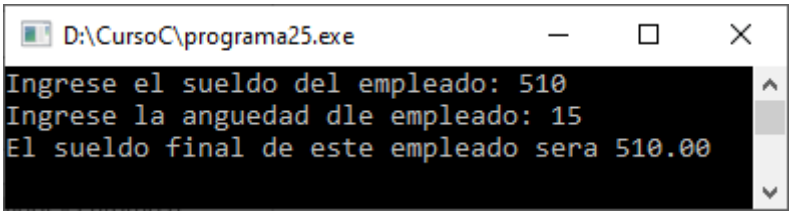
Vamos a ejecutar con un sueldo de 450 y una antigüedad de 12 años.



Vamos a ejecutar de nuevo con un sueldo de 450 y una antigüedad 5 años.



Por último un sueldo de 510 y una antigüedad de 15 años.



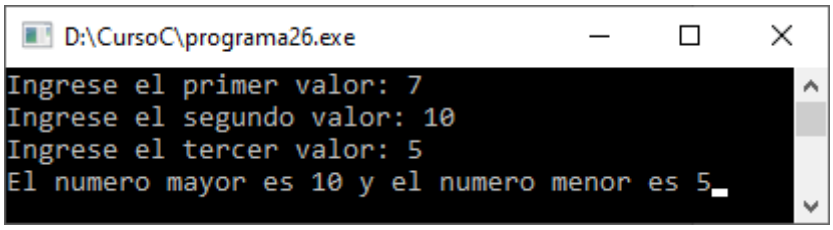
Capítulo 28.- Condiciones compuestas con operadores lógicos – 9

Problema propuesto

Escribir un programa en el cual: dada una lista de tres valores numéricos distintos se calcule e informe su rango de variación (debe mostrar el mayor y menor de ellos).

```
#include<stdio.h>
#include<conio.h>

int main()
{
    int num1, num2, num3, max, min;
    printf("Ingrese el primer valor: ");
    scanf("%i", &num1);
    printf("Ingrese el segundo valor: ");
    scanf("%i", &num2);
    printf("Ingrese el tercer valor: ");
    scanf("%i", &num3);
    if(num1>num2 && num1>num3){
        max=num1;
    }
    else
    {
        if(num2>num3)
        {
            max=num2;
        }
        else
        {
            max=num3;
        }
    }
     if(num1<num2 && num1<num3){
        min=num1;
    }
    else
    {
        if(num2<num3)
        {
            min=num2;
        }
        else
        {
            min=num3;
        }
    }
    printf("El numero mayor es %i y el numero menor es %i", max, min);
    getch();
    return 0;
}

```

Vamos a ejecutar:



Capítulo 29.- Estructura repetitiva while – 1

Hasta ahora hemos empleado estructuras SECUENCIALES y CONDICIONALES. Existe otro tipo de estructuras tan importantes como las anteriores que son las estructuras REPETITIVAS.

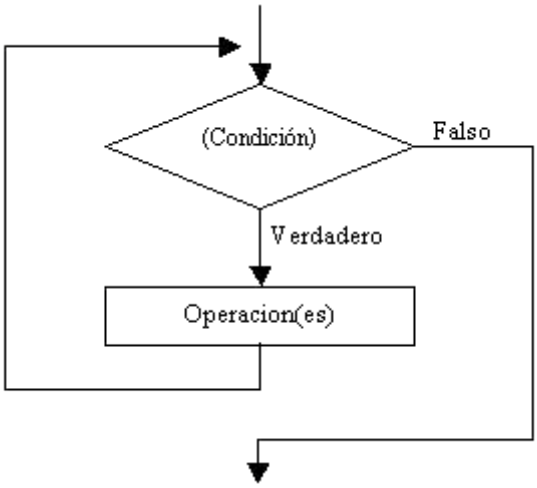
una estructura repetitiva permite ejecutar una instrucción o un conjunto de instrucciones varias veces.

Una ejecución repetitiva de sentencias se caracteriza por:

- La o las sentencias se repiten.
- El test o prueba de condición antes de cada repetición, que motivará que se repitan o no las sentencias.

Estructura repetitiva while

Representación gráfica de la estructura while:



No debemos confundir la representación gráfica de la estructura repetitiva while (Mientras) con la estructura condicional if (Si).

Funcionamiento: En primer lugar se verifica la condición, si la misma resulta verdadera se ejecutan las operaciones que indicamos en la rama Verdadero.

A la rama de verdadero la dibujamos en la parte inferior de la estructura repetitiva.

En caso que la condición sea Falsa continuará por la rama de Falso y sale de la estructura repetitiva para continuar con la ejecución del algoritmo.

El bloque se repite MIENTRAS la condición sea Verdadera.

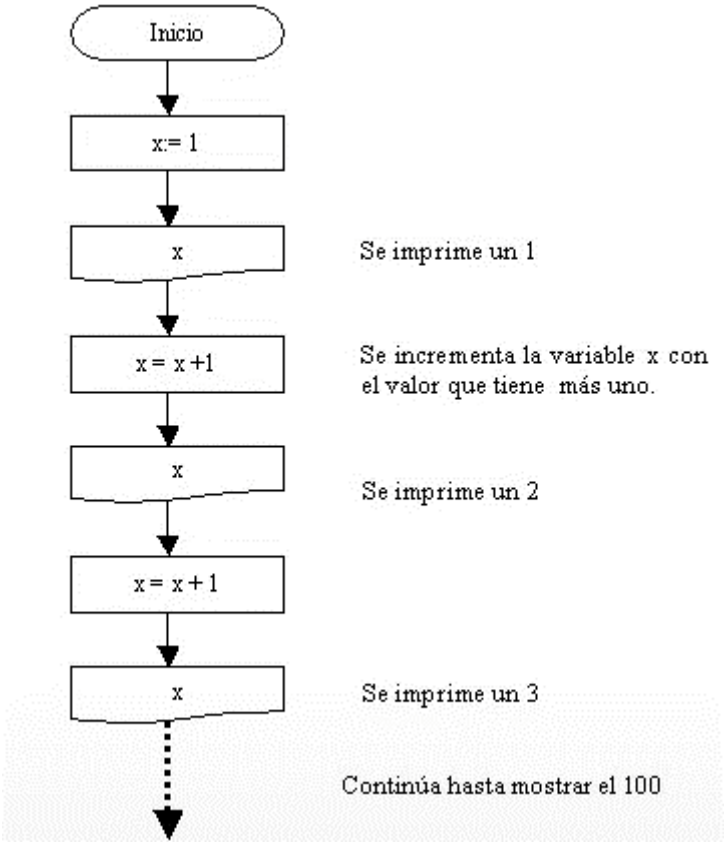
Importante: Si la condición siempre retorna verdadero estamos en presencia de un ciclo repetitivo infinito. Dicha situación es un error de programación, nunca finalizará el programa.

Problema:

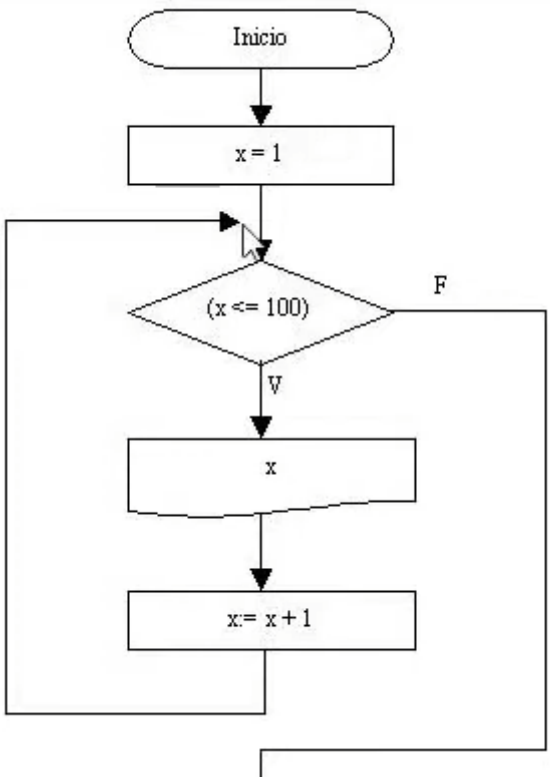
Realizar un programa que imprima en pantalla los números del 1 al 100.

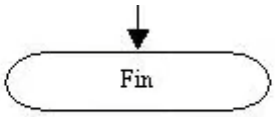
Sin conocer las estructuras repetitivas podemos resolver el problema empleando una estructura secuencial. Inicializamos una variable con el valor 1, luego imprimimos la variable, incrementamos nuevamente la variable y así sucesivamente.

Diagrama de flujo:



Si continuamos con el diagrama de flujo veríamos que es casi interminable.
Emplear una estructura secuencial para resolver este problema produce un diagrama de flujo y una programación en C muy largo.
Ahora veamos la solución empleando una estructura repetitiva while:





Es muy importante analizar este programa:

La primera operación inicializa la variable x en 1, seguidamente comienza la estructura repetitiva while y disponemos de la siguiente condición (x<=100), se lee MIENRAS la variable x sea menor o igual a 100.
Al ejecutarse la condición retorna VERDADERO porque el contenido de x (1) es menor o igual a 100.
Al ser la condición verdadera se ejecuta el bloque de instrucciones que contiene la estructura while.
El bloque de instrucciones contiene una salida y una operación.
Se imprime el contenido de x, y seguidamente se incrementa la variable en uno.
La operación x=x+1 se lee como "en la variable x se guarda el contenido de x más 1". Es decir, si x contiene 1 luego de ejecutarse este operación se almacenará en x un 2.
Al finalizar el bloque de instrucciones que contiene la estructura repetitiva se verifica nuevamente la condición de la estructura repetitiva y se repite el proceso explicado anteriormente.
Mientras la condición retorne verdadero se ejecutará el bloque de instrucciones; al retornar falso la verificación de la condición se sale de la estructura repetitiva y a continuación el algoritmo, en este caso finaliza el programa.

```c
#include<stdio.h>
#include<conio.h>

int main()
{

    int x=1;
    while (x<=100)
    {

        printf("%i", x);
        printf("-");
        x=x+1;
    }
    getch();
    return 0;
}

```

Si ejecutamos este será el resultado:

```
D:\CursoC\programa27.exe
1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25-26-27-28-29-30-31-32-33-34-35-36
-37-38-39-40-41-42-43-44-45-46-47-48-49-50-51-52-53-54-55-56-57-58-59-60-61-62-63-64-65-66-67-68-6
9-70-71-72-73-74-75-76-77-78-79-80-81-82-83-84-85-86-87-88-89-90-91-92-93-94-95-96-97-98-99-100-
```

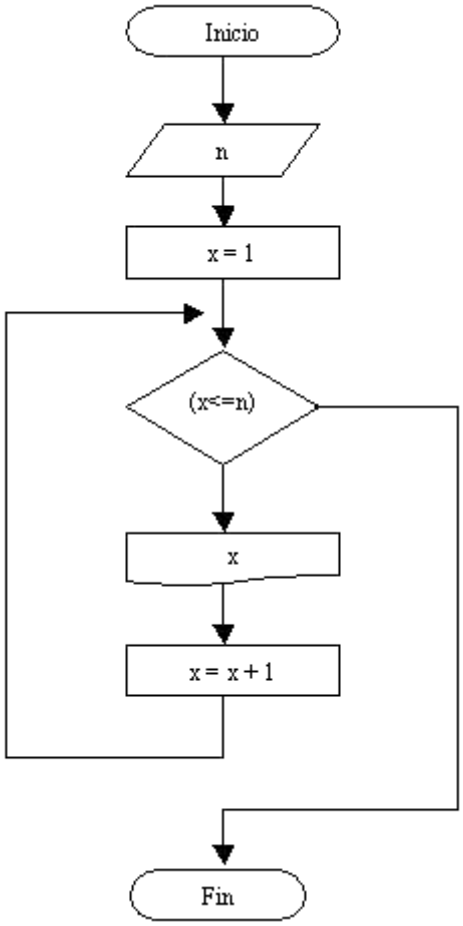
Capítulo 30.- Estructura repetitiva while – 2

Problema

Escribir un programa que solicite la carga de un valor positivo y nos muestre desde el 1 hasta el valor ingresado de uno en uno.

Ejemplo: Si ingresamos el 30 se deben mostrar en pantalla los números del 1 al 30.

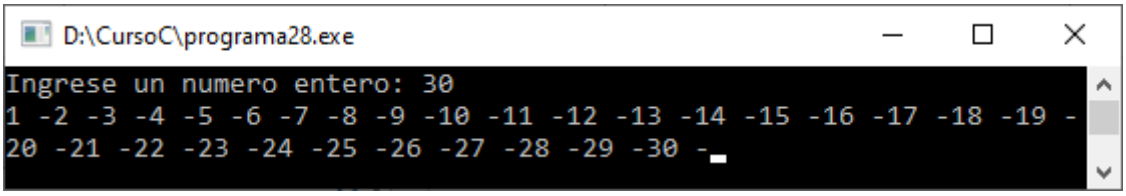
Diagrama de flujo:




```
#include<stdio.h>
#include<conio.h>

int main()
{
    int n, x=1;
    printf("Ingrese un numero entero: ");
    scanf("%i", &n);
    while (x<=n)
    {
        printf("%i -", x);
        x=x+1;
    }
```

```
14      getch();
15      return 0;
16  }
17
```

Vamos a ejecutar introduciendo el valor 30.

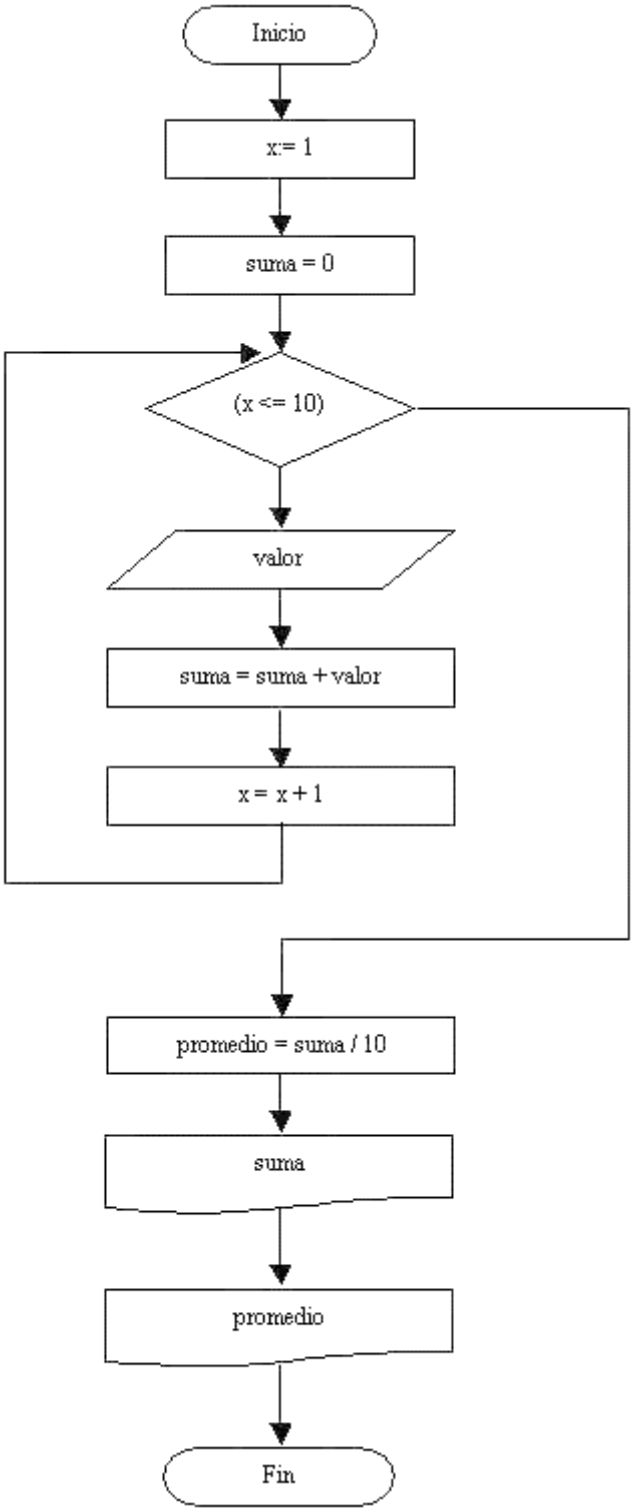


Capítulo 31.- Estructura repetitiva while – 3

Problema:

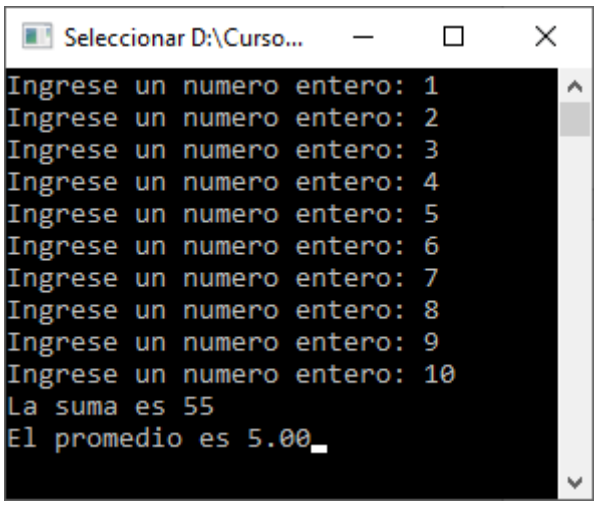
Desarrollar un programa que permita la carga de 10 valores por teclado y nos muestre posteriormente la suma de los valores ingresados y su promedio.

Diagrama de flujo:



```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int x=1;
    int suma=0, valor;
    float promedio;
    while (x<=10)
    {
        printf("Ingrese un numero entero: ");
        scanf("%i", &valor);
        suma=suma+valor;
        x=x+1;
    }
    promedio=suma/10;
    printf("La suma es %i", suma);
    printf("\n");
    printf("El promedio es %.2f", promedio);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



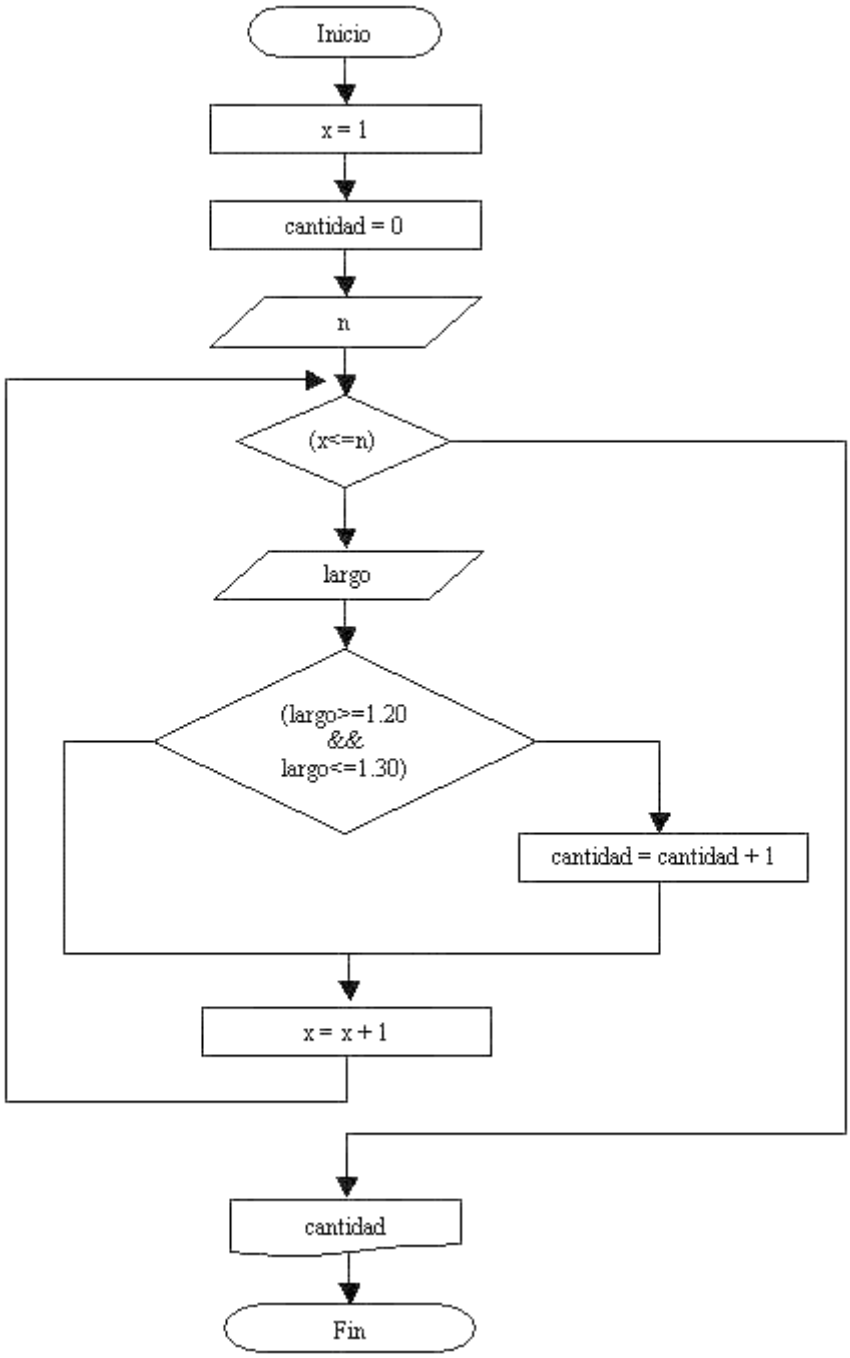
Capítulo 32.- Estructura repetitiva while – 4

Problema:

Una planta que fabrica perfiles de hierro posee un lote de n piezas.

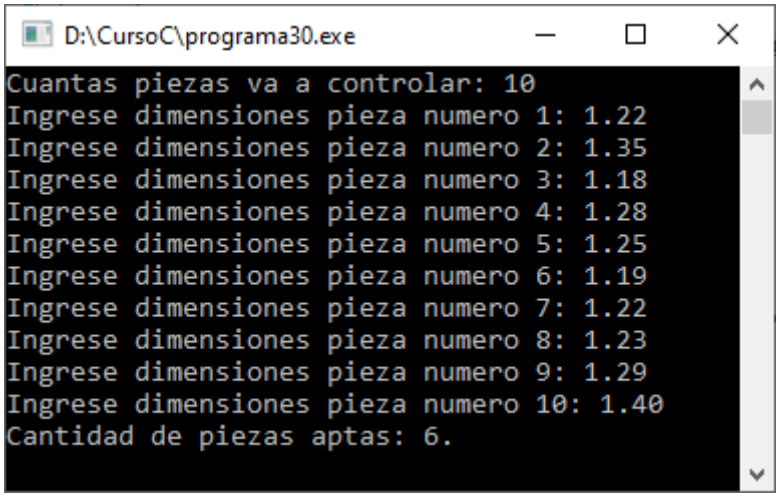
Confeccionar un programa que pida ingresar por teclado la cantidad de piezas a procesar y luego ingrese la longitud de cada perfil; sabiendo que la pieza cuya longitud esté comprendida en el rango de 1.20 y 1.30 son aptas. Imprimir por pantalla la cantidad de piezas aptas que hay en el lote.

Diagrama de flujo:



```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int x=1;
    int cantidad=0;
    int n;
    float largo;
    printf("Cuantas piezas va a controlar: ");
    scanf("%i", &n);
    while (x<=n)
    {
        printf("Ingrese dimensiones pieza numero %i: ", x);
        scanf("%f", &largo);
        if(largo>=1.20f && largo<=1.30f)
        {
            cantidad=cantidad+1;
        }
        x=x+1;
    }
    printf("Cantidad de piezas aptas: %i.", cantidad);
    getch();
    return 0;
}

```

Si ejecutamos este será el resultado:



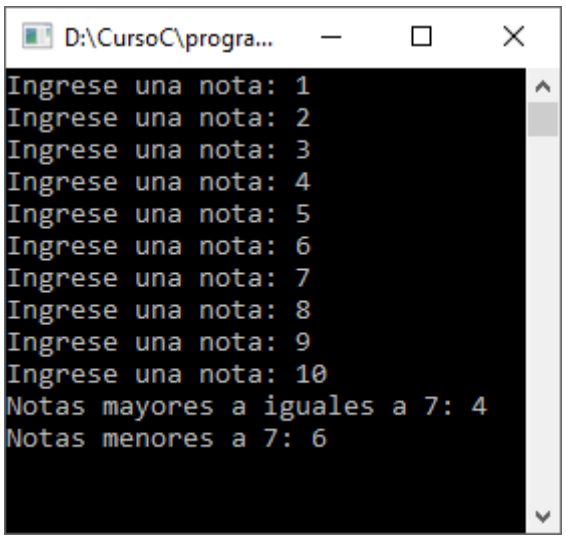
Capítulo 33.- Estructura repetitiva while – 5

Problema propuesto

Escribir un programa que solicite ingresar 10 notas de alumnos y nos informe cuántos tienen notas mayores o iguales a 7 y cuantos menores.

```
#include<stdio.h>
#include<conio.h>

int main()
{
    int nota, x=1;
    int altas=0, bajas=0;
    while(x<=10)
    {
        printf("Ingrese una nota: ");
        scanf("%i", &nota);
        if (x>=7)
        {
            altas=altas+1;
        }
        else
        {
            bajas=bajas+1;
        }
        x=x+1;
    }
    printf("Notas mayores a iguales a 7: ");
    printf("%i", altas);
    printf("\n");
    printf("Notas menores a 7: ");
    printf("%i", bajas);
    getch();
    return 0;
}

```

Si ejecutamos este será el resultado:



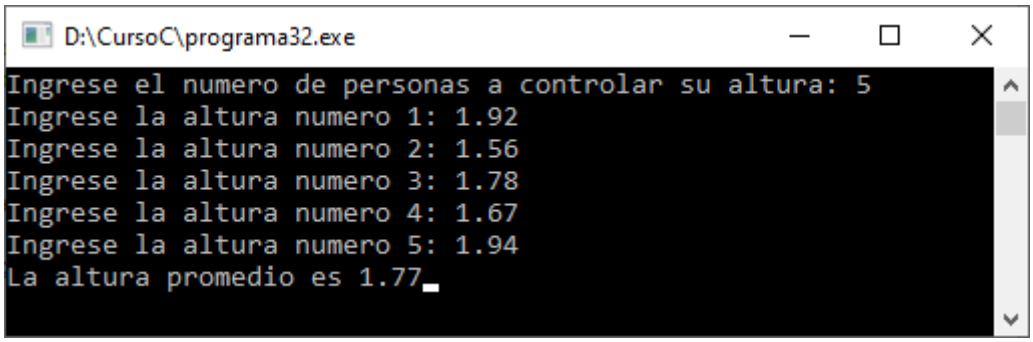
Capítulo 34.- Estructura repetitiva while – 6

Problema propuesto

Se ingresan un conjunto de n alturas de personas por teclado. Mostrar la altura promedio de las personas.

```
#include<stdio.h>
#include<conio.h>

int main()
{
    float altura, suma, promedio;
    int n, x=1;
    printf("Ingrese el numero de personas a controlar su altura: ");
    scanf("%i", &n);
    while (x<=n)
    {
        printf("Ingrese la altura numero %i: ", x);
        scanf("%f", &altura);
        suma=suma+altura;
        x=x+1;
    }
    promedio=suma/n;
    printf("La altura promedio es %.2f", promedio);
    getch();
    return 0;
}

```

Si ejecutamos este será el resultado:



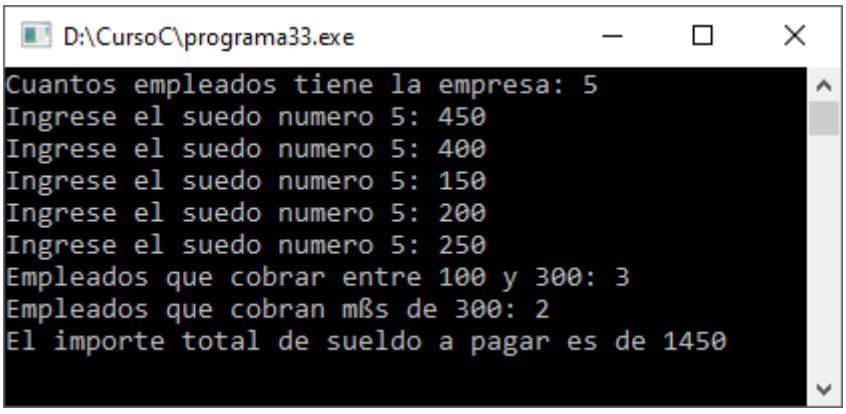
Capítulo 35.- Estructura repetitiva while – 7

Problema propuesto

En una empresa trabajan n empleados cuyos sueldos oscilan entre 100 y 500, realizar un programa que lea los sueldos que cobra cada empleado e informe cuantos empleados cobran entre 100 y 300 y cuantos cobran más de 300, además el programa deberá informar el importe que se gasta la empresa en sueldos de personal.

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int sueldo, menor=0, mayor=0, suma=0, n, x=1;
    printf("Cuantos empleados tiene la empresa: ");
    scanf ("%i", &n);
    while(x<=n)
    {
        printf("Ingrese el suedo numero %i: ", n);
        scanf("%i", &sueldo);
        if(sueldo>=100 && sueldo<=300)
        {
            menor=menor+1;
        }
        else
        {
            mayor=mayor+1;
        }
        suma=suma+sueldo;
        x=x+1;
    }
    printf("Empleados que cobrar entre 100 y 300: %i", menor);
    printf("\n");
    printf("Empleados que cobran más de 300: %i", mayor);
    printf("\n");
    printf("El importe total de sueldo a pagar es de %i", suma);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



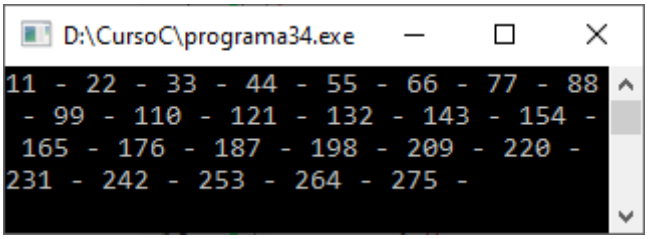
Capítulo 36.- Estructura repetitiva while – 8

Problema propuesto

Realizar un programa que imprima 25 términos de la serie 11 - 22 - 33 - 44, etc. (No se ingresan valores por teclado).

```
#include<stdio.h>
#include<conio.h>

int main()
{
    int x=1;
    while(x<=25)
    {
        printf("%i - ", x*11);
        x=x+1;
    }
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



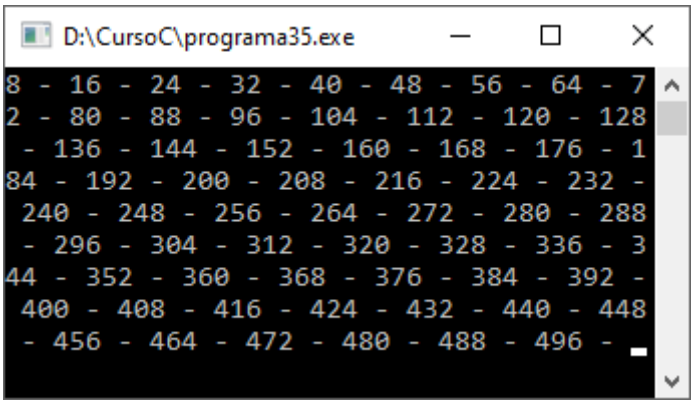
Capítulo 37.- Estructura repetitiva while – 9

Problema propuesto

Mostrar todos lo múltiplos de 8 que hay hasta el valor 500.
Debe aparecer en pantalla 8 – 16 – 24, etc.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   int main()
5   {
6       int x=8;
7       while (x<=500)
8       {
9           printf("%i - ", x);
10          x=x+8;
11      }
12
13      getch();
14      return 0;
15  }
16
```

Vamos a ejecutar:



Capítulo 38.- Estructura repetitiva while – 10

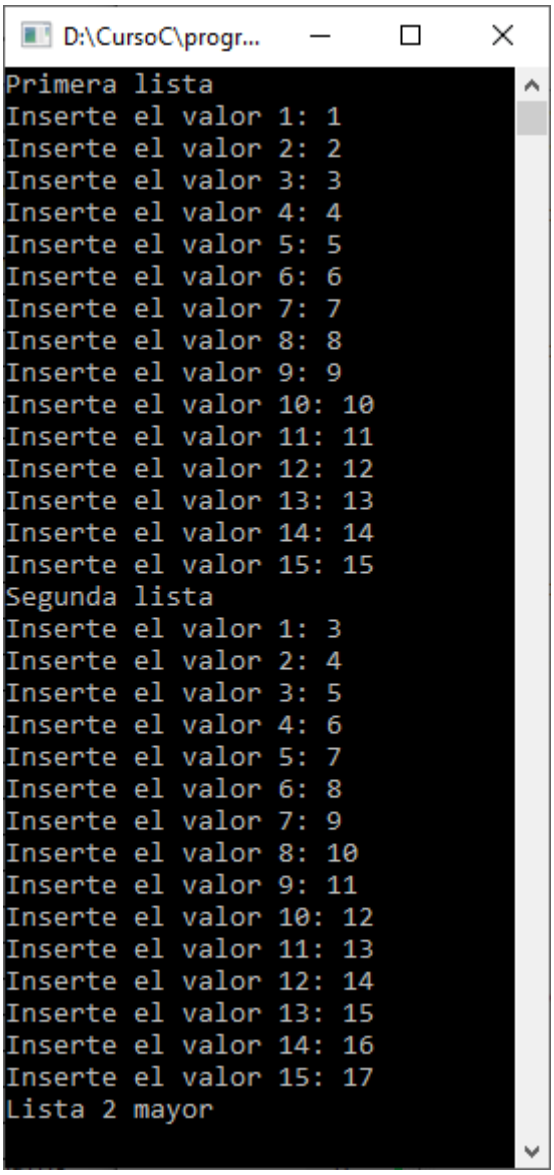
Problema propuesto

Realizar un programa que permita cargar dos listas de 15 valores cada una. Informar con un mensaje cual de las dos listas tiene un valor acumulado mayor (mensajes "Lista 1 mayor", "Lista 2 mayor", "Listas iguales").

Tener en cuenta que puede haber dos o más estructuras repetitivas en un algoritmo.

```
#include<stdio.h>
#include<conio.h>

int main()
{
    int sumLista1=0, sumLista2=0;
    int valor, x=1;
    printf("Primera lista\n");
    while(x<=15){
        printf("Inserte el valor %i: ", x);
        scanf("%i", &valor);
        sumLista1=sumLista1+valor;
        x=x+1;
    }
    printf("Segunda lista\n");
    x=1;
    while(x<=15){
        printf("Inserte el valor %i: ", x);
        scanf("%i", &valor);
        sumLista2=sumLista2+valor;
        x=x+1;
    }
    if (sumLista1>sumLista2)
    {
        printf("Lista 1 mayor");
    }
    else
    {
        if (sumLista2>sumLista1)
            {
                printf("Lista 2 mayor");
            }
            else
            {
                printf("Listas iguales");
            }
            }
    getch();
    return 0;
}

```

Si ejecutamos este será el resultado:


D:\CursoC\progr...
Primera lista
Inserte el valor 1: 1
Inserte el valor 2: 2
Inserte el valor 3: 3
Inserte el valor 4: 4
Inserte el valor 5: 5
Inserte el valor 6: 6
Inserte el valor 7: 7
Inserte el valor 8: 8
Inserte el valor 9: 9
Inserte el valor 10: 10
Inserte el valor 11: 11
Inserte el valor 12: 12
Inserte el valor 13: 13
Inserte el valor 14: 14
Inserte el valor 15: 15
Segunda lista
Inserte el valor 1: 3
Inserte el valor 2: 4
Inserte el valor 3: 5
Inserte el valor 4: 6
Inserte el valor 5: 7
Inserte el valor 6: 8
Inserte el valor 7: 9
Inserte el valor 8: 10
Inserte el valor 9: 11
Inserte el valor 10: 12
Inserte el valor 11: 13
Inserte el valor 12: 14
Inserte el valor 13: 15
Inserte el valor 14: 16
Inserte el valor 15: 17
Lista 2 mayor

Capítulo 39.- Estructura repetitiva while – 11

Problema propuesto

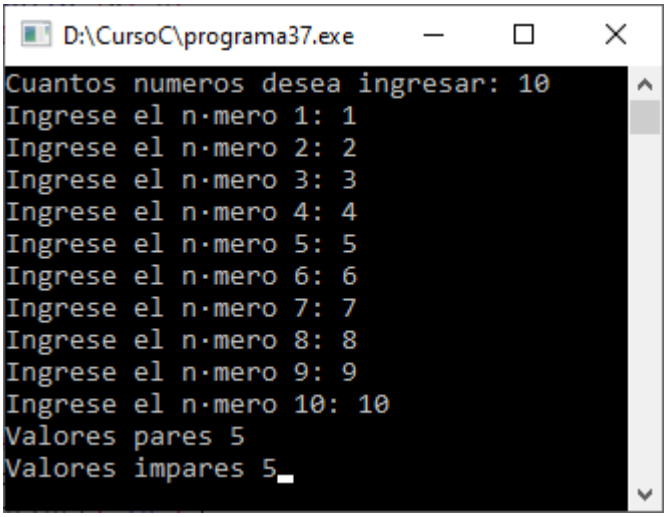
Desarrollar un programa que permita cargar n números enteros y luego no informe cuantos valores fueron pares y cuantos impares.

Emplear el operador "%" en la condición de la estructura condicional (este operador retorna el resto de la división de dos valores, por ejemplo 11%2 retorna 1):

```
if (valor%2==0)          //Si el if da verdadero luego es par.
```

```
#include<stdio.h>
#include<conio.h>

int main()
{
    int n, par=0, impar=0, x=1, numero;
    printf("Cuantos numeros desea ingresar: ");
    scanf("%i", &n);
    while (x<=n)
    {
        printf("Ingrese el número %i: ", x);
        scanf("%i", &numero);
        if (numero%2==0)
        {
            par=par+1;
        }
        else
        {
            impar=impar+1;
        }
        x=x+1;
    }
    printf("Valores pares %i", par);
    printf("\n");
    printf("Valores impares %i", impar);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:

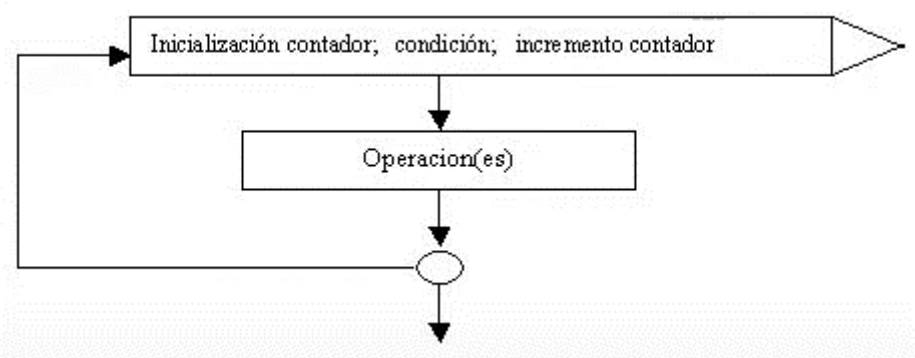


Capítulo 40.- Estructura repetitiva for – 1

Cualquier problema que requiera una estructura repetitiva se puede resolver empleando la estructura while. Pero hay otra estructura repetitiva cuyo planteo es más sencillo en ciertas situaciones.

En general, la estructura for se usa en aquellas situaciones en las cuales CONOCEMOS la cantidad de veces que queremos que se ejecute el bloque de instrucciones. Ejemplo: cargar 10 números, ingresar 5 notas de alumnos, etc. Conocemos de antemano la cantidad de veces que queremos que el bloque se repita. Veremos, sin embargo, que en el lenguaje C la estructura for puede usarse en cualquier situación repetitiva, porque en última instancia no es otra cosa qaue una estructura while generalizada.

Representación gráfica:



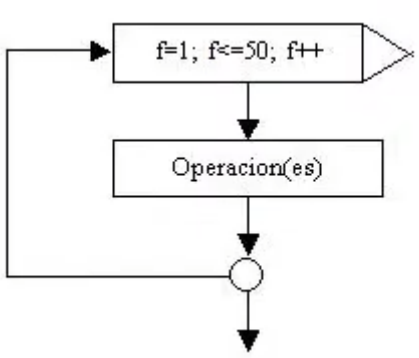
En su forma más típica y básica, esta estructura requiere una variable entera que cumple la función de un CONTADOR de vueltas. En la sección indicada como "inicialización contador", se suele colocar el nombre de la variable que hará de contador, asignándole a dicha variable un valor inicial.

En la sección de "condición" se coloca la condición que deberá ser verdadera para que el ciclo continúe (en caso de un falso, el ciclo se detendrá). Y finalmente, en la sección de "incremento contador" se coloca una instrucción que permite modificar el valor de la variable que hace de contador (para permitir que alguna vez la condición sea falsa y finalice el for).

Cuando el ciclo comienza, antes de dar la primera vuelta, la variable del for toma el valor indicado en la sección de "inicializar contador". Inmediatamente se verifica, en forma automática, si la condición es verdadera. En caso de serlo se ejecuta el bloque de operaciones del ciclo, y al finalizar el mismo se ejecuta la instrucción que se haya colocado en la tercera sección.

Seguidamente, se vuelve a controlar el valor de la condición, y así prosigue hasta que dicha condición entregue un falso.

Si conocemos la cantidad de veces que se repite el bloque es muy sencillo emplear un for, por ejemplo si queremos que se repita 50 veces el bloque de instrucciones puede hacerse así:



La variable del for puede tener cualquier nombre. En este ejemplo se la ha definido con el nombre f.

Analicemos el ejemplo:

- La variable f toma inicialmente el valor 1.
- Se controla automáticamente el valor de la condición: como f vale 1 y esto es menor o igual que 50, la condición es verdadero.
- Como la condición fue verdadera, se ejecuta la/s instrucción/es.
- Al finalizar de ejecutarlas, se retorna a la instrucción f++ (el operador ++ incrementa en uno la variable f, es lo mismo que escribir f=f+1, por lo que la variable f se incrementa en uno.
- Se vuelve a controlar (automáticamente) si f es menor o igual a 50. Como ahora su valor es 2, se ejecuta nuevamente el bloque de instrucciones e incrementa nuevamente la variable del for al terminar el mismo.
- El proceso se repetirá hasta que la variable f sea incrementada al valor 51. En este momento la condición será falsa, y el ciclo se detendrá.

La variable f PUEDE ser modificada dentro del bloque de operaciones del for, aunque esto podría causar problemas de lógica si el programador es inexperto.

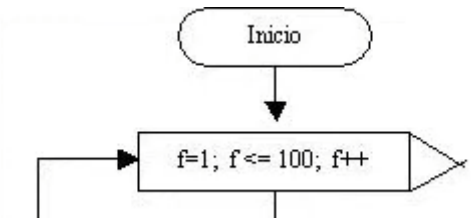
La variable f puede ser inicializada con cualquier valor y finalizar en cualquier otro valor. Además, no es obligatorio que la instrucción de modificación sea un incremento de tipo contador (f++).

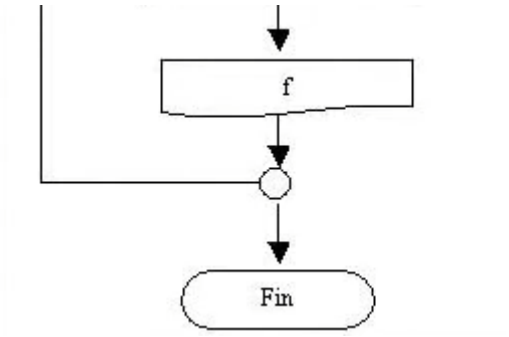
Cualquier instrucción que modifique el valor de la variable es válida. Si por ejemplo se escribe f=f+1 en lugar de f++, el valor de f será incrementado de a 2 en cada vuelta, y no de a 1. En este caso, esto significa que el ciclo no efectuará las 50 vueltas sino sólo 25.

Problema

Realizar un programa que imprima en pantalla los valores del 1 al 100.

Diagrama de flujo





Podemos observar y comparar con el problema realizado con el while. Con la estructura while el CONTADOR x sirve contar las vueltas. Con el for el CONTADOR f cumple dicha función.

Inicialmente f vale 1 y como no es superior a 100 se ejecuta el bloque, imprimimos el contendio de f, al finalizar el bloque repetitivo se incremente la variable f 2n 1, como 2 no es superior a 100 se repite el bloque de instrucciones.

Cuando la variable del for llega al 101 sale de la estructura repetitiva y continúa la ejecución del algoritmo que se indica después del circulo.

La variable f (o como sea que se decida llamarla) debe estar definida cono una variable más.

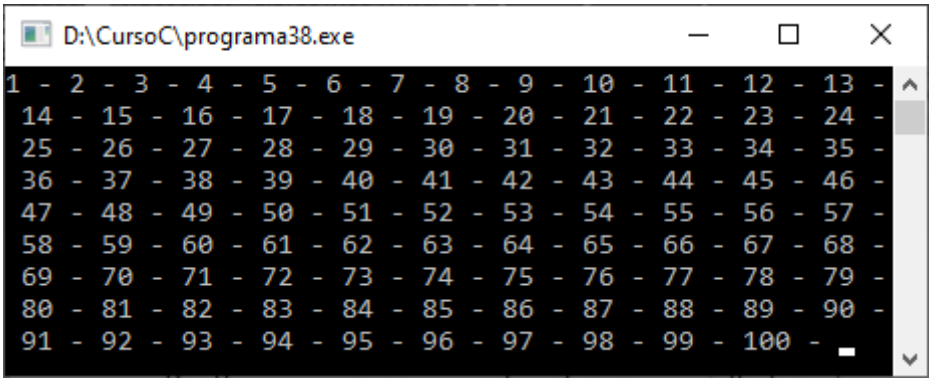
```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int f=1;
    while (f<=100){
        printf("%i - ", f);
        f++;
    }

    getch();
    return 0;
}

```

while

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int f;
    for (f=1; f<=100; f++)
    {
        printf("%i - ", f);
    }
    getch();
    return 0;
}

```

for

Este será el resultado con ambos ejemplos:

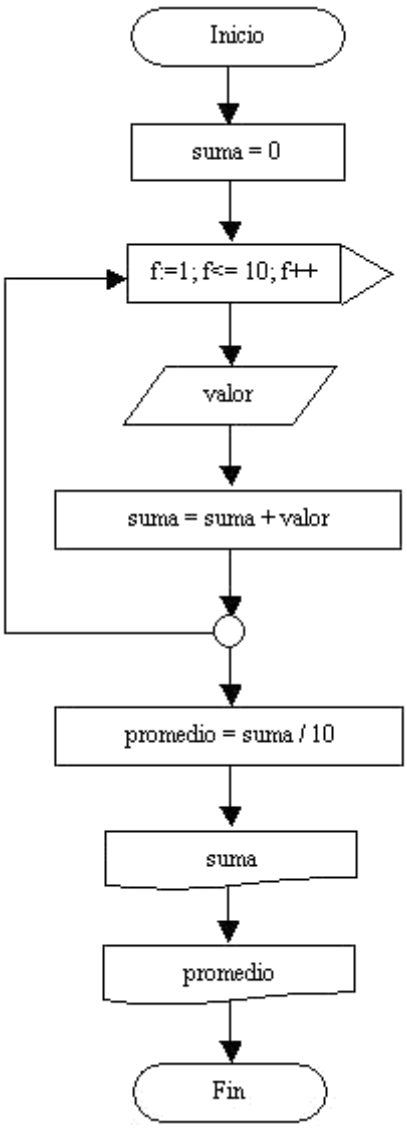


Capítulo 41.- Estructura repetitiva for – 2

Problema

Desarrollar un programa que permita la carga de 10 valores por teclado y nos muestre posteriormente la suma de los valores ingresados y su promedio. Este problema ya lo desarrollamos empleando el while, lo resolveremos empleando la estructura for.

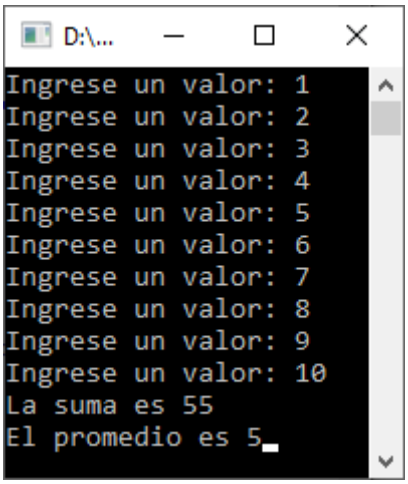
Diagrama de flujo:




```
1  #include<stdio.h>
2  #include<conio.h>
3
4  int main()
5  {
6      int suma=0;
7      int f, valor, promedio;
8      for(f=1; f<=10; f++)
9      {
```

```
10          printf("Ingrese un valor: ");
11          scanf("%i", &valor);
12          suma=suma+valor;
13      }
14      promedio=suma/10;
15      printf("La suma es %i", suma);
16      printf("\n");
17      printf("El promedio es %i", promedio);
18
19      getch();
20      return 0;
21  }
```

Si ejecutamos este será el resultado:



Capítulo 42.- Estructura repetitiva for – 3

Problema

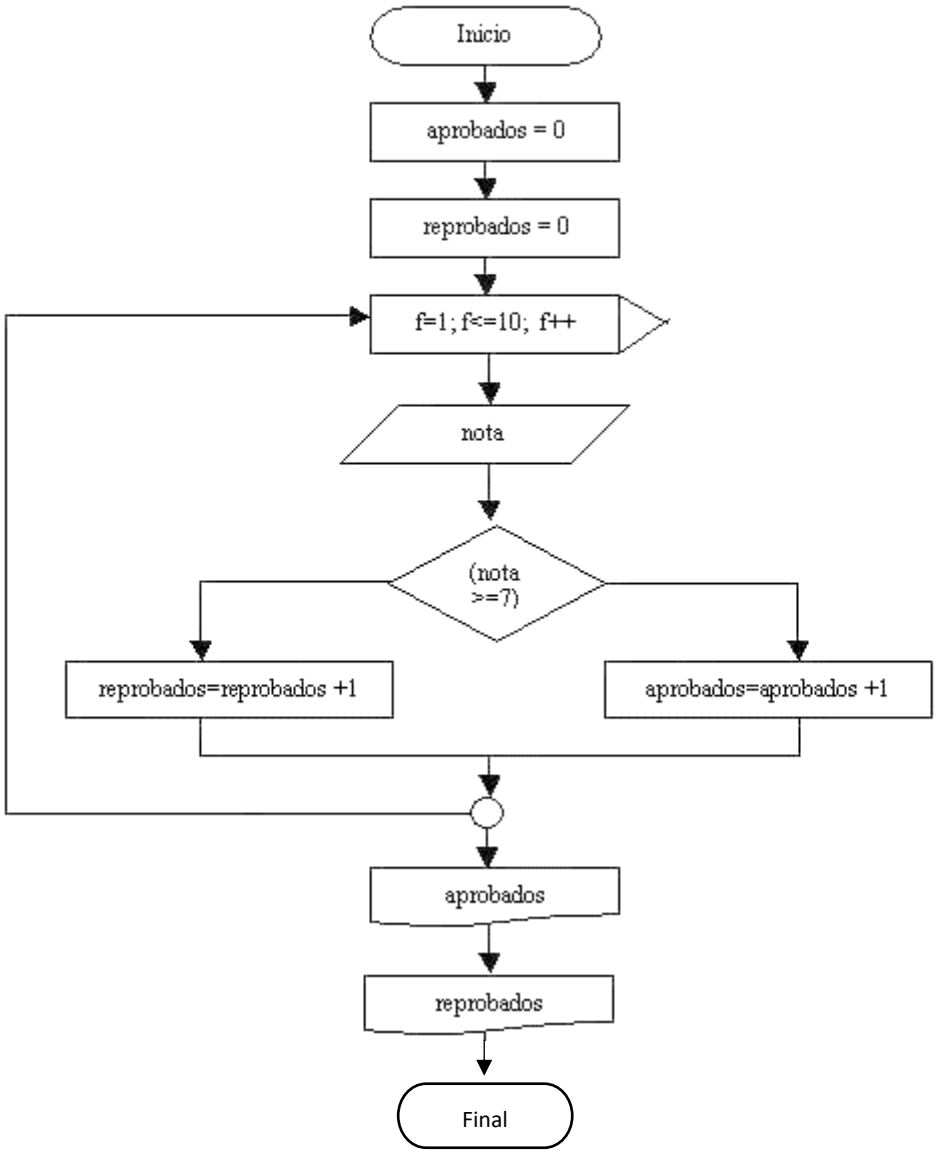
Escribir un programa que lea 10 notas de alumnos y nos informe cuantos tienen notas mayores a iguales a 7 y cuanto menores.

Para resolver este problema se requieren tres contadores:

aprobados (Cuenta la cantidad de alumnos aprobados)
reprobados (Cuenta la cantidad de alumnos reprobados)
f (es el contador del for)

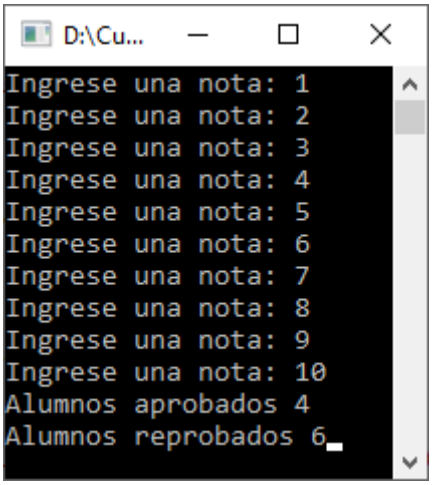
Dentro de la estructura repetitiva debemos hacer la carga de la variable nota y verificar con una estructura condicional si el contenido de la variable nota es mayor o igual a 7 para incrementar el contador de aprobados, en caso de que la condición retorne falso debemos incrementar la variable reprobados.

Diagrama de flujo:



```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int aprobados=0, reprobados=0, f, nota;
    for (f=1; f<=10; f++)
    {
        printf("Ingrese una nota: ");
        scanf("%i", &nota);
        if(nota>=7)
        {
            aprobados=aprobados+1;
        }
        else
        {
            reprobados=reprobados+1;
        }
    }
    printf("Alumnos aprobados %i", aprobados);
    printf("\n");
    printf("Alumnos reprobados %i", reprobados);

    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:

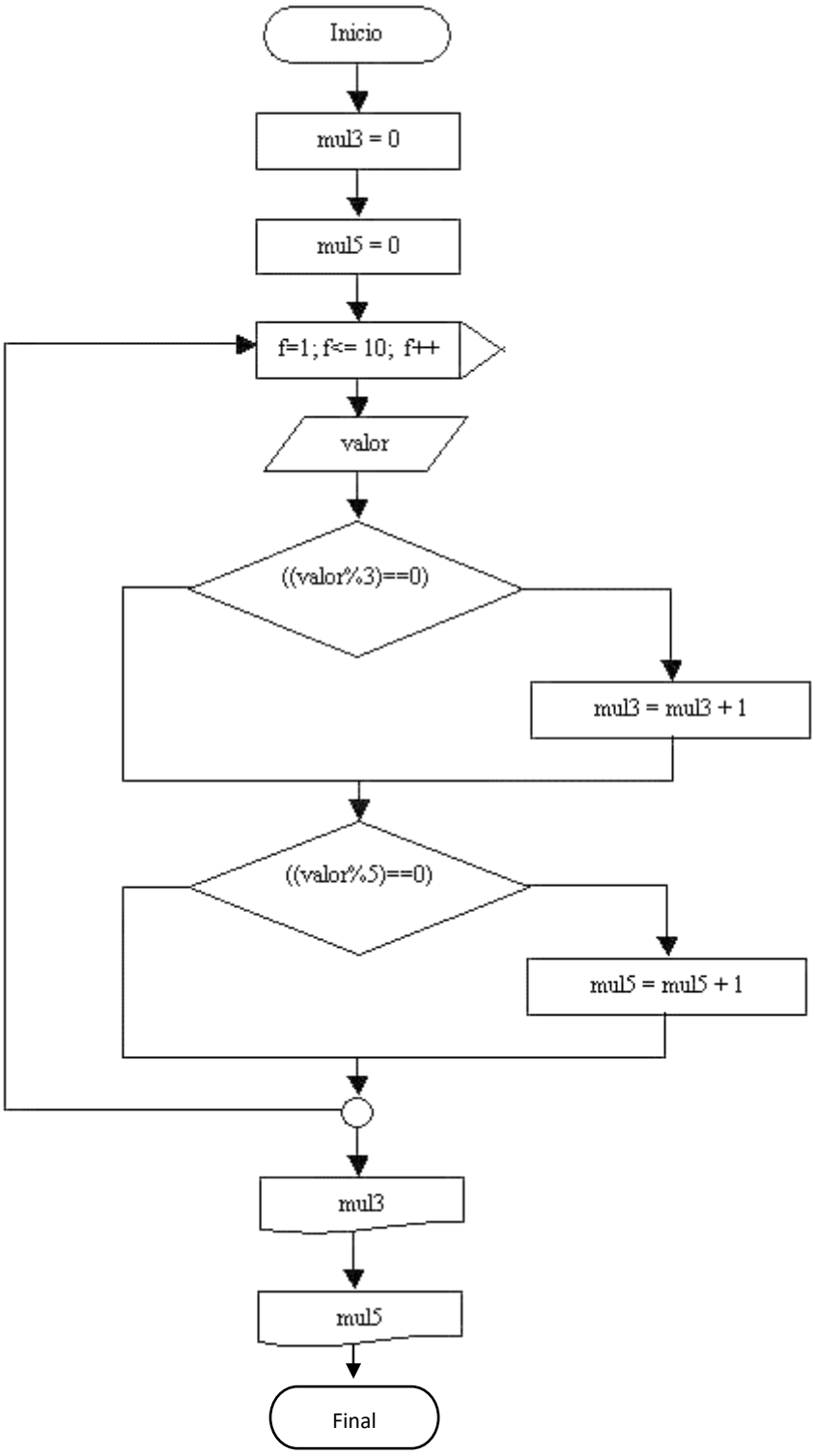


Capítulo 43.- Estructura repetitiva for – 4

Problema

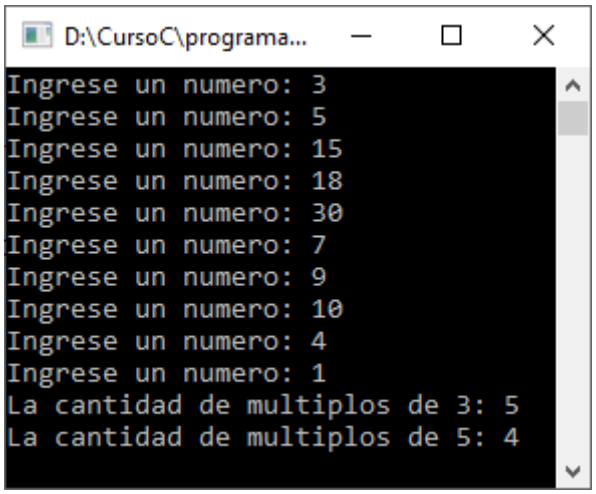
Escribir un programa que lea 10 números enteros y luego muestre cuántos valores ingresados fueros múltiplos de 3 y cuantos de 5. Debemos tener en cuenta que hay números que son múltiplos de 3 y de 5 a la vez.

Diagrama de flujo:



```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int mul3=0, mul5=0;
    int f, valor;
    for (f=1; f<=10; f++)
    {
        printf("Ingrese un numero: ");
        scanf("%i", &valor);
        if (valor%3==0)
        {
           mul3=mul3+1;
        }
        if (valor%5==0)
        {
            mul5=mul5+1;
        }
    }
    printf("La cantidad de multiplos de 3: %i", mul3);
    printf("\n");
    printf("La cantidad de multiplos de 5: %i", mul5);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 44.- Estructura repetitiva for – 5

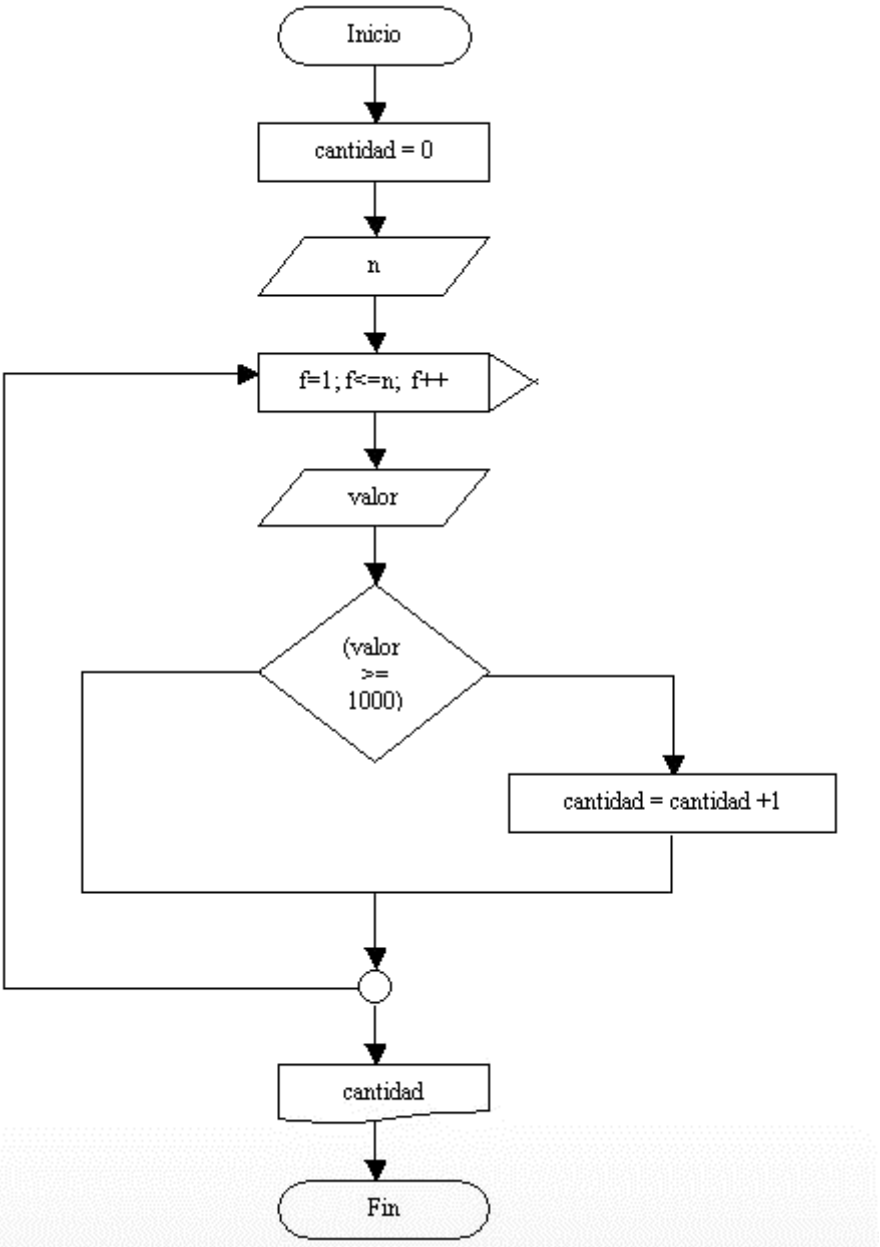
Problema

Escribir un programa que lea n números enteros y calcule la cantidad de valores mayores o iguales a 1000.

Este tipo de problemas también se puede resolver empleando la estructura repetitiva for. Lo primero que se hace es cargar una variable que indique la cantidad de valores a ingresar. Dicha variable se carga antes de entrar a la estructura repetitiva for.

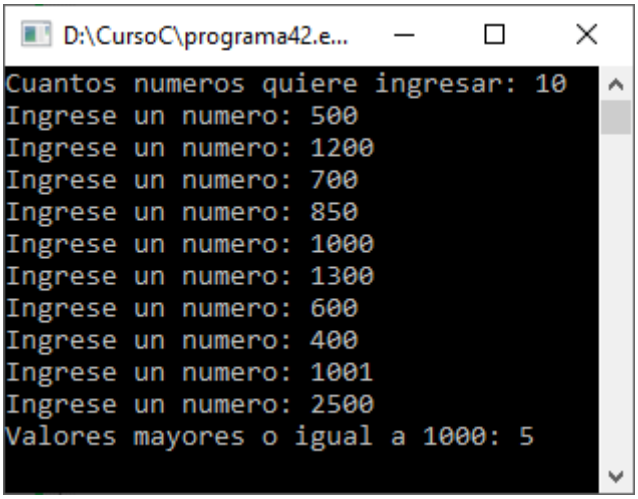
La estructura for permite que el valor inicial o final dependan de una variable cargada previamente por el teclado.

Diagrama de flujo:



```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int cantidad=0;
    int n, f, valor;
    printf("Cuantos numeros quiere ingresar: ");
    scanf("%i", &n);
    for(f=1; f<=n; f++)
    {
        printf("Ingrese un numero: ");
        scanf("%i", &valor);
        if (valor>=1000)
        {
            cantidad=cantidad+1;
        }
    }
    printf("Valores mayores o igual a 1000: %i", cantidad);

    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 45.- Estructura repetitiva for – 6

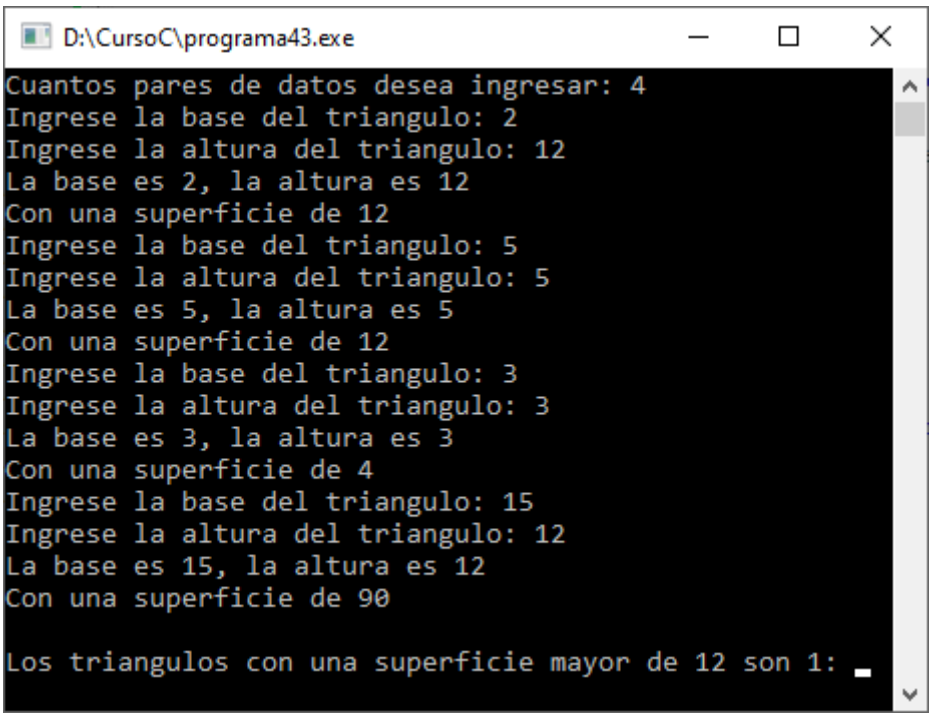
Problema propuesto

Confeccionar un programa a que lea n pares de datos, cada par de datos corresponde a la medida de la base y la altura de un triángulo. En programa deberá informar:

a) De cada triángulo la medida de su base, su altura y su superficie.
b) La cantidad de triángulos cuya superficie es mayor a 12.

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int base, altura, superficie, n, x;
    int cantidad=0;
    printf("Cuantos pares de datos desea ingresar: ");
    scanf("%i", &n);
    for (x=1; x<=n; x++)
        {
            printf("Ingrese la base del triangulo: ");
            scanf("%i", &base);
            printf("Ingrese la altura del triangulo: ");
            scanf("%i", &altura);
            superficie=base*altura/2;
            printf("La base es %i, la altura es %i", base, altura);
            printf("\n");
            printf("Con una superficie de %i", superficie);
            printf("\n");
            if (superficie>12)
            {
                cantidad++;
            }
        }
    printf("\n");
    printf("Los triangulos con una superficie mayor de 12 son %i: ", cantidad);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



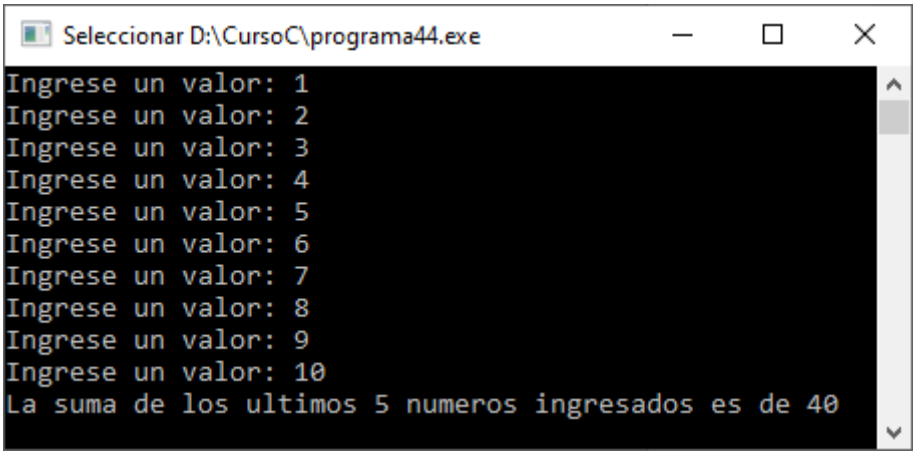
Capítulo 46.- Estructura repetitiva for – 7

Problema propuesto

Desarrollar un programa que solicite la carga de 10 números e imprima la suma de los últimos 5 valores ingresados.

```
#include<stdio.h>
#include<conio.h>

int main()
{
    int x, valor;
    int suma=0;
    for (x=1; x<=10; x++)
    {
        printf("Ingrese un valor: ");
        scanf("%i", &valor);
        if(x>5)
        {
            suma=suma+valor;
        }
    }
    printf("La suma de los ultimos 5 numeros ingresados es de %i", suma);
    getch();
    return 0;
}

```

Si ejecutamos este será el resultado:



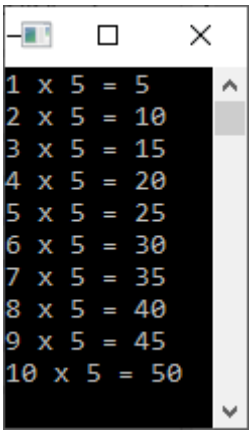
Capítulo 47.- Estructura repetitiva for – 8

problema propuesto

Desarrollar un programa que muestre la tabla de multiplicar del 5 (del 5 al 50).

```
#include<stdio.h>
#include<conio.h>

int main()
{
    int t;
    for (t=1; t<=10; t++)
    {
        printf("%i x 5 = %i", t, t*5);
        printf("\n");
    }
     getch();
     return 0;
}
```

Si ejecutamos este será el resultado:

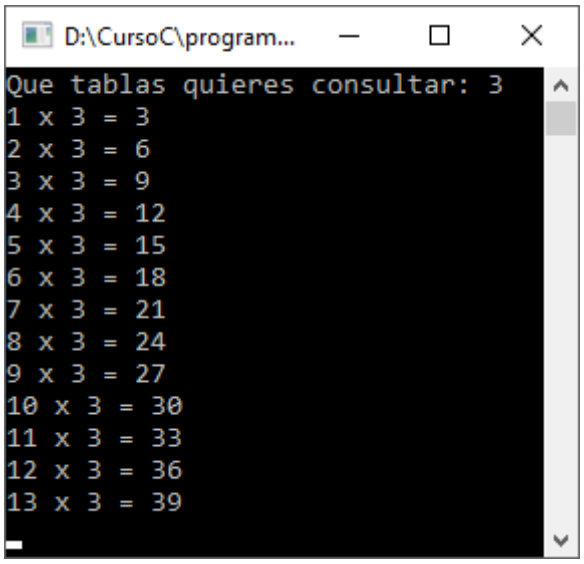


Capítulo 48.- Estructura repetitiva for – 9

Problema propuesto

Confeccionar un programa que permita ingresar un valor del 1 al 10 y nos muestre la tabla de multiplicar del mimo (los primeros 13 términos).

Ejemplo: si ingresamos la tabla del 3.



Este será el código:

```
#include<stdio.h>
#include<conio.h>

int main()
{
   int t, tabla;
   printf("Que tablas quieres consultar: ");
   scanf("%i", &tabla);
   for (t=1; t<=13; t++)
   {
       printf("%i x %i = %i", t, tabla, t*tabla);
       printf("\n");
   }
    getch();
    return 0;
}
```

Capítulo 49.- Estructura repetitiva for – 10

Problema propuesto

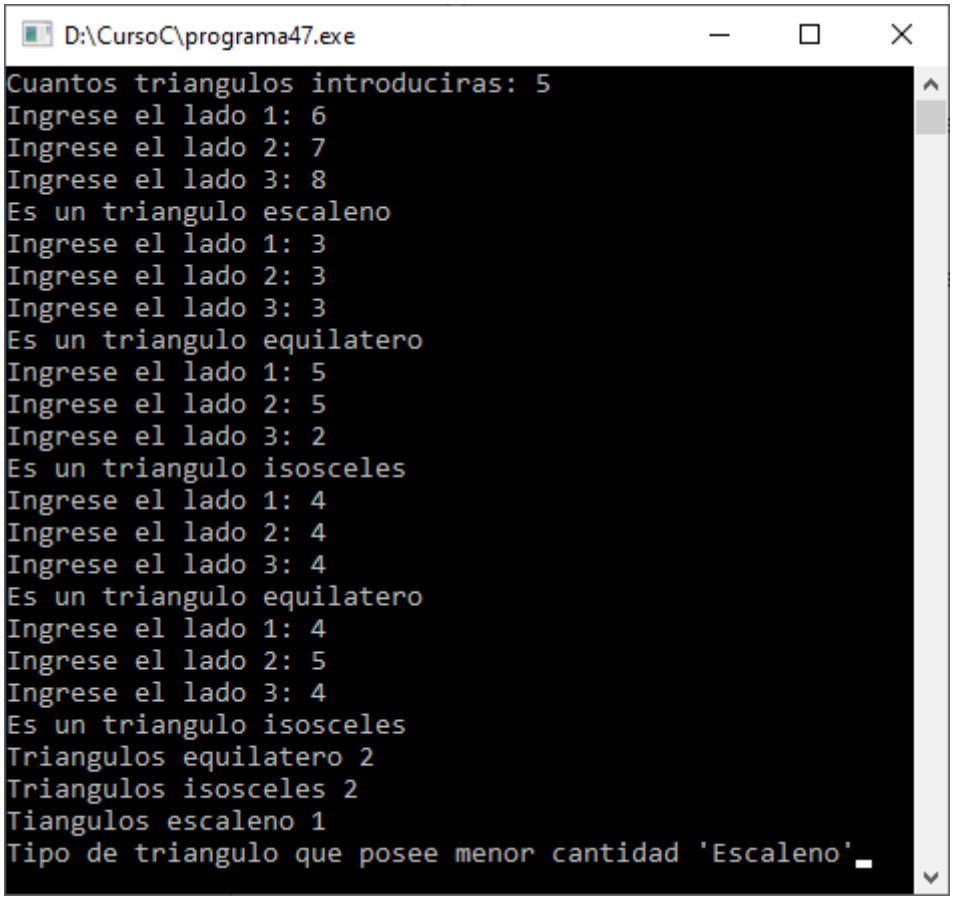
Realizar un programa que lea los lados de n triángulos, e informar:

a) De cada uno de ellos, qué tipo de triángulo es: equilátero (tres lados iguales), isósceles(dos lados iguales), o escaleno (ningún lado igual).
b) Cantidad de triángulos de cada tipo.
c) Tipo de triángulo que posee menor cantidad.

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int lado1, lado2, lado3;
    int n, x;
    int equi=0, isos=0, esca=0;
    printf("Cuantos triangulos introduciras: ");
    scanf("%i", &n);
    for (x=1; x<=n; x++)
    {
        printf("Ingrese el lado 1: ");
        scanf("%i", &lado1);
        printf("Ingrese el lado 2: ");
        scanf("%i", &lado2);
        printf("Ingrese el lado 3: ");
        scanf("%i", &lado3);
        if (lado1==lado2 && lado1==lado3)
        {
            equi++;
            printf("Es un triangulo equilatero\n");
        }
        else
        {
            if(lado1==lado2 || lado1==lado3 || lado2==lado3)
            {
                isos++;
                printf("Es un triangulo isosceles\n");
            }
            else
            {
               esca++;
               printf("Es un triangulo escaleno\n");
            }
        }
    }
    printf("Triangulos equilatero %i", equi);
    printf("\n");
    printf("Triangulos isosceles %i", isos);
    printf("\n");
    printf("Tiangulos escaleno %i", esca);
    printf("\n");

    if(equi<isos && equi<esca)
    {
        printf("Tipo de triangulo que posee menor cantidad 'Equilatero'");
    }
    else
```

```
50      {
51          if (isos<esca)
52          {
53              printf("Tipo de triangulo que posee menor cantidad 'Isosceles'");
54          }
55          else
56          {
57              printf("Tipo de triangulo que posee menor cantidad 'Escaleno'");
58          }
59      }
60
61      getch();
62      return 0;
63  }
64
```

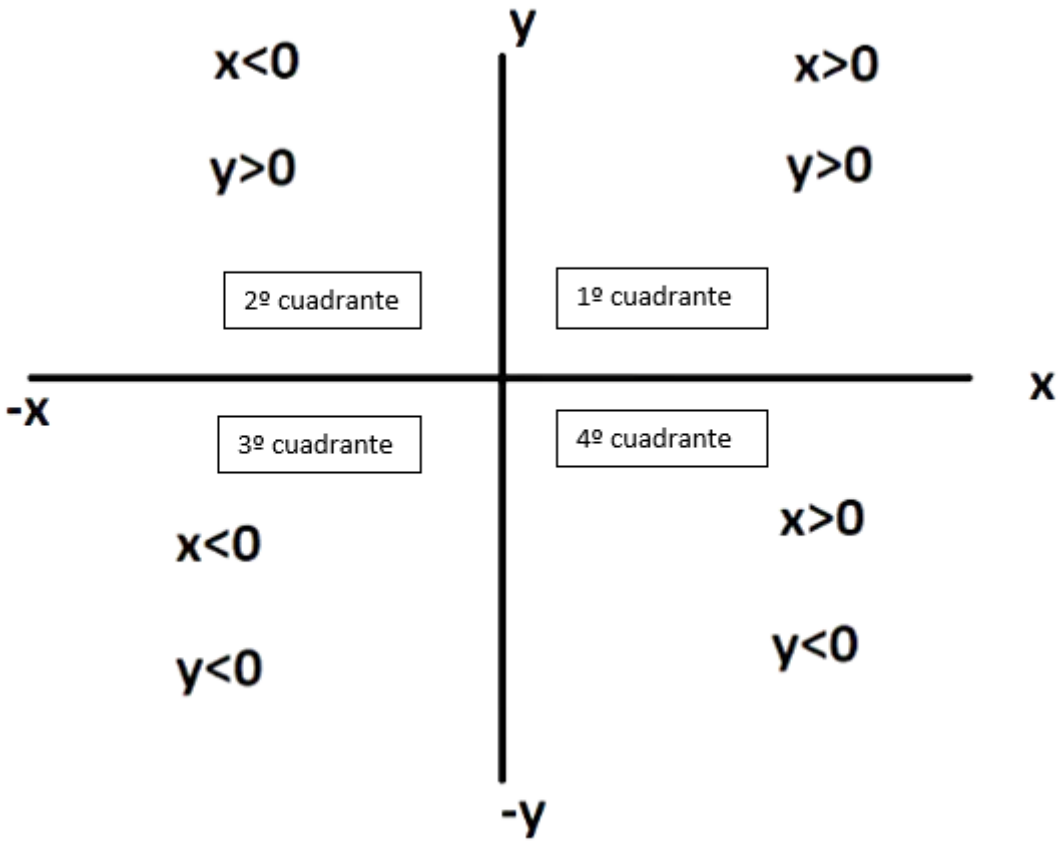
Si ejecutamos este será el resultado:



Capítulo 50.- Estructura repetitiva for – 11

Problema propuesto

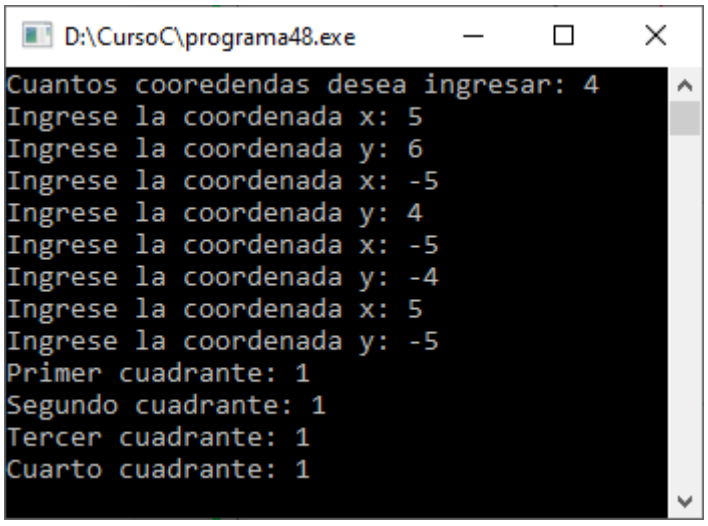
Escribir un programa que pida ingresar coordenadas (x,y) que representan puntos en el plano. Informar cuantos puntos se han ingresado en el primer, segundo, tercer y cuarto cuadrante. Al comenzar el programa se pide que ingrese la cantidad de puntos a procesar.




```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int primer=0, segundo=0, tercer=0, cuarto=0;
    int n, f, x, y;
    printf("Cuantos cooredendas desea ingresar: ");
    scanf("%i", &n);
    for (f=1; f<=n; f++)
    {
        printf("Ingrese la coordenada x: ");
        scanf("%i", &x);
        printf("Ingrese la coordenada y: ");
        scanf("%i", &y);
        if(x>0 && y>0)
        {
            primer++;
        }
```

```
20          else
21          {
22              if(x<0 && y>0)
23              {
24                  segundo++;
25              }
26              else
27              {
28                  if(x<0 && y<0)
29                  {
30                      tercer++;
31                  }
32                  else
33                  {
34                      if(x>0 && y<0)
35                      {
36                          cuarto++;
37                      }
38                  }
39              }
40          }
41      }
42      printf("Primer cuadrante: %i", primer);
43      printf("\n");
44      printf("Segundo cuadrante: %i", segundo);
45      printf("\n");
46      printf("Tercer cuadrante: %i", tercer);
47      printf("\n");
48      printf("Cuarto cuadrante: %i", cuarto);
49      getch();
50      return 0;
51  }
```

Ejecutamos este será el resultado:



Capítulo 51.- Estructura repetitiva for – 12

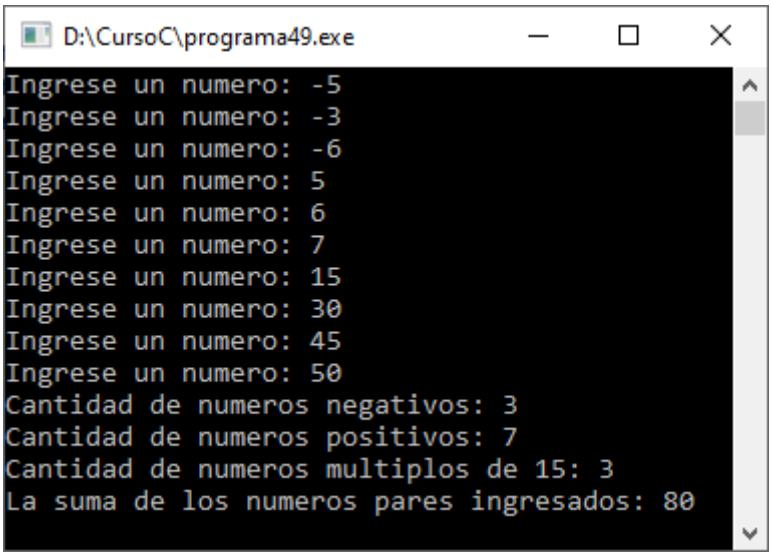
Problema propuesto

Se realiza una carga de 10 valores enteros por teclado. Se desea conocer:

a) La cantidad de valores ingresados negativos.
b) La cantidad de valores ingresados positivos.
c) La cantidad de múltiplos de 15.
d) El valor acumulado de los números ingresados que son pares.

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int positivo=0, negativo=0, multl5=0, sumapares=0;
    int n, numero;
    for (n=1; n<=10; n++)
    {
        printf("Ingrese un numero: ");
        scanf("%i", &numero);
        if (numero>0){
            positivo++;
        }
        else
        {
            if(numero<0)
            {
                negativo++;
            }
        }
        if (numero%15==0)
        {
            multl5++;
        }
        if (numero%2==0)
        {
            sumapares=sumapares+numero;
        }
    }
    printf("Cantidad de numeros negativos: %i", negativo);
    printf("\n");
    printf("Cantidad de numeros positivos: %i", positivo);
    printf("\n");
    printf("Cantidad de numeros multiplos de 15: %i",multl5);
    printf("\n");
    printf("La suma de los numeros pares ingresados: %i", sumapares);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 52.- Estructura repetitiva for – 13

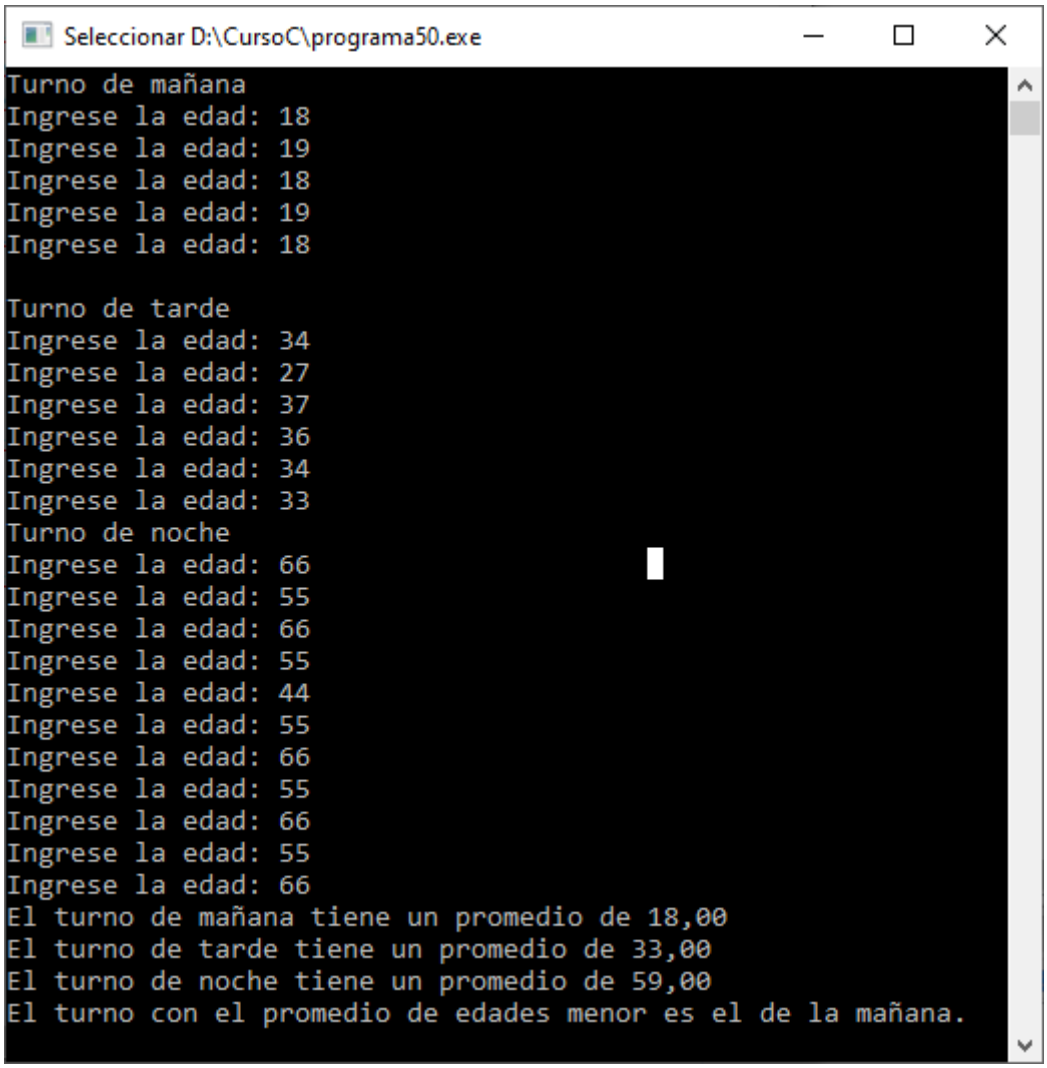
Problema propuesto

Se cuenta con la siguiente información:
Las edades de 5 estudiantes del turno de mañana.
Las edades de 6 estudiantes del turno de tarde.
Las edades de 11 estudiantes del turno de noche.
Las edades de cada estudiante deben ingresarse por teclado.

a) Obtener el promedio de las edades de cada turno (tres promedios)
b) Imprimir dichos promedios (promedio por turno)
c) Mostrar por pantalla un mensaje que indique cuál de los turnos tiene un promedio de edades menor.

```c
#include<stdio.h>
#include<conio.h>
#include<locale.h>

int main()
{
    setlocale(LC_ALL,"");
    int ma=0, ta=0, no=0;
    float proma, prota, prono;
    int x, edad;
    printf("Turno de mañana\n");
    for (x=1; x<=5; x++){
        printf("Ingrese la edad: ");
        scanf("%i", &edad);
        ma=ma+edad;
    }
    proma=ma/5;
    printf("\n");
    printf("Turno de tarde\n");
    for (x=1; x<=6; x++){
        printf("Ingrese la edad: ");
        scanf("%i", &edad);
        ta=ta+edad;
    }
    prota=ta/6;
    printf("Turno de noche\n");
    for (x=1; x<=11; x++){
        printf("Ingrese la edad: ");
        scanf("%i", &edad);
        no=no+edad;
    }
    prono=no/11;
    printf("El turno de mañana tiene un promedio de %.2f\n", proma);
    printf("El turno de tarde tiene un promedio de %.2f\n", prota);
    printf("El turno de noche tiene un promedio de %.2f\n", prono);

    if (proma<prota && proma<prono)
    {
        printf("El turno con el promedio de edades menor es el de la mañana.");
    }
    else
    {
        if (prota<prono)
        {
            printf("El turno con el promedio de edades menor es el de la tarde.");

        }
        else
        {
            printf("El turno con el promedio de edades menor es el de la noche.");
        }
    }

    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:


Seleccionar D:\CursoC\programa50.exe
Turno de mañana
Ingrese la edad: 18
Ingrese la edad: 19
Ingrese la edad: 18
Ingrese la edad: 19
Ingrese la edad: 18
Turno de tarde
Ingrese la edad: 34
Ingrese la edad: 27
Ingrese la edad: 37
Ingrese la edad: 36
Ingrese la edad: 34
Ingrese la edad: 33
Turno de noche
Ingrese la edad: 66
Ingrese la edad: 55
Ingrese la edad: 66
Ingrese la edad: 55
Ingrese la edad: 44
Ingrese la edad: 55
Ingrese la edad: 66
Ingrese la edad: 55
Ingrese la edad: 66
Ingrese la edad: 55
Ingrese la edad: 66
El turno de mañana tiene un promedio de 18,00
El turno de tarde tiene un promedio de 33,00
El turno de noche tiene un promedio de 59,00
El turno con el promedio de edades menor es el de la mañana.

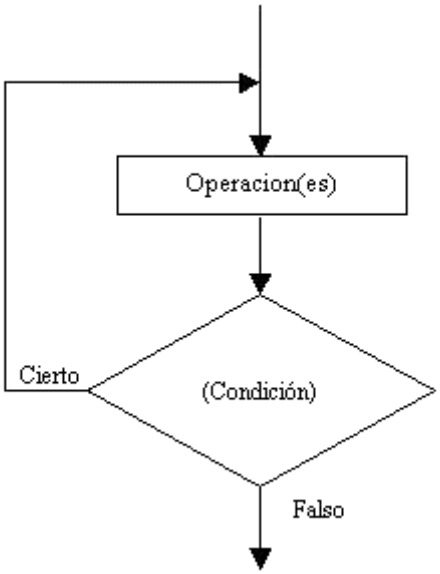
Capítulo 53.- Estructura repetitiva do while – 1

La estructura do while es otra estructura repetitiva del lenguaje C, la cual ejecuta al menos una vez el bloque repetitivo, a diferencia del while o del for que podrían no ejecutar el bloque.

Esta estructura repetitiva se utiliza cuando conocemos de antemano que por lo menos una vez se ejecutará el bloque repetitivo.

La condición de la estructura está abajo del bloque a repetir, a diferencia del while o del for que está en la parte superior.

Representación gráfica:



El bloque de operaciones se repite MIENTRAS la condición sea Verdadera.

Si la condición retorna Falso el ciclo se detiene. En C, todos los ciclos repiten por verdadero y cortan por falso.

Es importante analizar y ver que las operaciones se ejecutan como mínimo una vez.

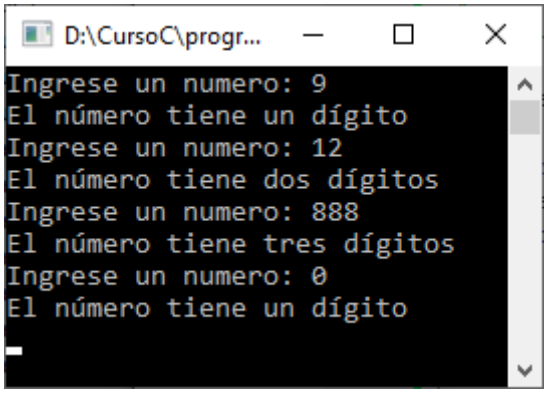
Problema

Escribir un programa que solicite la carga de un número entre 0 y 999, y nos muestre un mensaje de cuantos dígitos tiene el mismo. Finalizar el programa cuando se cargue el valor 0.

```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<locale.h>
4
5   int main()
6   {
7       int numero;
8       setlocale(LC_ALL,"");
9       do
10      {
11          printf("Ingrese un numero: ");
12          scanf("%i", &numero);
```

```
13          if(numero>=100)
14        {
15            printf("El número tiene tres dígitos");
16        }
17        else
18        {
19            if(numero>=10)
20            {
21                printf("El número tiene dos dígitos");
22            }
23            else
24            {
25                printf("El número tiene un dígito");
26            }
27        }
28        printf("\n");
29      }
30      while(numero!=0);
31      getch();
32      return 0;
33  }
34
```

Si ejecutamos este será el resultado:



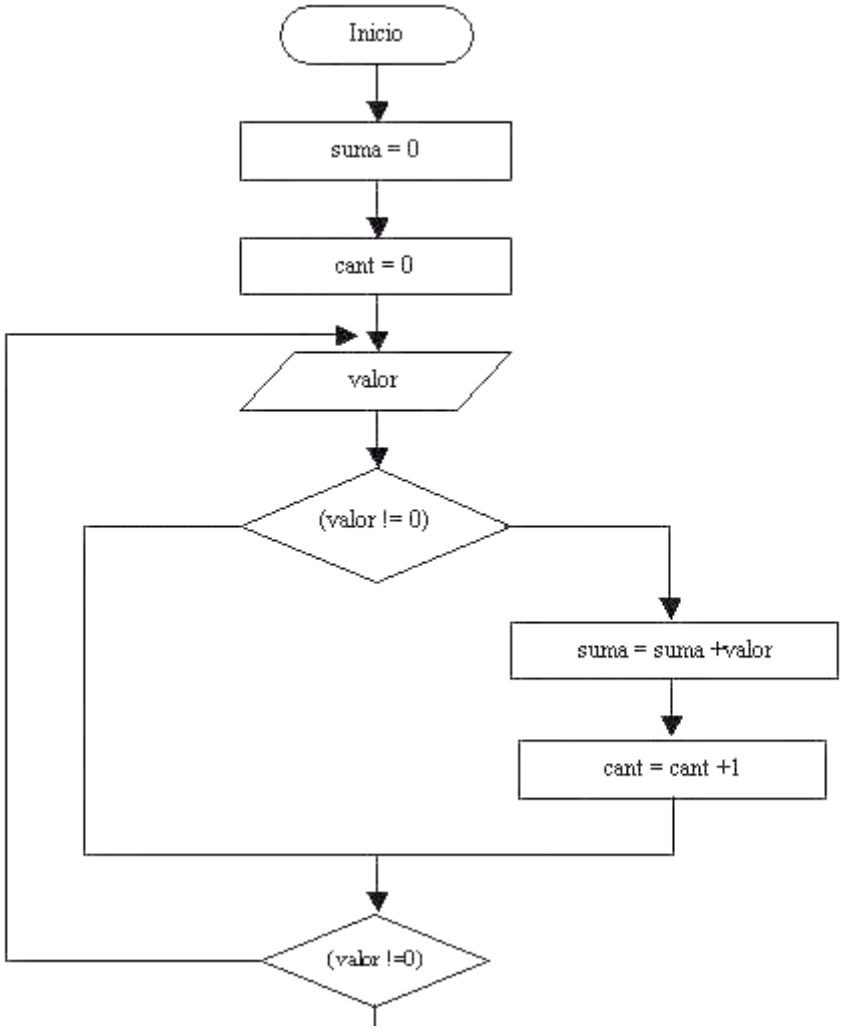
Capítulo 54.- Estructura repetitiva do while – 2

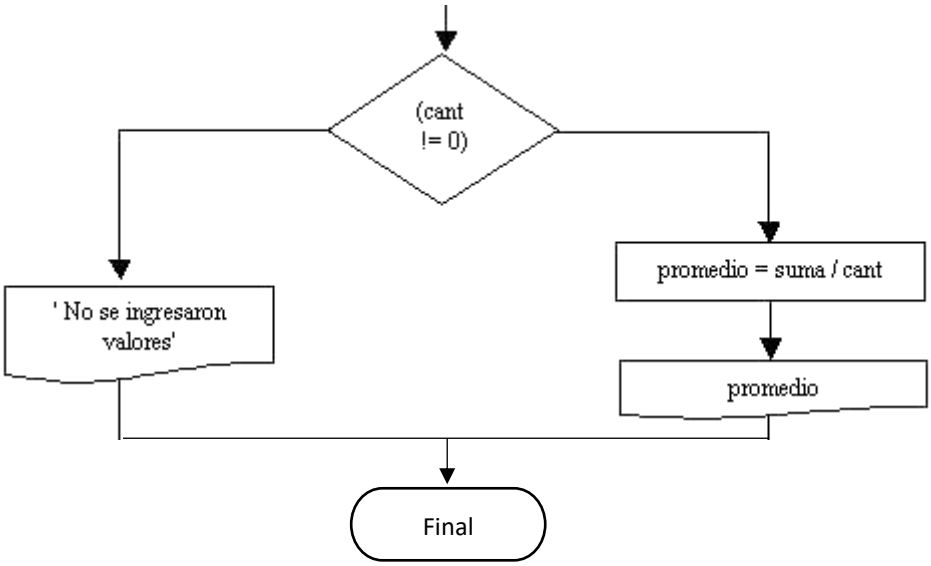
Problema

Escribir un programa que solicite la carga de números por teclado, obtener su promedio. Finalizar la carga de valores cuando se ingresa el valor 0.

Cuando la finalización depende de algún valor ingresado por el operador conviene el empleo de la estructura do while, por lo menos se cargará un valor (en el caso más extremo se carga 0, que indica la finalización de la carga de valores).

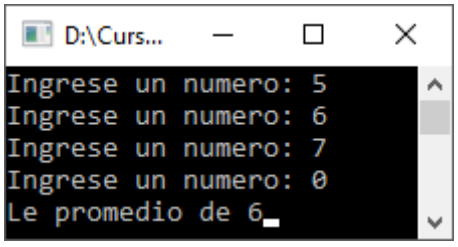
Diagrama de flujo:






```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int suma=0, cant=0;
    int valor;
    do
    {
        printf("Ingrese un numero: ");
        scanf("%i", &valor);
        if (valor!=0)
        {
            suma=suma+valor;
            cant++;
        }
    }
    while (valor!=0);
    if (cant!=0)
    {
        int promedio=suma/cant;
        printf("Le promedio de %i", promedio);
    }
    else{
        printf("No se ingresaron valores");
    }
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



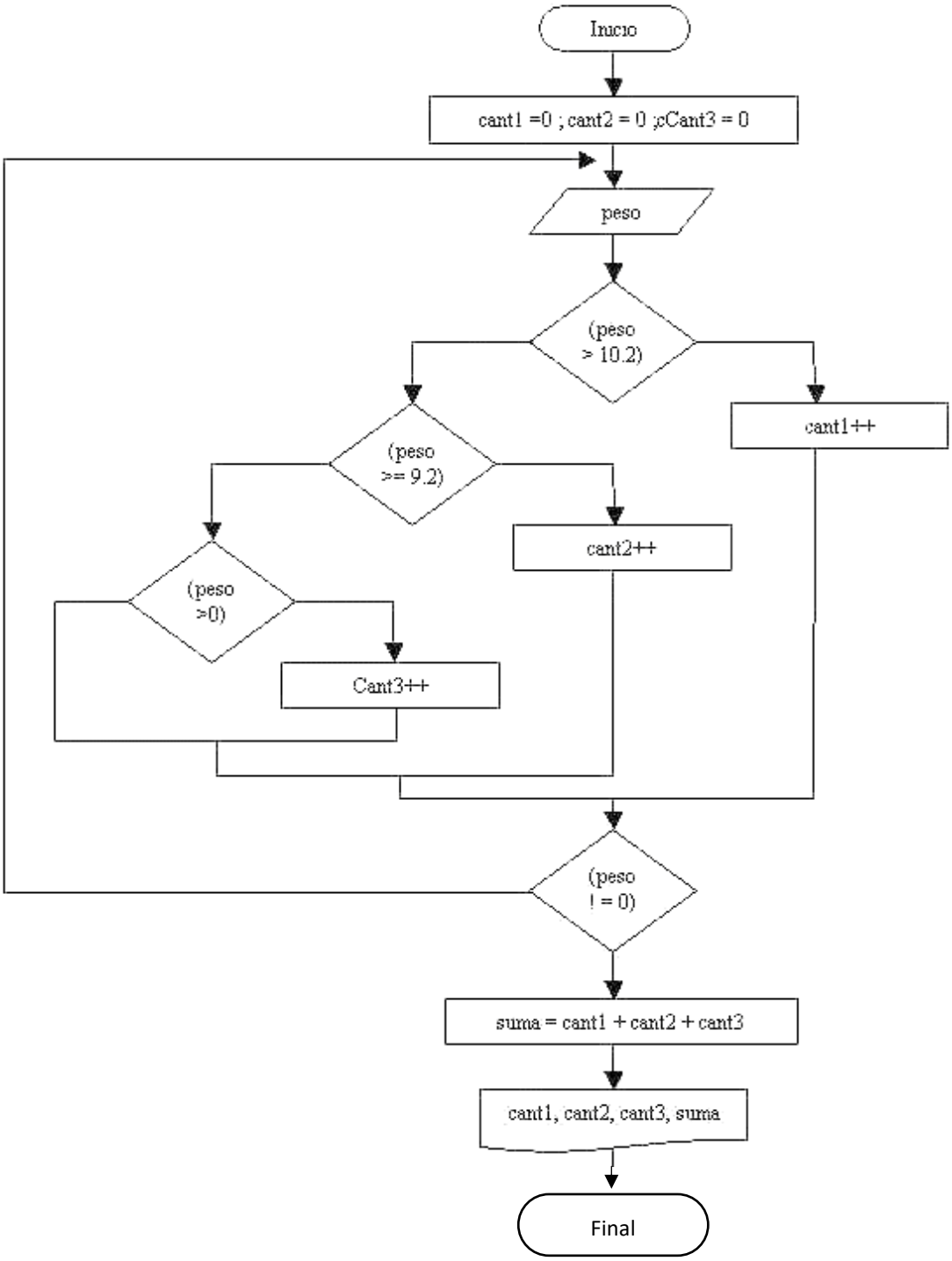
Capítulo 55.- Estructura repetitiva do while – 3

Problema

Realizar un programa que permita ingresar el peso (en kilogramos) de piezas. El proceso termina cuando ingresamos el valor 0. Se debe informar:

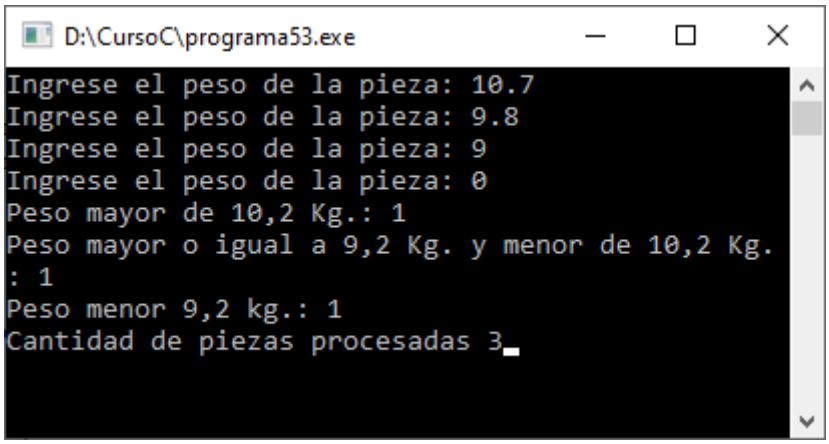
a) ¿Cuántas piezas tiene un peso entre 9.2 Kg. y 10.2 Kg.?, ¿Cuántas con más de 10.2 Kg.? y ¿Cuántas con menos de 9.2 Kg.?
b) La cantidad de piezas procesadas.

Diagrama de flujo:



```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int cant1=0, cant2=0, cant3=0;
    float peso;
    do
    {
      printf("Ingrese el peso de la pieza: ");
      scanf("%f", &peso);
      if (peso>10.2)
      {
          cant1++;
      }
      else
      {
          if (peso>=9.2)
          {
              cant2++;
          }
          else
          {
              if(peso>0)
              {
                  cant3++;
              }
          }
      }
    } while (peso!=0);
    int suma=cant1+cant2+cant3;
    printf("Peso mayor de 10,2 Kg.: %i\n", cant1);
    printf("Peso mayor o igual a 9,2 Kg. y");
    printf(" menor de 10,2 Kg.: %i\n", cant2);
    printf("Peso menor 9,2 kg.: %i\n", cant3);
    printf("Cantidad de piezas procesadas %i", suma);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



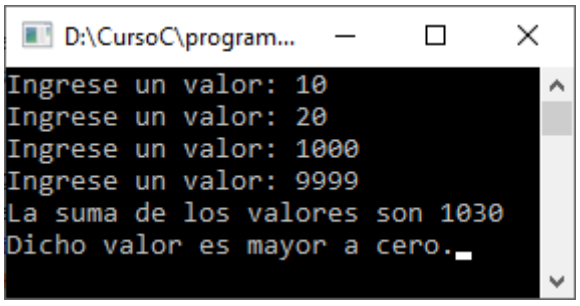
Capítulo 56.- Estructura repetitiva do while – 4

Problema propuesto

Realizar un programa que acumule (sume) valores ingresados por teclado hasta ingresar 9999 (no sumar dicho valor, indica que ha finalizado la carga). Imprimir el valor acumulado e informar si dicho valor es cero, mayor a cero o menor a cero.

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int suma=0;
    int valor;
    do
    {
        printf("Ingrese un valor: ");
        scanf("%i", &valor);
        if (valor!=9999)
        {
            suma=suma+valor;
        }
    }while(valor!=9999);
    printf("La suma de los valores son %i", suma);
    printf("\n");
    if(suma==0){
        printf("Dicho valor es cero.");
    }
    else
    {
        if (suma>0)
        {
            printf("Dicho valor es mayor a cero.");
        }
        else
        {
            printf("Dicho valor es menor a cero");
        }
    }
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 57.- Estructura repetitiva do while – 5

Problema propuesto

En un banco se procesan datos de las cuentas corrientes de sus clientes. De cada cuenta corriente se conoce: número de cuenta y saldo actual. El ingreso de datos debe finalizar al ingresar un valor negativo en el número de cuenta.

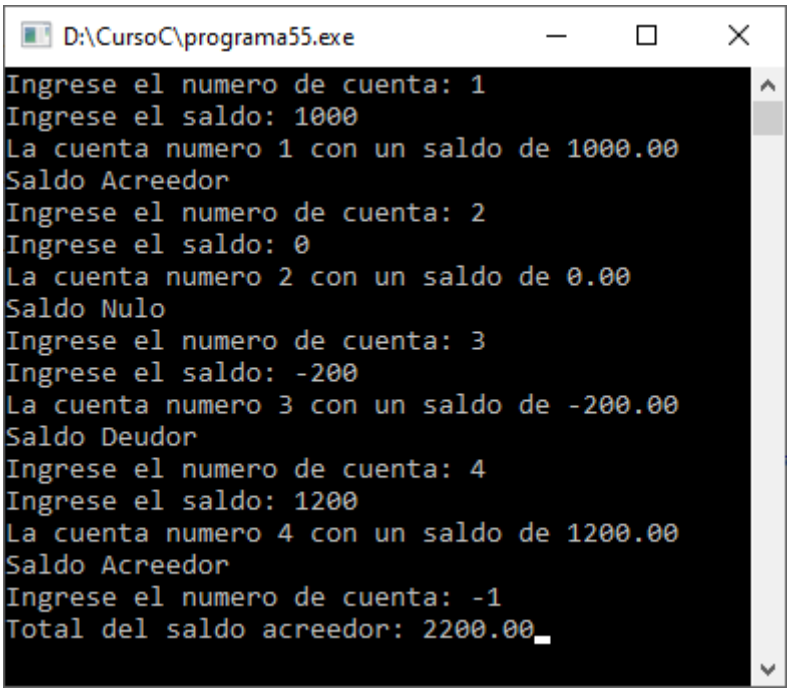
Se pide confeccionar un programa que lea los datos de las cuentas corrientes e informe:

a) De cada cuenta: número de cuenta y estado de la cuenta según su saldo, sabiendo que:
Estado de la cuenta: 'Acreedor' si el saldo es >0.
'Deudor' si el saldo es <0.
'Nulo' si el saldo es =0.

b) La suma total de los saldos acreedores.

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int cuenta;
    float saldo, suma=0;
    do
    {
        printf("Ingrese el numero de cuenta: ");
        scanf("%i", &cuenta);
        if(cuenta>=0)
        {
             printf("Ingrese el saldo: ");
            scanf("%f", &saldo);
            if(saldo>0)
            {
                printf("La cuenta numero %i con un saldo de %.2f", cuenta, saldo);
                printf("\nSaldo Acreedor");
                printf("\n");
                suma=suma+saldo;
            }else
            {
                if (saldo<0)
                {
                    printf("La cuenta numero %i con un saldo de %.2f", cuenta, saldo);
                    printf("\nSaldo Deudor");
                    printf("\n");
                }
                else
                {
                    printf("La cuenta numero %i con un saldo de %.2f", cuenta, saldo);
                    printf("\nSaldo Nulo");
                    printf("\n");
                }
            }
        }

    }while(cuenta>=0);
    printf("Total del saldo acreedor: %.2f", suma);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:


D:\CursoC\programa55.exe
Ingrese el numero de cuenta: 1
Ingrese el saldo: 1000
La cuenta numero 1 con un saldo de 1000.00
Saldo Acreedor
Ingrese el numero de cuenta: 2
Ingrese el saldo: 0
La cuenta numero 2 con un saldo de 0.00
Saldo Nulo
Ingrese el numero de cuenta: 3
Ingrese el saldo: -200
La cuenta numero 3 con un saldo de -200.00
Saldo Deudor
Ingrese el numero de cuenta: 4
Ingrese el saldo: 1200
La cuenta numero 4 con un saldo de 1200.00
Saldo Acreedor
Ingrese el numero de cuenta: -1
Total del saldo acreedor: 2200.00

Capítulo 58.- Estructura de datos tipo vector elementos int y float – 1

Hemos empleado variables numéricas de distinto tipo para el almacenamiento de datos (variables int y float). En este concepto veremos otro tipos de variables que permiten almacenar un conjunto de datos en una única variable.

Un vector en el lenguaje C es una estructura de datos que permite almacenar un CONJUNTO de datos del MISMO tipo.

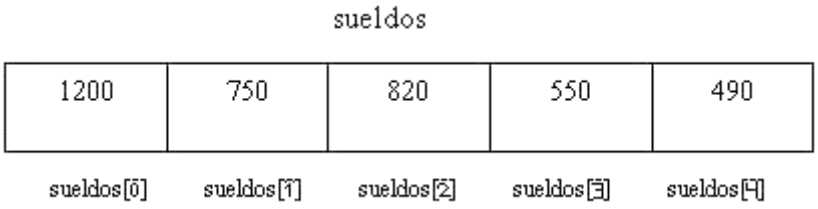
Con un único nombre se define un vector y por medio de un subíndice hacemos referencia a cada elemento del mismo (componente).

Problema

Se desea guardar los sueldos de 5 operarios.

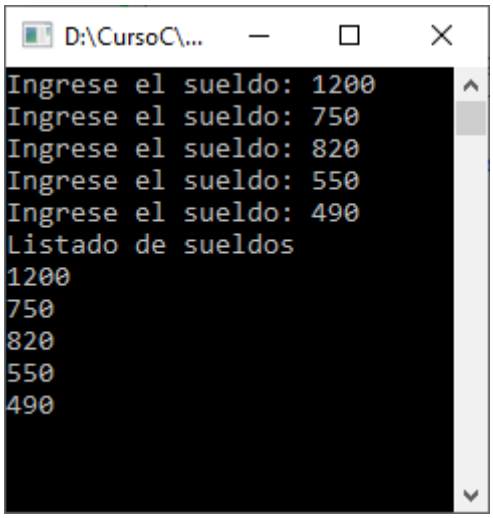
Según lo conocido deberíamos definir 5 variables si queremos tener en un cierto momento los 5 sueldos almacenados en memoria.

Empleando un vector solo se requiere definir un único nombre y accedemos a cada elemento por medio del subíndice.




```
#include<stdio.h>
#include<conio.h>

int main()
{
    int sueldo[5];
    int f;
    // Cargar el vector
    for (f=0; f<5; f++)
    {
        printf("Ingrese el sueldo: ");
        scanf("%i", &sueldo[f]);
    }
    printf("Listado de sueldos\n");
    for (f=0; f<5; f++)
    {
        printf("%i \n", sueldo[f]);
    }

    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Para la declaración de un vector le agregamos corchetes abiertos y cerrados con un valor que indica la cantidad de sueldos que podrá almacenar (en este ejemplo reservamos espacio para cinco sueldos):

```
int sueldos[5];
```

Para cargar cada componente debemos indicar entre corchetes que elemento del vector estamos accediendo.

```
for (f=0; f<5; f++)
{
    printf("Ingrese valor del sueldo:");
    scanf("%i",&sueldos[f]);
}
```

Para definir un comentario en nuestro programa en C debemos anteceder dos caracteres //.

Todo lo que dispongamos después de estos caracteres el compilador no lo tiene en cuenta y solo le es útil al programador para recordar el objetivo de esa parte del algoritmo.

```
//Carga del vector
```

La estructura de programación que más se adapta para cargar en forma completa los elementos del vector es un for, ya que sabemos de antemano la cantidad de valores a cargar, de todos modos no es obligatorio tener que utilizar el for.

Cuando f vale cero estamos accediendo al primer elementos del vector (en nuestro caso sería:

```
scanf("%i",&sueldos[f]);
```

Lo más común es utilizar una estructura repetitiva for para recorrer cada componente del vector.

Utilizar un for nos reduce la cantidad de código, si no utilizo un for debería en forma secuencial implementar el siguiente código:

```
printf("Ingrese valor de la componente:");
scanf("%i",&sueldos[0]);
printf("Ingrese valor de la componente:");
scanf("%i",&sueldos[1]);
printf("Ingrese valor de la componente:");
scanf("%i",&sueldos[2]);
printf("Ingrese valor de la componente:");
scanf("%i",&sueldos[3]);
printf("Ingrese valor de la componente:");
scanf("%i",&sueldos[4]);
```

La impresión de los elementos del vector lo hacemos con un for:

```
for(f=0; f<5; f++)
{
    printf("%i",sueldos[f]);
    printf("\n");
}
```

Siempre que queremos acceder a un elemento del vector deberemos indicar entre corchetes la posición que ocupa en el vector, esta posición empieza por 0, 1, 2, 3, …etc.

Capítulo 59.- Estructura de datos tipo vector elementos int y float – 2

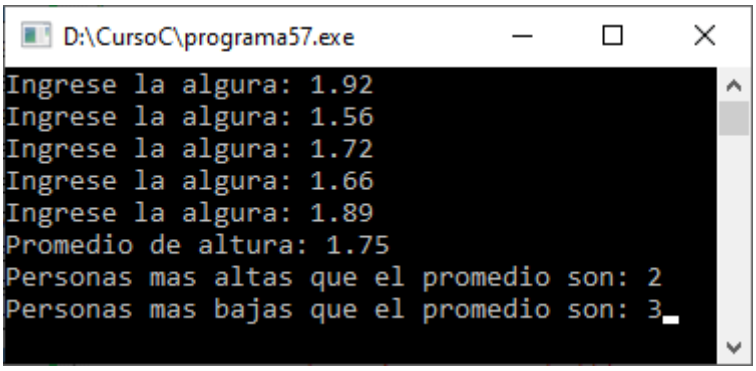
Problema

Definir un vector de 5 componentes de tipo float que representen las alturas de 5 personas.

Obtener el promedio de las mismas. Contar cuantas personas son más altas que el promedio y cuantas más bajas.

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    float alturas[5], suma, promedio;
    int x, altas=0, bajas=0;
    for (x=0; x<5; x++)
    {
        printf("Ingrese la algura: ");
        scanf("%f", &alturas[x]);
    }
    // Obtener promedio
    for (x=0; x<5; x++){
        suma=suma+alturas[x];
    }
    promedio=suma/5;
    printf("Promedio de altura: %.2f", promedio);
    printf("\n");
    // Contar más altas y bajas que el promedio
    for (x=0; x<5; x++)
    {
        if(alturas[x]>promedio)
        {
            altas++;
        }
        else
        {
            if(alturas[x]<promedio)
            {
                bajas++;
            }
        }
    }
    // Imprimir cantidad altas y bajas
    printf("Personas mas altas que el promedio son: %i", altas);
    printf("\n");
    printf("Personas mas bajas que el promedio son: %i", bajas);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:


D:\CursoC\programa57.exe
Ingrese la algura: 1.92
Ingrese la algura: 1.56
Ingrese la algura: 1.72
Ingrese la algura: 1.66
Ingrese la algura: 1.89
Promedio de altura: 1.75
Personas mas altas que el promedio son: 2
Personas mas bajas que el promedio son: 3

Capítulo 60.- Estructura de datos tipo vector elementos int y float – 3

Problema

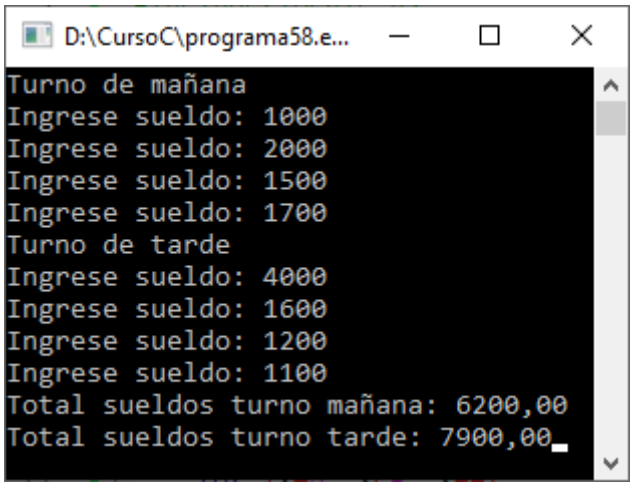
Una empresa tiene dos turnos (mañana y tarde) e los que trabajan 8 empleados (4 por la mañana y 4 por la tarde).

Confeccionar un programa que permita almacenar los sueldos de los empleados agrupados por turno (definir los dos vectores con componentes float).

Imprimir los gastos en sueldos de cada turno.

```
#include<stdio.h>
#include<conio.h>
#include<locale.h>
int main()
{
    setlocale(LC_ALL,"");
    float ma[4], ta[4];
    int x;
    float totalma=0, totalta=0;
    printf("Turno de mañana\n");
    for (x=0; x<4; x++)
    {
       printf("Ingrese sueldo: ");
       scanf("%f", &ma[x]);
    }
    printf("Turno de tarde\n");
    for (x=0; x<4; x++)
    {
       printf("Ingrese sueldo: ");
       scanf("%f", &ta[x]);
    }
    // Sueldos agrupados por turno
    for (x=0; x<4; x++)
    {
        totalma=totalma+ma[x];
        totalta=totalta+ta[x];
    }
    printf("Total sueldos turno mañana: %.2f\n", totalma);
    printf("Total sueldos turno tarde: %.2f", totalta);
    getch();
    return 0;
}

```

Si ejecutamos este será el resultado:


D:\CursoC\programa58.e...
Turno de mañana
Ingrese sueldo: 1000
Ingrese sueldo: 2000
Ingrese sueldo: 1500
Ingrese sueldo: 1700
Turno de tarde
Ingrese sueldo: 4000
Ingrese sueldo: 1600
Ingrese sueldo: 1200
Ingrese sueldo: 1100
Total sueldos turno mañana: 6200,00
Total sueldos turno tarde: 7900,00

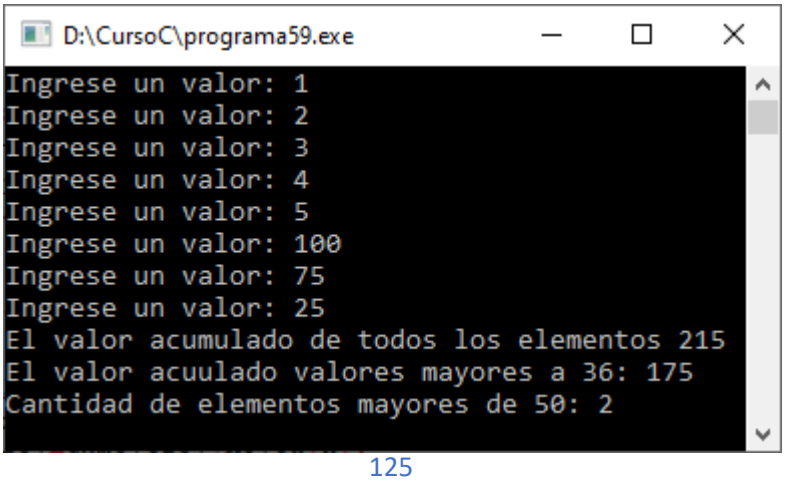
Capítulo 61.- Estructura de datos tipo vector elementos int y float – 4

Problema propuesto

Desarrollar un programa que permita ingresar un vector de 8 elementos, e informe:
El valor acumulado de todos los elementos del vector.
El valor acumulado de los elementos del vector que sean mayores a 36.
Cantidad de valores mayores a 50.

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int valor[8];
    int x, cantidad=0, sumaTotal=0, sumaMayor36=0;
    // Cargar valores al vector
    for(x=0; x<8; x++)
    {
        printf("Ingrese un valor: ");
        scanf("%i", &valor[x]);
    }
    // Leer valores del vector
    for (x=0; x<8; x++){
        sumaTotal=sumaTotal+valor[x];
        if (valor[x]>36)
        {
            sumaMayor36=sumaMayor36+valor[x];
        }
        if(valor[x]>50)
        {
            cantidad++;
        }
    }
    // Imprimir resultados
    printf("El valor acumulado de todos los elementos %i\n", sumaTotal);
    printf("El valor acuulado valores mayores a 36: %i\n", sumaMayor36);
    printf("Cantidad de elementos mayores de 50: %i", cantidad);

    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 62.- Estructura de datos tipo vector elementos int y float – 5

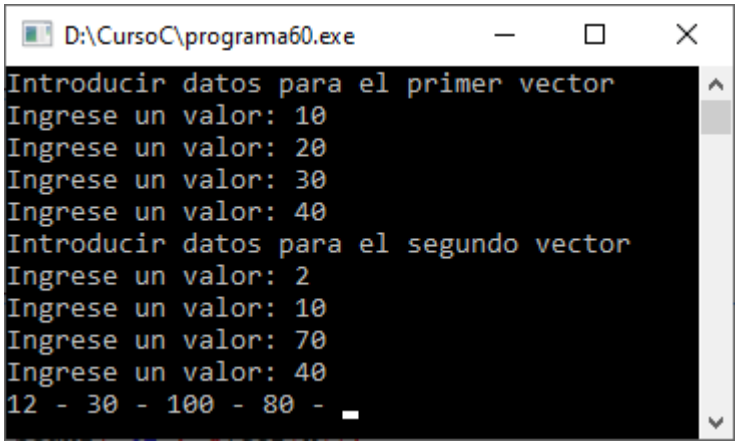
Problema propuesto

Realizar un programa que pida la carga de dos vectores numéricos enteros de 4 elementos. Obtener la suma de los dos vectores, dicho resultado guardarlo en un tercer vector del mismo tamaño. Sumar componente a componente.

vec1	10	20	30	40
vec2	2	10	70	40
vecSuma	12	30	100	80

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int vec1[8], vec2[8], vecSuma[8];
    int x;
    printf("Introducir datos para el primer vector\n");
    for (x=0; x<4; x++)
    {
        printf("Ingrese un valor: ");
        scanf("%i", &vec1[x]);
    }
    printf("Introducir datos para el segundo vector\n");
    for (x=0; x<4; x++)
    {
        printf("Ingrese un valor: ");
        scanf("%i", &vec2[x]);
    }
    // Agregar valores a vecSuma
    for (x=0; x<4; x++)
    {
        vecSuma[x]=vec1[x]+vec2[x];
    }
    // Mostrar los elementos vecSuma
    for (x=0; x<4; x++)
    {
        printf("%i - ", vecSuma[x]);
    }
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 63.- Estructura de datos tipo vector elementos int y float – 6

Problema propuesto

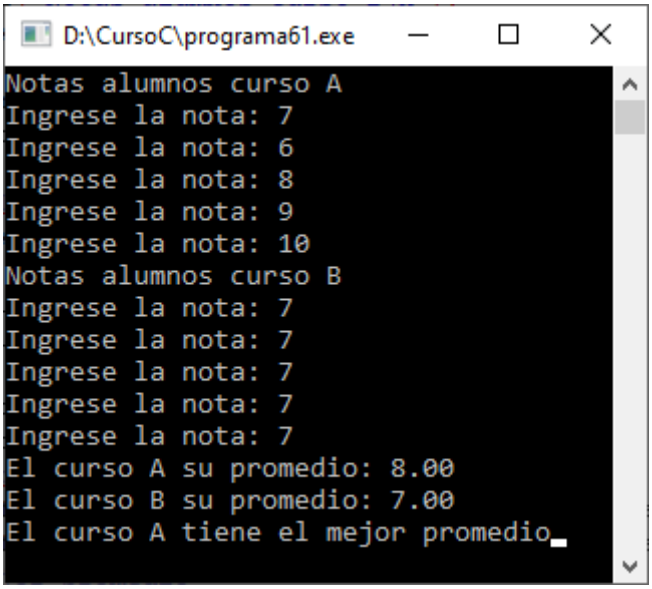
Se tiene las notas del primer parcial de los alumnos de dos cursos, el curso A y el curso B, cada curso cuenta con 5 alumnos.

Realizar un programa que muestre el curso que obtuvo el mayor promedio general.

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int notasA[5], notasB[5];
    float promedioA, promedioB;
    int x,sumaA=0,sumaB=0;
    printf("Notas alumnos curso A\n");
    for (x=0; x<5; x++)
    {
        printf("Ingrese la nota: ");
        scanf("%i", &notasA[x]);
    }
    printf("Notas alumnos curso B\n");
    for (x=0; x<5; x++)
    {
        printf("Ingrese la nota: ");
        scanf("%i", &notasB[x]);
    }
    // Calculo promedio por curso
    for (x=0; x<5; x++)
    {
        sumaA=sumaA+notasA[x];
        sumaB=sumaB+notasB[x];
    }
    promedioA=sumaA/5;
    promedioB=sumaB/5;
    // Imprimir promedios
    printf("El curso A su promedio: %.2f\n", promedioA);
    printf("El curso B su promedio: %.2f\n", promedioB);
    // Mejor promedio
    if(promedioA>promedioB)
    {
        printf("El curso A tiene el mejor promedio");
    }
    else
    {
        if(promedioB>promedioA)
        {
            printf("El curso B tiene el mejor promedio");
        }
        else
        {
            printf("Los dos cursos tienen el mismo promedio");
        }
```

```
47      }
48      getch();
49      return 0;
50  }
51
```

Si ejecutamos este será el resultado:



Capítulo 64.- Estructura de datos tipo vector elementos int y float – 7

Problema propuesto

Cargar un vector de 10 elementos y verificar posteriormente si el mismo está ordenador de menor a mayor.

10 20 60 67 80 120 200 230 600 700

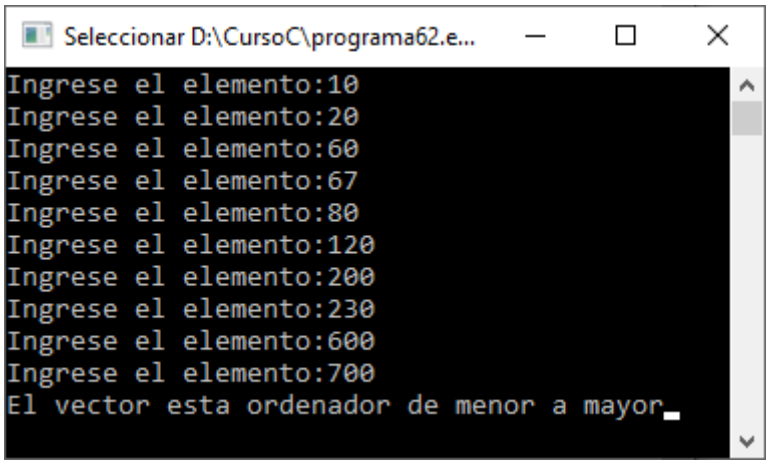
```
#include<stdio.h>
#include<conio.h>

int main()
{
    int vector[10]={10,20,60,67,80,120,200,230,600,700};
    int x, compara=0;
    // Mostrar elementos vectos
    for (x=0; x<10; x++)
    {
        printf("%i - ", vector[x]);
    }
    printf("\n");
    for (x=0; x<9; x++)
    {
      if(vector[x]<vector[x+1])
      {
          compara++;
      }
    }
    // Verificar si el vector esta ordenado
    if (compara==9){
        printf("El vector esta ordenado");
    }
    else
    {
        printf("El vector no esta ordenado");
    }

    getch();
    return 0;
}
```

Otra forma correcta de realizarlo:

```
#include<stdio.h>
#include<conio.h>

int main()
{
   int vec[10];
   int f;
   int orden;
   for (f=0; f<10; f++)
```

```
10      {
11          printf("Ingrese el elemento:");
12          scanf("%i", &vec[f]);
13      }
14      orden=1;
15      for(f=0; f<9; f++)
16      {
17          if(vec[f+1]<vec[f])
18          {
19              orden=0;
20              break;
21          }
22       }
23       if (orden==1)
24       {
25           printf("El vector esta ordenador de menor a mayor");
26       }
27       else
28       {
29           printf("El vector no esta ordenador de menor a mayor");
30       }
31       getch();
32       return 0;
33  }
34
```

Si ejecutamos este será el resultado:



Capítulo 65.- Tipo de datos char – 1

Hasta ahora hemos utilizado variables de tipo entera (int) y real (float).

Utilizar un tipo de dato adecuado hace más eficiente a nuestro programa, no es decuado definir siempre variables de tipo float aunque en estas también podemos almacenar un entero.

Veremos ahora un tercer tipo de dato llamado char.

En una variable de tipo char podemos almacenar un valor entero comprendido entre -128 y 27.

Los valores enteros positivos de las variables tipo char están relacionada con la tabla de caracteres ASCII:

ASCCI	Valor	ASCC I	Valor	ASCCI	Valor	ASCCI	Valor
0	NUL	32	Espacio	64	@	96	`
1		33	!	65	A	97	a
2		34	"	66	B	98	b
3		35	#	67	C	99	c
4		36	$	68	D	100	d
5		37	%	69	E	101	e
6		38	&	70	F	102	f
7		39	'	71	G	103	g
8		40	(	72	H	104	h
9	TAB	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11		43	+	75	K	107	k
12		44	,	76	L	108	l
13		45	-	77	M	109	m
14		46	.	78	N	110	n
15		47	/	79	O	111	o
16		48	0	80	P	112	p
17		49	1	81	Q	113	q
18		50	2	82	R	114	r
19		51	3	83	S	115	s
20		52	4	84	T	116	t
21		53	5	75	U	117	u
22		54	6	86	V	118	v
23		55	7	87	W	119	w
24		56	8	88	X	120	x
25		57	9	89	Y	121	Y
26		58	:	90	Z	122	z
27		59	;	91	[	123	{
28		60	%lt;	92	\	124	|
29		61	=	93	]	125	}
30		62	>	94	^	126	~`
31		63	?	95	_	127	

Si en una variable char almacenamos el valor entero 65 luego podemos imprimir dicho valor como carácter y si vemos en la tabla ASCII dicho valor está asociado al carácter 'A'.

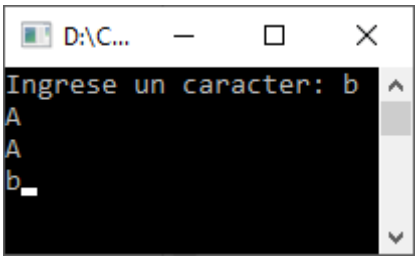
En el lenguaje C una variable char es similar a una variable int pero con la salvedad que puede almacenar un valor entero más pequeño (en el rango de -128 a 127).

Problema

Definir tres variables de tipo char y cargar dos por asignación y la tercera por teclado. Imprimir los valores de las mismas.

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    char letra1=65;
    char letra2='A';
    char letra3;
    printf("Ingrese un caracter: ");
    scanf("%c", &letra3);
    printf("%c", letra1);
    printf("\n");
    printf("%c", letra2);
    printf("\n");
    printf("%c", letra3);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



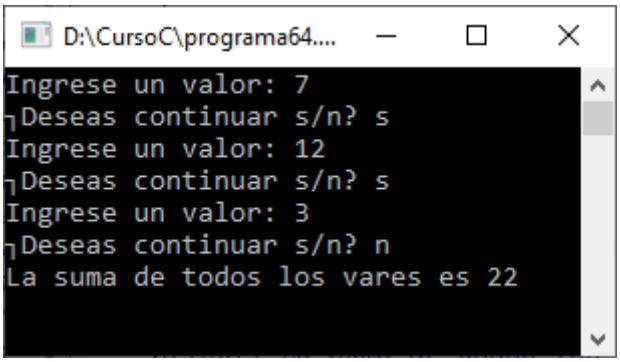
Capítulo 66.- Tipo de datos char – 2

Problema

Confeccionar un programa que permita la carga de valores enteros por teclado. Luego de ingresar el valor mostrar un mensaje por pantalla que pida confirmar al usuario si desea cargar otro valor ingresando los caracteres 's' o 'n'. Mostrar al final la suma de los valores ingresados.

```
#include<stdio.h>
#include<conio.h>

int main()
{
    char continuar;
    int numero, suma=0;
    do
    {
       printf("Ingrese un valor: ");
       scanf("%i", &numero);
       suma=suma+numero;
       printf("¿Deseas continuar s/n? ");
       // Importante el  espacio antes del %C
       // para eliminar el buffer del teclado.
       scanf(" %c", &continuar);
    }while(continuar=='s');
    printf("La suma de todos los vares es %i", suma);
    getch();
    return 0;
}

```

Cuando ejecutamos este será el resultado:



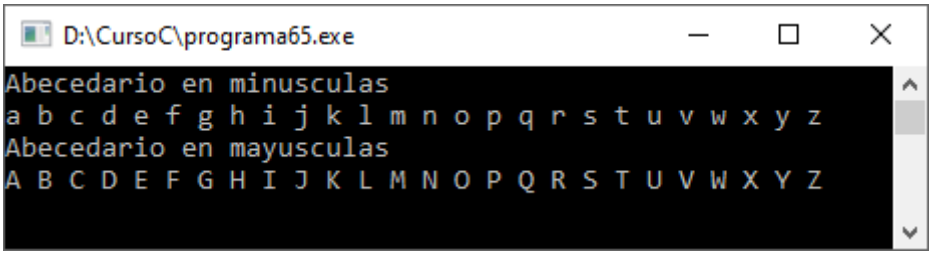
Capítulo 67.- Tipo de datos char – 3

Problema

Mostrar el abecedario de la 'A' a l a'Z' primero en mayúsculas y luego en minúsculas. Utiliza una variable de tipo char dentro de un for.

```
#include<stdio.h>
#include<conio.h>

int main()
{
    char letra;
    printf("Abecedario en minusculas\n");
    for (letra='a'; letra<='z'; letra++)
    {
        printf("%c ", letra);
    }
    printf("\n");
    printf("Abecedario en mayusculas\n");
    for (letra=65; letra<=90; letra++)
    {
        printf("%c ", letra);
    }
    getch();
    return 0;
}
```

Cuando ejecutemos este será el resultado:



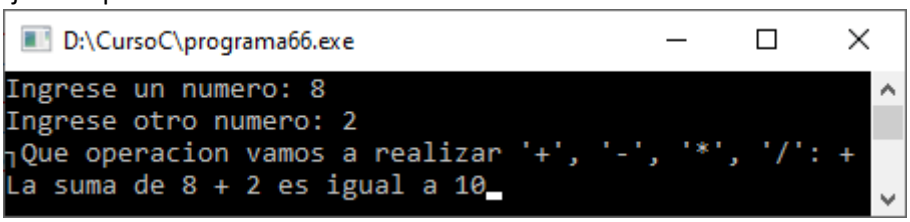
Capítulo 68.- Tipo de datos char – 4

Problema propuesto

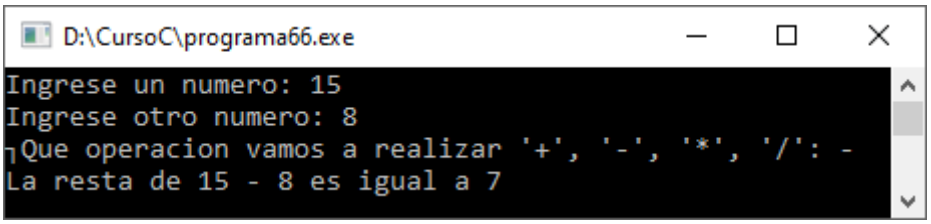
Confeccionar un programa que solicite la carga de dos valores enteros por teclado. Luego solicitar que se cargue alguno de los caracteres: '+', '-', '*' o '/'.
Según el carácter ingresado proceder a mostrar la suma, resta, multiplicación o división de los valores ingresados.

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int num1, num2;
    char operador;
    printf("Ingrese un numero: ");
    scanf("%i", &num1);
    printf("Ingrese otro numero: ");
    scanf("%i", &num2);
    printf("¿Que operacion vamos a realizar");
    printf(" '+', '-', '*', '/': ");
    scanf(" %c", &operador);
    if (operador=='+')
    {
       int suma=num1+num2;
       printf("La suma de %i + %i es igual a %i", num1, num2, suma);
    }
    else
    {
        if(operador=='-')
        {
            int resta=num1-num2;
            printf("La resta de %i - %i es igual a %i", num1, num2, resta);
        }
        else
        {
            if(operador=='*')
            {
                int multiplicar=num1*num2;
                printf("La multplicacion de %i * %i es igual a %i", num1, num2, multiplicar);
            }
            else
            {
                if(operador='/')
                {
                    int division=num1/num2;
                    printf("La division de %i / %i es igual a %i", num1, num2, division);
                }
            }
        }
    }
    getch();
    return 0;
}

```

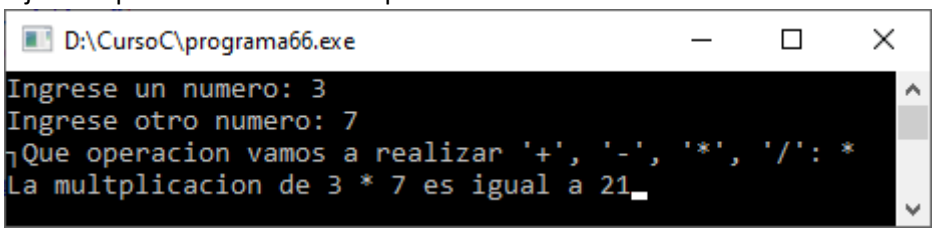
Vamos a ejecutar para realizar una suma:



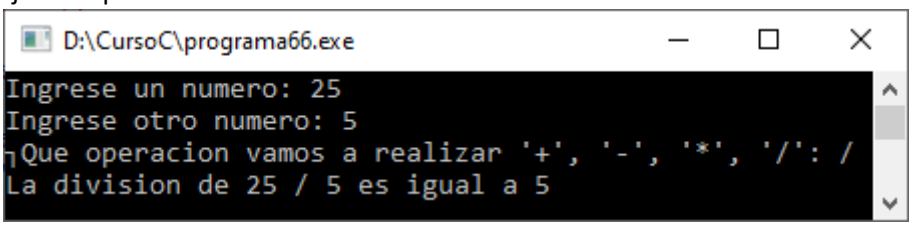
Vamos a ejecutar para realizar una resta:



Vamos a ejecutar para realizar una multiplicación:



Vamos a ejecutar para realizar una división:



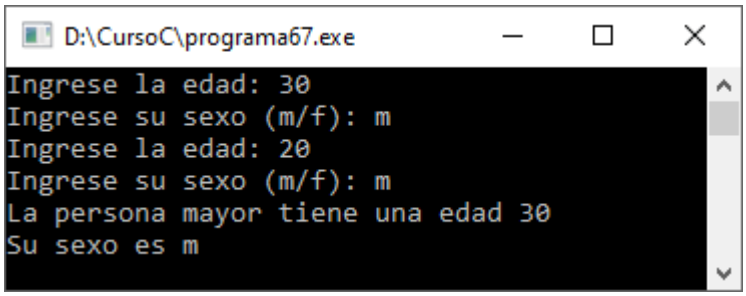
Capítulo 69.- Tipo de datos char – 5

Problema propuesto

Realizar un programa que solicite la carga de la edad y sexo de dos personas. Luego mostrar la edad y el sexo de la persona mayor. Para almacenar el sexo definir una variable de tipo char donde se almacenará el carácter 'm' o 'f' indicando si es del sexo masculino o femenino.

```
#include<stdio.h>
#include<conio.h>

int main()
{
    int edad[2];
    char sexo[2];
    int x;
    for (x=0; x<2; x++)
    {
        printf("Ingrese la edad: ");
        scanf("%i", &edad[x]);
        printf("Ingrese su sexo (m/f): ");
        scanf(" %c", &sexo[x]);
    }
    if(edad[0]>edad[1])
        {
            printf("La persona mayor tiene una edad %i\n", edad[0]);
            printf("Su sexo es %c", sexo[0]);
        }
        else
        {
            printf("La persona mayor tiene una edad %i\n", edad[1]);
            printf("Su sexo es %c", sexo[1]);
        }
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Otra posible solución es:

```
#include<stdio.h>
#include<conio.h>

int main()
{
   int edad1,edad2;
   char sexo1,sexo2;
   printf("Ingrese edad de la persona:");
   scanf("%i",&edad1);
   printf("Sexo de la persona [m/f]:");
   scanf(" %c",&sexo1);
   printf("Ingrese edad de la persona:");
   scanf("%i",&edad2);
```

```
   printf("Sexo de la persona [m/f]:");
   scanf(" %c",&sexo2);
   if (edad1>edad2)
   {
      printf("La edad mayor es:");
      printf("%i",edad1);
      printf("\n");
      if (sexo1=='m')
      {
         printf("sexo:masculino");
      }
      else
      {
         if (sexo1=='f')
         {
            printf("sexo:femenino");
         }
      }
   }
   else
   {
      if (edad2>edad1)
      {
         printf("La edad mayor es:");
         printf("%i",edad2);
         printf("\n");
         if (sexo2=='m')
         {
            printf("sexo:masculino");
         }
         else
         {
            if (sexo2=='f')
            {
               printf("sexo:femenino");
            }
         }
      }
      else
      {
         printf("Tienen la misma edad.");
      }
   }
   getch();
   return 0;
}
```

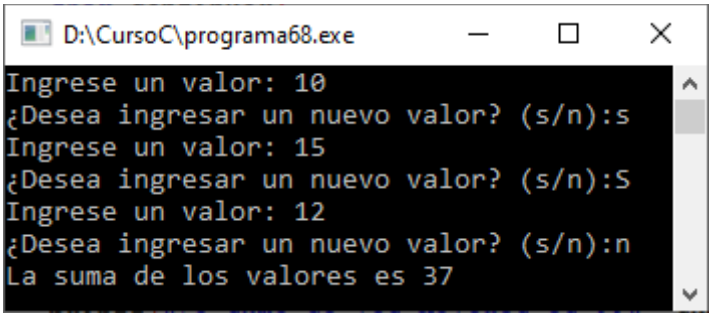
Capítulo 70.- Tipo de datos char – 6

Problema propuesto

Realizar la carga de valores enteros por teclado y sumarlos. Cada vez que se carga un valor pedir al operador que ingrese si quiere cargar otro valor ingresando un 's' o 'S' (minúsculas o mayúsculas).

```
#include<stdio.h>
#include<conio.h>
#include<locale.h>
int main()
{
    setlocale(LC_ALL, "");
    int valor, suma=0;
    char continuar;
    do
    {
        printf("Ingrese un valor: ");
        scanf("%i", &valor);
        suma=suma+valor;
        printf("¿Desea ingresar un nuevo valor? (s/n):");
        scanf(" %c", &continuar);
    }while(continuar=='s' || continuar=='S');
    printf("La suma de los valores es %i", suma);
    getch();
    return 0;
}
```

Si ejecutamos este este será el resultado:



Capítulo 71.- Cadena o vectores de caracteres en C (elementos de tipo char) – 1

Ahora en este concepto veremos como se puede almacenar un conjunto de caracteres relacionados, como por ejemplo cuando necesitamos almacenar en una variable un apellido y un nombre.

Hasta ahora podemos resolver problemas que requieran la carga de valores de tipo int (enteros), float (reales) y char (un carácter ASCII individual)

Si queremos almacenar una cadena de caracteres en el lenguaje C debemos definir un vector con componentes de tipo char.

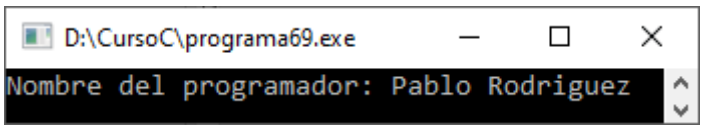
Para definir un vector de caracteres en C debemos indicar entre corchetes la cantidad de caracteres a reservar y tener en cuenta que uno de esas posiciones se utilizará como carácter de control, es decir que si tenemos que almacenar 10 caracteres el vector lo definiremos de 11 caracteres.

Problema

Definir una variable para almacenar el nombre y apellidos del programador. Mostrar dicho nombre en pantalla.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   int main()
5   {
6       char programador[16]="Pablo Rodriguez";
7       printf("Nombre del programador: ");
8       printf("%s", programador);
9       getch();
10      return 0;
11  }
12
```

Si ejecutamos este será el resultado:



Debemos definir una variable de tipo vector con elementos de tipo char:

```
char programador[16]="Pablo Rodriguez";
```

El tamaño 16 no es un capricho, si contamos la cantidad de caracteres que hay entre comillas incluyendo el espacio en blanco veremos que es 15. Luego recordar que debe haber un carácter extra en el vector donde se indica que ahí finalizan los datos de la cadena. No hay problemas si el vector reservas más de 16 caracteres, en cambio se generarán errores inesperados si el vector tiene menos de 16 caracteres.

```
componente   [0]  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]  [9]  [10] [11] [12]
[13] [14] [15]


programador  'P'  'a'  'b'  'l'  'o'  ' '  'R'  'o'  'd'  'r'  'i'  'g'  'u'
'e'  'z'  '\0'
```

En memoria se almacenan en cada posición del vector el carácter respectivo y cuando se ejecuta el programa y se muestran los datos mediante la función printf se muestran todos hasta que encuentra en carácter NULL que coincide con el carácter ASCII 0.

Si modificamos el programa y solo guardamos "Pablo" sin modificar el tamaño del vector luego en memorita tenemos:

```
    char programador[16]="Pablo";
```

```
componente   [0]  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]  [9]  [10] [11] [12]
[13] [14] [15]


programador  'P'  'a'  'b'  'l'  'o'  '\0' ''   ''   ''   ''   ''   ''   ''   ''
''   ''
```

Lo que se almacena luego del carácter '\0' no nos importa y se lo considera basura (puede haber cualquier carácter ASCII).

Para imprimir un vector de caracteres con la función printf debemos indicar en la cadena de formato de carácter s (string):

```
printf("%s",programador);
```

Capítulo 72.- Cadena o vectores de caracteres en C (elementos de tipo char) – 2

Carga de teclado de una cadena

Para cargar por teclado una cadena de caracteres debemos emplear por ahora la función gets.

La función gets tiene como parámetro el nombre de la variable:

```
gets(nombre del vector);
```

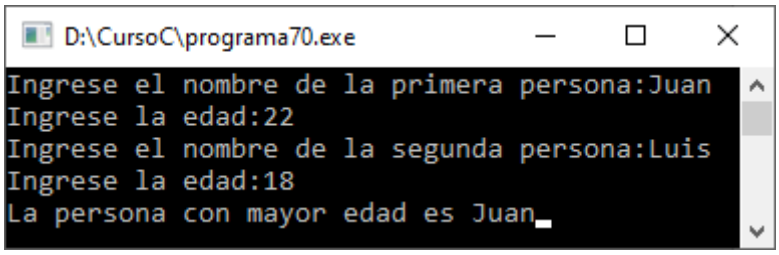
El problema de esta función que si el operador carga más caracteres que los reservados en la variable produce errores inesperados. Más adelante veremos otras soluciones de carga de cadena de caracteres.

Problema

Cargar el nombre de dos personas y sus edades. Mostrar el nombre de la persona que tiene mayor edad. Los nombres de las personas no superan los 20 caracteres.

```
#include<stdio.h>
#include<conio.h>

int main()
{
    char nombre1[21];
    char nombre2[21];
    int edad1;
    int edad2;
    printf("Ingrese el nombre de la primera persona:");
    gets(nombre1);
    printf("Ingrese la edad:");
    scanf("%i", &edad1);
    // Liberamos el Buffer del teclado.
    fflush(stdin);
    printf("Ingrese el nombre de la segunda persona:");
    gets(nombre2);
    printf("Ingrese la edad:");
    scanf("%i", &edad2);
    if (edad1>edad2)
    {
        printf("La persona con mayor edad es %s", nombre1);
    }
    else
    {
        if(edad2>edad1)
        {
            printf("La persona con mayor edad es %s", nombre2);
        }
        else
        {
            printf("Tienen la misma edad %s y %s", nombre1, nombre2);
        }
    }
    getch();
    return 0;
}

```

Si ejecutamos este será el resultado:



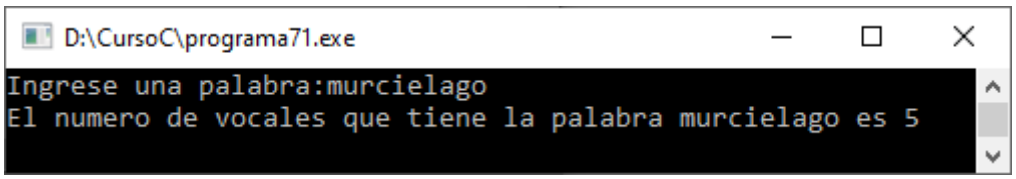
Capítulo 73.- Cadena o vectores de caracteres en C (elementos de tipo char) – 3

Problema

Ingresar por teclado una palabra en minúsculas. Mostrar por pantalla la cantidad de vocales que tiene dicha palabra.

```
#include<stdio.h>
#include<conio.h>

int main()
{
    char palabra[30];
    int x=0, cant=0;
    printf("Ingrese una palabra:");
    gets(palabra);
    while (palabra[x]!='\0')
    {
        if (palabra[x]=='a' || palabra[x]=='e' || palabra[x]=='i'
            ||palabra[x]=='o' || palabra[x]=='u')
        {
           cant++;
        }
        x++;
    }
    printf("El numero de vocales que tiene la palabra %s es %i", palabra, cant);
    getch();
    return 0;
}

```

Cuando ejecutemos este será el resultado:



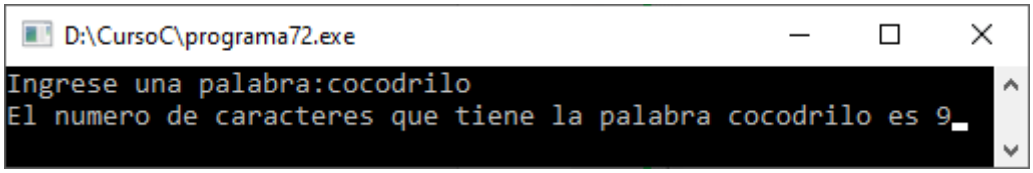
Capítulo 74.- Cadena o vectores de caracteres en C (elementos de tipo char) – 4

Problema propuesto

Ingresar una palabra por teclado. Mostrar por pantalla la palabra y la cantidad de caracteres que tiene dicha palabra.

```
#include<stdio.h>
#include<conio.h>

int main()
{
    char palabra[30];
    int x=0;
    printf("Ingrese una palabra:");
    gets(palabra);
    while (palabra[x]!='\0')
    {
        x++;
    }
    printf("El numero de caracteres que tiene la palabra %s es %i", palabra, x);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



También es correcto de la siguiente forma:

```
#include<stdio.h>
#include<conio.h>

int main()
{
    char palabra[30];
    int x=0;
    printf("Ingrese una palabra:");
    gets(palabra);
    while (palabra[x]!='\0')
        x++;
    printf("El numero de caracteres que tiene la palabra %s es %i", palabra, x);
    getch();
    return 0;
}
```

Se pueden eliminar las llaves porque solo tiene una instrucción, también funciona con los 'if'.

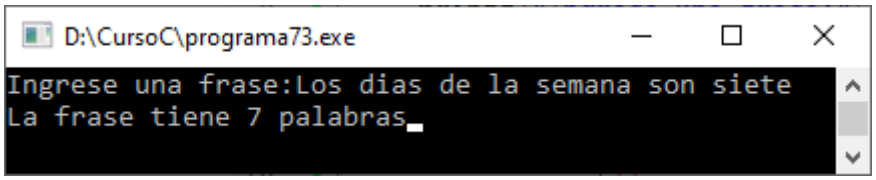
Capítulo 75.- Cadena o vectores de caracteres en C (elementos de tipo char) – 5

Problema propuesto

Ingresar por teclado una oración de 200 caracteres. Se sabe que el operador ingresa solo un carácter en blanco entre palabras. Imprimir por pantalla la cantidad de palabras que tiene la oración.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   int main()
5   {
6       char frase[201];
7       int x=0,cant=0;
8       printf("Ingrese una frase:");
9       gets(frase);
10      while (frase[x]!='\0')
11      {
12          if (frase[x]==' ')
13              cant++;
14          x++;
15      }
16      printf("La frase tiene %i palabras", cant+1);
17      getch();
18      return 0;
19  }
20
```

Si ejecutamos este será el resultado:



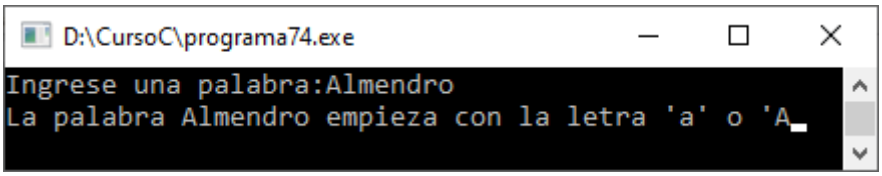
Capítulo 76.- Cadena o vectores de caracteres en C (elementos de tipo char) – 6

Problema propuesto

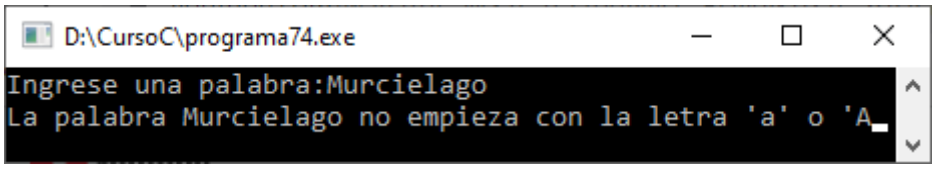
Confeccionar un programa que permita ingresar una palabra y luego muestre un mensaje si la misma comienza con la vocal 'A' o 'a'.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   int main()
5   {
6       char palabra[50];
7       printf("Ingrese una palabra:");
8       gets(palabra);
9       if (palabra[0]=='a' || palabra[0]=='A')
10          printf("La palabra %s empieza con la letra 'a' o 'A", palabra);
11      else
12          printf("La palabra %s no empieza con la letra 'a' o 'A", palabra);
13      getch();
14      return 0;
15  }
```

Vamos a ejecutar introduciendo una palabra que empieza por la letra 'a' o 'A'.



Ahora vamos a introducir una palabra que no empieza por la letra 'a' o 'A'.



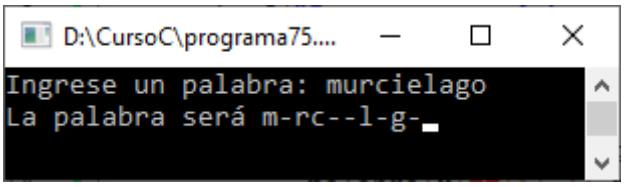
Capítulo 77.- Cadena o vectores de caracteres en C (elementos de tipo char) – 7

Problema propuesto

Permitir el ingreso de una palabra en minúsculas por teclado. Cambiar todas las vocales por un carácter de guión: '-'.

```
#include<stdio.h>
#include<conio.h>
#include<locale.h>
int main()
{
    setlocale(LC_ALL, "");
    char palabra[20];
    int x=0;
    printf("Ingrese un palabra: ");
    gets(palabra);
    while(palabra[x]!='\0')
    {
        if (palabra[x]=='a' || palabra[x]=='e'||
            palabra[x]=='i' || palabra[x]=='o' || palabra[x]=='u')
        {
            palabra[x]='-';
        }
        x++;
    }
    printf("La palabra será %s", palabra);

    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 78.- Funciones que facilitan el trabajo con cadenas de caracteres – string.h – 1

Hemos visto en el capítulo anterior que en el lenguaje C para trabajar con cadenas de caracteres debemos definir un vector de caracteres de un tamaño definido, por ejemplo:

char palabra[31];

En la línea anterior estamos definiendo un vector de caracteres que nos permite almacenar hasta 30 caracteres (recordar que una posición del vector se requiere para la terminación de cadena '\0').

Desarrollamos anteriormente un algoritmo para contar la cantidad de letras que almacena una cadena de caracteres, básicamente recorrimos mediante un while y contamos cada carácter hasta que encontramos la terminación de cadena '\0'.

Hay muchos algoritmos de uso común para trabajar con cadenas de caracteres, el lenguaje C viene con un conjunto de funciones que nos permiten administrar las cadenas de caracteres.

Para facilitar el trabajo con las cadenas de caracteres debemos incluir el archivo string.h y a partir de esto usar sus funcionalidades:

#include<string.h>

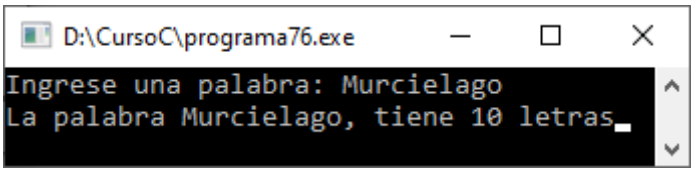
Función: strlen

problema

Ingresar por teclado una palabra. Mostrar por pantalla la cantidad de letras que tiene.

```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<string.h>
4
5   int main()
6   {
7       char palabra[31];
8       int cantLetras;
9       printf("Ingrese una palabra: ");
10      gets(palabra);
11      cantLetras=strlen(palabra);
12      printf("La palabra %s, tiene %i letras", palabra, cantLetras);
13      getch();
14      return 0;
15  }
16
```

Si ejecutamos este será el resultado:



Capítulo 79.- Funciones que facilitan el trabajo con cadenas de caracteres – string.h – 2

#incluede<string.h>

Función: strcmp

Hasta ahora no desarrollamos programas que comparen el contenido de dos cadenas de caracteres. Lo primero que hay que decir que no están definidos los operadores relacionales ==, >. <, etc.

Es incorrecto tratar de comparar si dos cadenas son iguales.

```
if (nombre1==nombre2)
{
    printf("Son iguales los nombres")
}
```

Podemos implementar nosotros un análisis dentro de un while y comparar carácter a carácter, pero este algoritmo no es tan fácil y cómodo de implementar cada vez que tenemos que comparar dos cadenas.

Entonces el lenguaje C tiene otra función llamada strcmp que le pasamos dos cadenas a comparar y nos retorna un entero:

```
int strcmp(cadena1,cadena2)
```

Retorna un cero si las dos cadenas son exactamente iguales.

Retorna un valor mayor a cero si la cadena1 es mayor alfabéticamente que la segunda.

Retorna un valor menor a cero si la cadena2 es mayor alfabéticamente que la primera.

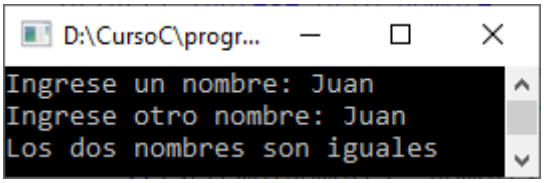
Problema

Ingresar dos nombres por teclado. Mostrar un mensaje si son iguales y si no mostrar el que es mayor alfabéticamente.

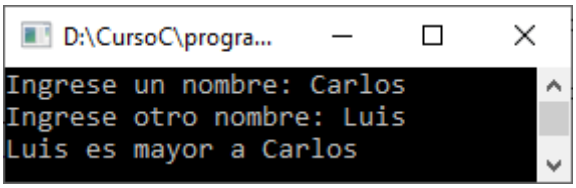
```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<string.h>
4
5   int main()
6   {
7       char nombre1[31];
8       char nombre2[31];
9       printf("Ingrese un nombre: ");
10      gets(nombre1);
11      printf("Ingrese otro nombre: ");
12      gets(nombre2);
13      if (strcmp(nombre1, nombre2)==0)
14          printf("Los dos nombres son iguales");
15      else
16          if(strcmp(nombre1, nombre2)>0)
17              printf( "%s es mayor a %s", nombre1, nombre2);
18          else
```

```
19              printf( "%s es mayor a %s", nombre2, nombre1);
20      getch();
21      return 0;
22  }
```

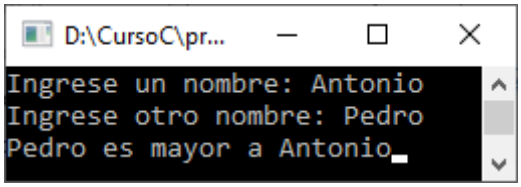
Vamos a ejecutar poniendo los nombres Juan y Juan:



Vamos a ejecutar poniendo los nombres Carlos y Luis.



Vamos a ejecutar poniendo los nombres Antonio y Pedro.



Funcionamiento interno de strcmp

Definimos deos cadenas de 7 elementos y las analizamos:

```
char cadena1[7]="Bien";
char cadena2[7]="Bueno";
```

El contenido de los componentes de cada una de las cadenas son las siguientes.

componente	[0]	[1]	[2]	[3]	[4]	[5]	[6]
cadena1	'B'	'i'	'e'	'n'	'\0'		
cadena2	'B'	'u'	'e'	'n'	'o'	'\0'	

El contenido de los componentes de cada una de las cadenas (en valores ASCII) es el siguiente:

componente	[0]	[1]	[2]	[3]	[4]	[5]	[6]
cadena1	66	105	101	110	0		
cadena2	66	117	101	110	111	0	

Seguidamente llamamos a la función strcmp, la cual devuelve un valor entero que le asignamos a una variable resultado:

```
int resultado=strcmp(cadena1,cadena2);
```

Internamente la función hace lo siguiente:

Resta el valor ASCII del primer componente de la cadena1 (66) el valor del primer componente cadena2 (66). Como el resultado es cero (0), continúa con el siguiente componente. Resta al valor del segundo componente de la cadena 1 (105) el valor del segundo componente de la cadena 2 (117), como el resultado es diferente a cero no sigue comparando y devuelve ese valor (-12):

componente	[0]	[1]	[2]	[3]	[4]	[5]
cadena1	66	105	101	110	0	
cadena2	66	117	101	110	111	0
Resultado	0	-12				

Esa es la explicación de porque strcmp retorna 0 si son iguales las dos cadenas y un valor mayor a 0 si la primera cadena es mayor a la segunda cadena y viceversa.

Capítulo 80.- Funciones que facilitan el trabajo con cadenas de caracteres – string.h – 3

#include<string.h>

Función: strcpy

Hemos visto que si necesitamos comparar dos string (string es sinónimo de cadena de caracteres o vectores) no tenemos disponibles los operadores relacionales ==, >, etc.

Si necesitamos copiar un string en otro tampoco funciona el operador de asignación =, para resolver este problema el lenguaje C proporciona la función srtcpy.

La función strcpy tiene dos parámetros de tipo string, el primer parámetro se le copia el string del segundo parámetro.

strcpy(cadena1, cadena2)

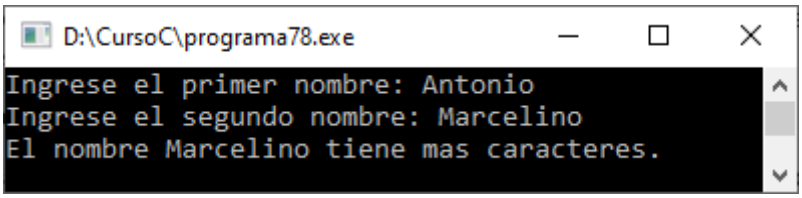
Es decir que en cadena1 se copia el contenido de cadena2, si ya tenía información previa cadena1 se le borra el contenido anterior.

Problema

Cargar por teclado dos nombres de personas que tengan distinta cantidad de caracteres. Almacenar en un tercer vector de caracteres el nombre que tenga más caracteres. Luego imprimir dicho vector.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

int main()
{
    char nombre1[31];
    char nombre2[31];
    char nombreLargo[31];
    printf("Ingrese el primer nombre: ");
    gets(nombre1);
    printf("Ingrese el segundo nombre: ");
    gets(nombre2);
    if (strlen(nombre1)>strlen(nombre2))
    {
        strcpy(nombreLargo, nombre1);
    }
    else
    {
        strcpy(nombreLargo, nombre2);
    }
    printf("El nombre %s tiene mas caracteres.", nombreLargo);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Definimos tres string, dos los cargamos por teclado y el tercero lo inicializamos mediante la función strcpy:

```
char nombre1[31];
char nombre2[31];
char nombreLargo[31];
```

Mediante la función strlen verificamos cual de los dos string tiene más caracteres y copiamos en la variable de tipo string nombreLargo el contenido del nombre1 o nombre2 según corresponda.

```
if (strlen(nombre1)>strlen(nombre2))
{
    strcpy(nombreLargo,nombre1);
}
else
{
    strcpy(nombreLargo,nombre2);
}
```

Tener en cuenta que no existe la asignación:

```
nombreLargo=nombre1;
```

Si en un problema necesitamos dejar un string vacío podemos utilizar la función strcpy pasando un string sin caracteres (no debe haber ni un espacio en blanco).

```
strcpy(cadena,"");
```

Otra forma de hacer lo mismo sin utilizar la función strcpy es asignar el terminador de candena a la primera posición del string:

cadena[0]='\0';

Capítulo 81.- Funciones que facilitan el trabajo con cadenas de caracteres – string.h – 4

#include<string.h>

Función: strcat

Si necesitamos agregar a un string otro string podemos utilizar la función strcat.

La función strcat tiene dos parámetros de tipo string, al primer parámetro se le añade o agrega al final el string del segundo parámetro, es obligatorio que el primer parámetro esté inicializado:

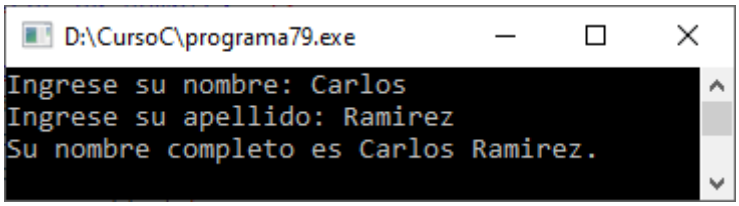
```
strcat(cadena1,cadena2)
```

Problema

Cargar por teclado en dos variables de tipo string el nombre y el apellido de una persona. Definir un tercer string y guardar la concatenación del nombre y apellido.

```
1  #include<stdio.h>
2  #include<conio.h>
3  #include<string.h>
4
5  int main()
6  {
7      char nombre[31];
8      char apellido[31];
9      char nomape[61];
10     printf("Ingrese su nombre: ");
11     gets(nombre);
12     printf("Ingrese su apellido: ");
13     gets(apellido);
14     strcpy(nomape, nombre);
15     strcat(nomape, " ");
16     strcat(nomape, apellido);
17     printf("Su nombre completo es %s.", nomape);
18     getch();
19     return 0;
20 }
21
```

Si ejecutamos este será el resultado:



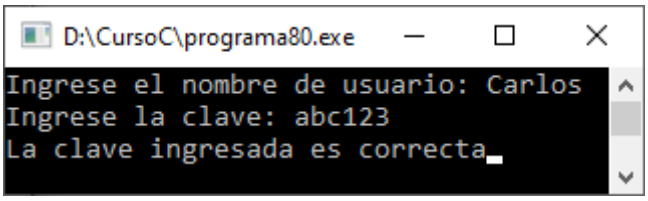
Capítulo 82.- Funciones que facilitan el trabajo con cadenas de caracteres – string.h – 5

Problema propuesto

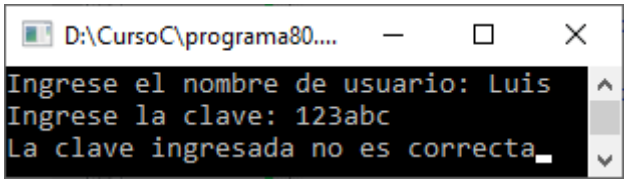
Confeccionar un programa que pida ingresar el nombre de usuario y clave en dos string. Mostrar un mensaje de "Correcto" si se ingresa como clave la cadena "abc123".

```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<string.h>
4
5   int main()
6   {
7       char nombreUsuario[31];
8       char clave[10];
9       printf("Ingrese el nombre de usuario: ");
10      gets(nombreUsuario);
11      printf("Ingrese la clave: ");
12      gets(clave);
13      if (strcmp(clave, "abc123")==0)
14          printf("La clave ingresada es correcta");
15      else
16          printf("La clave ingresada no es correcta");
17      getch();
18      return 0;
19  }
20
```

Ejecutamos introduciendo la clave correcta:



Ejecutamos de nuevo introduciendo una clave incorrecta:



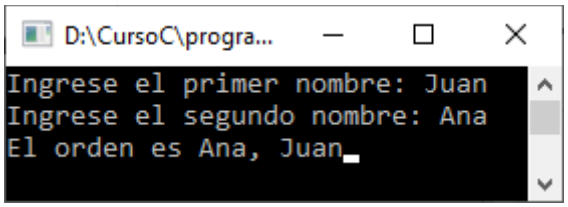
Capítulo 83.- Funciones que facilitan el trabajo con cadenas de caracteres – string.h – 6

Problema propuesto

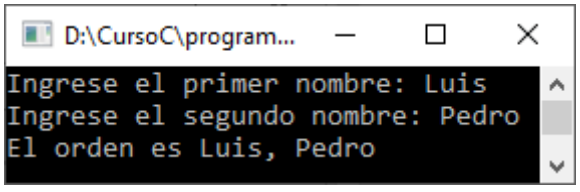
Ingresar por teclado dos nombres de personas y luego mostrarlas ordenados alfabéticamente.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

int main()
{
    char nombre1[21];
    char nombre2[21];
    printf("Ingrese el primer nombre: ");
    gets(nombre1);
    printf("Ingrese el segundo nombre: ");
    gets(nombre2);
    if (strcmp(nombre1, nombre2)<0)
        printf("El orden es %s, %s", nombre1, nombre2);
    else
        printf("El orden es %s, %s", nombre2, nombre1);
    getch();
    return 0;
}
```

Vamos a ejecutar introduciendo los nombres Juan y Ana.



Ejecutamos de nuevo introduciendo los nombres Luis y Pedro.



Capítulo 84.- Funciones que facilitan el trabajo con cadenas de caracteres – string.h – 7

Problema propuesto

Ingresar por teclado dos nombres de personas y luego mostrarlas primero el que tiene más caracteres y luego el que tiene menos caracteres.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<locale.h>

int main()
{
    setlocale(LC_ALL, "");
    char nombre1[21];
    char nombre2[21];
    printf("Ingrese el primer nombre: ");
    gets(nombre1);
    printf("Ingrese el segundo nombre: ");
    gets(nombre2);
    if (strlen(nombre1)>strlen(nombre2))
    {
        printf("El que tiene más caracteres es %s", nombre1);
        printf(" y el que tiene menos caracteres %s", nombre2);
    }
    else
    {
        printf("El que tiene más caracteres es %s", nombre2);
        printf(" y el que tiene menos caracteres %s", nombre1);
    }
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:

```
D:\CursoC\programa82.exe
Ingrese el primer nombre: Ana
Ingrese el segundo nombre: Juan
El que tiene más caracteres es Juan y el que tiene menos caracteres Ana
```

Ejecutamos de nuevo invirtiendo los valores.

```
D:\CursoC\programa82.exe
Ingrese el primer nombre: Luis
Ingrese el segundo nombre: Ana
El que tiene más caracteres es Luis y el que tiene menos caracteres Ana
```

Capítulo 85.- Funciones que facilitan el trabajo con cadenas de caracteres – string.h – 8

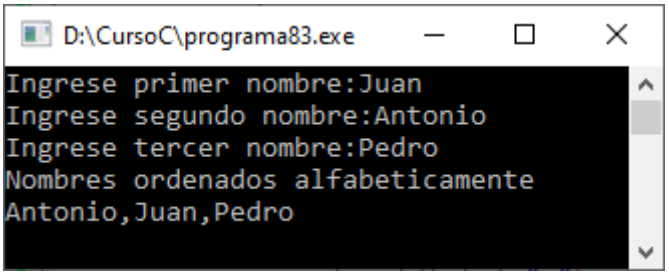
Problema propuesto

Cargar tres nombres por teclado. Generar un cuarto string que almacene los tres nombres ingresados por teclado en orden alfabético separados por una coma.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

int main()
{
    char nombre1[31];
    char nombre2[31];
    char nombre3[31];
    char total[93];
    printf("Ingrese primer nombre:");
    gets(nombre1);
    printf("Ingrese segundo nombre:");
    gets(nombre2);
    printf("Ingrese tercer nombre:");
    gets(nombre3);
    if (strcmp(nombre1,nombre2)<0 && strcmp(nombre1,nombre3)<0)
    {
        strcpy(total,nombre1);
        strcat(total,",");
        if (strcmp(nombre2,nombre3)<0)
        {
            strcat(total,nombre2);
            strcat(total,",");
            strcat(total,nombre3);
        }
        else
        {
            strcat(total,nombre3);
            strcat(total,",");
            strcat(total,nombre2);
        }
    }
    else
    {
        if (strcmp(nombre2,nombre3)<0)
        {
            strcpy(total,nombre2);
            strcat(total,",");
            if (strcmp(nombre1,nombre3)<0)
            {
                strcat(total,nombre1);
                strcat(total,",");
                strcat(total,nombre3);
            }
```

```
46              else
47              {
48                  strcat(total,nombre3);
49                  strcat(total,",");
50                  strcat(total,nombre1);
51              }
52          }
53          else
54          {
55              strcpy(total,nombre3);
56              strcat(total,",");
57              if (strcmp(nombre1,nombre2)<0)
58              {
59                  strcat(total,nombre1);
60                  strcat(total,",");
61                  strcat(total,nombre2);
62              }
63              else
64              {
65                  strcat(total,nombre2);
66                  strcat(total,",");
67                  strcat(total,nombre1);
68              }
69          }
70      }
71      printf("Nombres ordenados alfabeticamente\n");
72      printf("%s",total);
73      getch();
74      return 0;
75  }
76
```

Vamos a ejecutar introduciendo 3 nombres:



Capítulo 86.- Concepto de funciones – Programación estructurada – 1

Hasta ahora hemos trabajado con una metodología de programación lineal. Todas las instrucciones de nuestro programa se ejecutan en forma secuencial de principio a fin.

Esta forma de organizar un programa solo puede ser llevado a cabo si el mismo es muy pequeño (decenas de líneas).

Cuando los problemas a resolver tienden a ser más grandes la metodología de programación lineal se vuelve ineficiente y compleja.

El paradigma de programación que propone el lenguaje C es la programación estructurada.

La programación estructurada busca dividir o descomponer un problema complejo en pequeños problemas. La solución de cada uno de esos pequeños problemas nos trae la solución del problema complejo.

En el lenguaje C el planteo de esas pequeñas soluciones al problema se hace dividiendo el programa en funciones.

Una función es un conjunto de instrucciones que resuelven un problema específico. En el lenguaje C es obligatorio el planteo de una función como mínimo, llamada main:

```
int main()
{
    ....
}
```

El lenguaje C tiene librerías que proveen algunas funcionalidades básicas. Algunas de ellas ya las utilizamos en conceptos anteriores como son las funciones: scanf, printf, strlen, strcpy, etc.

Veamos ahora como crear nuestras propias funciones.

El tema de funciones en un principio puede presentar dificultades para entenderlo y ver sus ventajas ante la metodología de programación lineal que veníamos trabajando todo en la función main.

Los primeros problemas que presentaremos nos pueden parecer que sea más conveniente trabajar todo en la misma función main en ver de dividir el problema en varias funciones.

A medida que avancemos veremos que si un programa empieza a ser más complejo (cientos de líneas, miles de líneas o más) la división en pequeñas funciones nos permitirán tener un programa más ordenado y fácil de entender y por lo tanto en mantener.

Estructura de una función

Una función tiene un nombre, puede tener parámetros y encerrada entre llaves le sigue su algoritmo:

```
[valor que retorna]   [nombre de la función] ([parámetros de la función])
{
    [altoritmo]
}
```

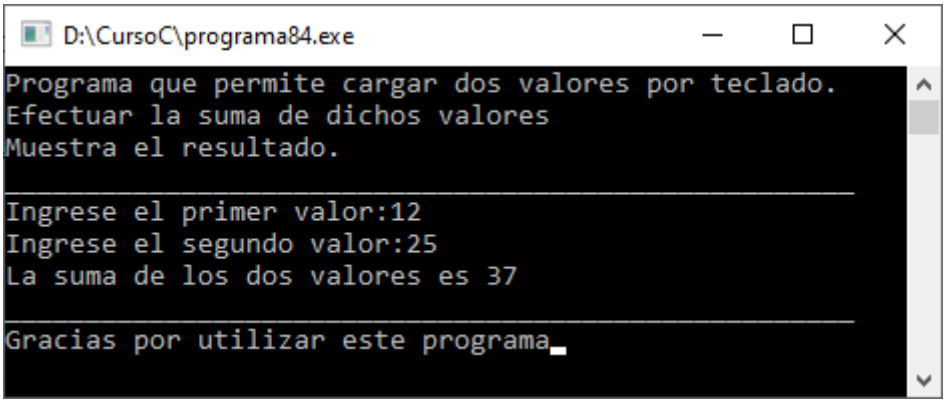
Problema

Confeccionar un programa que muestre una presentación en pantalla del programa. Solicite la carga de dos valores y nos muestre su suma. Mostrar finalmente un mensaje de despedida del programa.

Implementar estas actividades en tres funciones:

```c
#include<stdio.h>
#include<conio.h>

void presentacion()
{
    printf("Programa que permite cargar dos valores por teclado.\n");
    printf("Efectuar la suma de dichos valores\n");
    printf("Muestra el resultado.\n");
    printf("_____\n");
}

void cargarSumar()
{
    int valor1, valor2, suma;
    printf("Ingrese el primer valor:");
    scanf("%i", &valor1);
    printf("Ingrese el segundo valor:");
    scanf("%i", &valor2);
    suma=valor1+valor2;
    printf("La suma de los dos valores es %i\n", suma);
}

void finalizacion()
{
    printf("_____\n");
    printf("Gracias por utilizar este programa");
}

int main()
{
    presentacion();
    cargarSumar();
    finalizacion();

    getch();
    return 0;
}

```

Vamos a ejecutar:



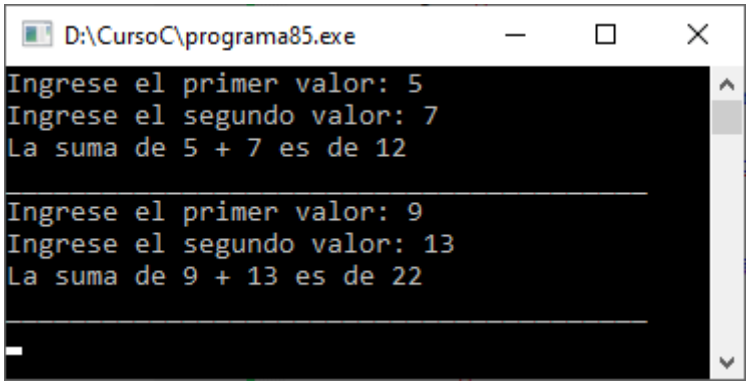
Capítulo 87.- Concepto de funciones – Programación estructurada – 2

Problema

Confeccionar una aplicación que solicite la carga de dos valores enteros y muestre su suma.
Repetir al carga de otros dos valores, sumarlos y mostrar.
Mostrar una línea separadora después de cada vez que cargamos dos valores y mostramos su suma.

```
#include<stdio.h>
#include<conio.h>

void cargar()
{
    int num1, num2, suma;
    printf("Ingrese el primer valor: ");
    scanf("%i", &num1);
    printf("Ingrese el segundo valor: ");
    scanf("%i", &num2);
    suma=num1+num2;
    printf("La suma de %i + %i es de %i\n", num1, num2, suma);
}

void linea()
{
    printf("_____________________________________________________\n");
}

int main()
{
    cargar();
    linea();
    cargar();
    linea();
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



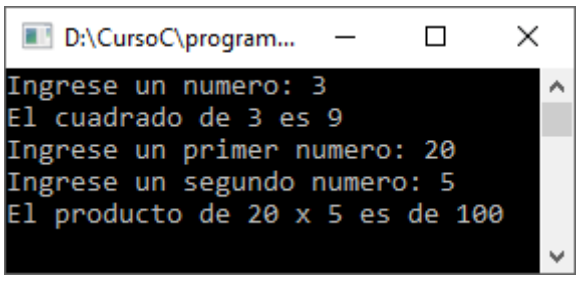
Capítulo 88.- Concepto de funciones – Programación estructurada – 3

Problema propuesto

Desarrollar un programa con dos funciones aparte del main.
La primera que solicite el ingreso de un entero y muestre el cuadrado de dicho valor.
La segunda que solicite la carga de dos valores y muestre el producto de los mismos.
Llamar desde la main a ambas funciones.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   void cuadrado()
5   {
6       int num, cuadrado;
7       printf("Ingrese un numero: ");
8       scanf("%i", &num);
9       cuadrado=num*num;
10      printf("El cuadrado de %i es %i\n", num, cuadrado);
11  }
12
13  void producto()
14  {
15      int num1, num2, producto;
16      printf("Ingrese un primer numero: ");
17      scanf("%i", &num1);
18      printf("Ingrese un segundo numero: ");
19      scanf("%i", &num2);
20      producto=num1*num2;
21      printf("El producto de %i x %i es de %i\n", num1, num2, producto);
22  }
23
24  int main()
25  {
26      cuadrado();
27      producto();
28      getch();
29      return 0;
30  }
31
```

Si ejecutamos este será el resultado:



Capítulo 89.- Concepto de funciones – Programación estructurada – 4

Problema propuesto

Desarrollar una función que solicite la carga de tres valores y muestre el menor. Desde la función main llamar 3 veces a dicha función.

```c
#include<stdio.h>
#include<conio.h>

void mostrarMenor()
{
    int num1, num2, num3;
    printf("Ingrese el primer numero: ");
    scanf("%i", &num1);
    printf("Ingrese el segundo numero: ");
    scanf("%i", &num2);
    printf("Ingrese el tercer numero: ");
    scanf("%i", &num3);
    if(num1<num2 &&  num1<num3)
    {
        printf("El numero menor es %i", num1);
    }
    else
    {
        if(num2<num3)
        {
            printf("El numero menor es %i", num2);
        }
        else
        {
            printf("El numero menor es %i", num3);
        }
    }
    printf("\n");
}

int main()
{
    int f;
    for (f=0; f<3; f++)
    {
        mostrarMenor();
    }
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:

```
D:\CursoC\programa87....
Ingrese el primer numero: 7
Ingrese el segundo numero: 3
Ingrese el tercer numero: 10
El numero menor es 3
Ingrese el primer numero: 5
Ingrese el segundo numero: 8
Ingrese el tercer numero: 12
El numero menor es 5
Ingrese el primer numero: 10
Ingrese el segundo numero: 7
Ingrese el tercer numero: 2
El numero menor es 2
```

Capítulo 90.- Funciones con parámetros de tipo int, float y char – 1

Una función puede recibir uno o más parámetros para que sea más flexible y se adapte a viarias situaciones.
Veremos primera funciones que reciban uno o más parámetros de tipo int, float o char.
La sintaxis cuando una función recibe parámetros es:

```
[valor que retorna]   [nombre de la función] ([parámetros de la función])
{
    [altoritmo]
}
```

Los parámetros se disponen luego del nombre de la función encerrado entre paréntesis y separados por coma, algunos ejemplos:

```
void funcion1(int v1)
{
    ....
}

void funcion2(int v1,int v2)
{
    ....
}

void funcion3(float f1,int v1)
{
    ....
}

void funcion4(char letra)
{
    ....
}
```

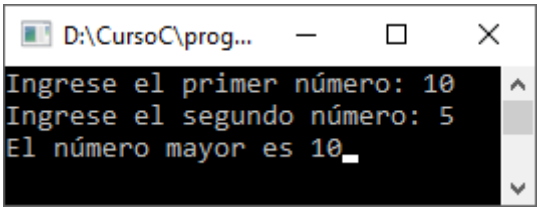
Arriba vemos como declarar funciones con uno o más parámetros de tipos int, float o char.

Problema
Confeccionar una función que reciba dos enteros e imprima el mayor de ellos. Llamar a la función desde la main cargando previamente dos valores por teclado.

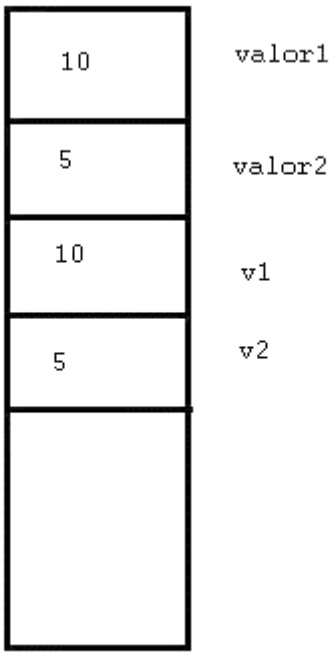
```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<locale.h>
4
5   void numeroMayor(int v1, int v2)
6   {
7       if (v1>v2){
8           printf("El número mayor es %i", v1);
9       }
10      else
11      {
12          printf("El número mayor es %i", v2);
13      }
14  }
```

```
15
16  int main()
17  {
18      setlocale(LC_ALL, "");
19      int valor1, valor2;
20      printf("Ingrese el primer número: ");
21      scanf("%i", &valor1);
22      printf("Ingrese el segundo número: ");
23      scanf("%i", &valor2);
24      numeroMayor(valor1, valor2);
25      getch();
26      return 0;
27  }
28
```

Si ejecutamos este será el resultado:



En este ejemplo la memoria funciona así:



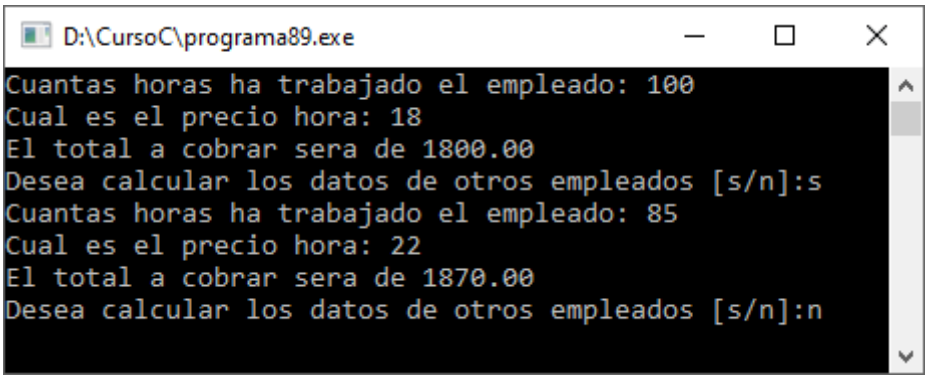
Capítulo 91.- Funciones con parámetros de tipo int, float y char – 2

Problema

Confeccionar un programa que solicite el pago por hora de un empleado y la cantidad de horas trabajadas dentro de una estructura repetitiva en la función main. Elabora una función que reciba como parámetro el valor de la hora y la cantidad de horas trabajadas y nos muestre el total a pagar.

```c
#include<stdio.h>
#include<conio.h>

void totalPagar(float precioHora, int cantidadHoras)
{
    float total=precioHora*cantidadHoras;
    printf("El total a cobrar sera de %.2f\n", total);
}

int main()
{
    int x, horas;
    float precio;
    char opcion;
    do {
        printf("Cuantas horas ha trabajado el empleado: ");
        scanf("%i", &horas);
        printf("Cual es el precio hora: ");
        scanf("%f", &precio);
        totalPagar(precio, horas);
        printf("Desea calcular los datos de otros empleados [s/n]:");
        scanf(" %c", &opcion);
    }while (opcion=='s');
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 92.- Funciones con parámetros de tipo int, float y char – 3

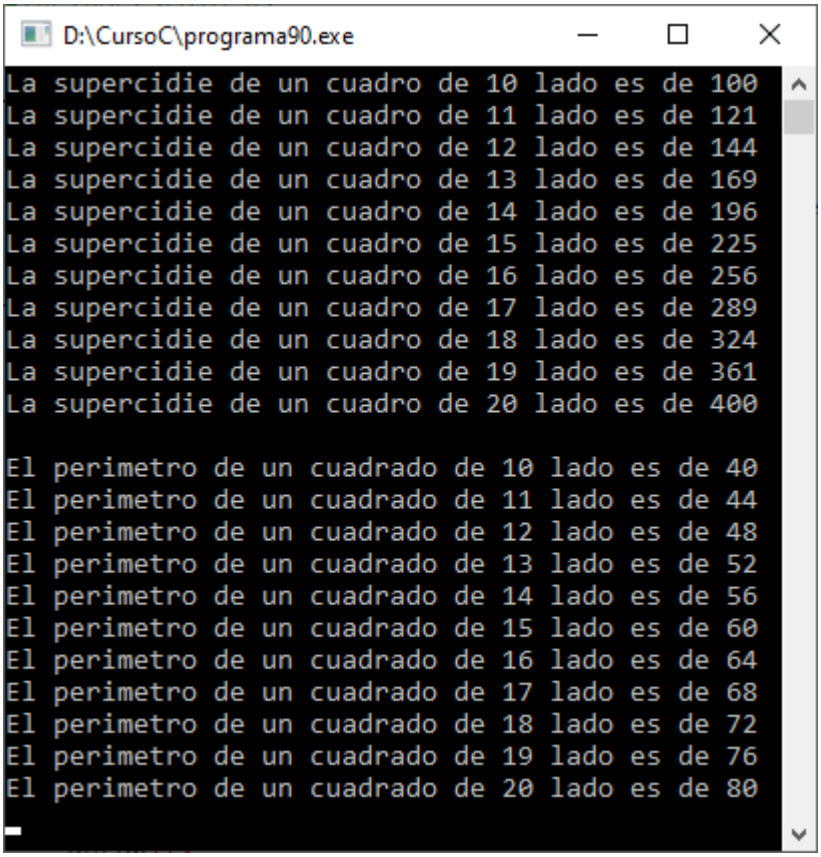
Problema

Desarrollar dos funciones que reciban como parámetro el valor del lado de un cuadrado. La primera debe calcular y mostrar la superficie y la segunda calcular y mostrar el perímetro.

En la main llamar a las funciones pasando los valores enteros comprendidos entre 10 y 20, de uno en uno.

```
#include<stdio.h>
#include<conio.h>

void superficie(int lado)
{
    int area=lado*lado;
    printf("La supercidie de un cuadro de %i lado es de %i\n", lado, area);
}

void perimetro(int lado)
{
    int per=lado*4;
    printf("El perimetro de un cuadrado de %i lado es de %i\n", lado, per);
}

int main()
{
    int x;
    for (x=10; x<=20; x++){
        superficie(x);
    }
    printf("\n");
    for (x=10; x<=20; x++){
        perimetro(x);
    }
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 93.- Funciones con parámetros de tipo int, float y char – 4

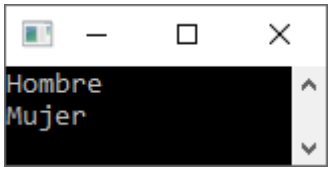
Problema

Desarrollar una función que reciba como parámetro un carácter. Si llega una 'h' mostrar en pantalla el mensaje "hombre", si llega una 'm' mostrar el mensaje "mujer".

Llamar desde la función main pasando una vez una 'h' y otra vez 'm'.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   void sexo(char s)
5   {
6       if(s=='h')
7           printf("Hombre\n");
8       if(s=='m')
9           printf("Mujer\n");
10  }
11
12  int main()
13  {
14      sexo('h');
15      sexo('m');
16      getch();
17      return 0;
18  }
```

Si ejecutamos este será el resultado:



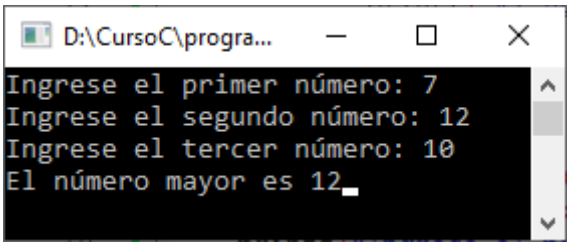
Capítulo 94.- Funciones con parámetros de tipo int, float y char – 5

Problema propuesto

Confeccionar una función que reciba tres enteros y nos muestre el mayor de ellos. La carga de los valores hacerlos por teclado en la función main.

```
#include<stdio.h>
#include<conio.h>
#include<locale.h>
void numeroMayor(int n1, int n2, int n3)
{
    if(n1>n2 && n1>n3)
        printf("El número mayor es %i", n1);
    else
        if (n2>n3)
            printf("El número mayor es %i", n2);
        else
            printf("El número mayor es %i", n3);
}

int main()
{
    setlocale(LC_ALL, "");
    int num1, num2, num3;
    printf("Ingrese el primer número: ");
    scanf("%i", &num1);
    printf("Ingrese el segundo número: ");
    scanf("%i", &num2);
    printf("Ingrese el tercer número: ");
    scanf("%i", &num3);
    numeroMayor(num1, num2, num3);
    getch();
    return 0;
}

```

Si ejecutamos este será el resultado:



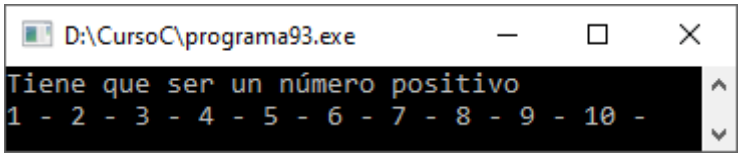
Capítulo 95.- Funciones con parámetros de tipo int, float y char – 6

Problema propuesto

Elaborar una función que reciba un valor entero y nos muestre desde el 1 hasta dicho valor. Si la función recibe un valor negativo mostrar un mensaje de error.

```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<locale.h>
4
5   void mostrarHasta(int n)
6   {
7       if(n<0)
8           printf("Tiene que ser un número positivo\n");
9       else
10      {
11          int x;
12          for(x=1; x<=n; x++)
13          {
14              printf("%i - ", x);
15          }
16          printf("\n");
17      }
18  }
19
20  main()
21  {
22      setlocale(LC_ALL, "");
23      mostrarHasta(-5);
24      mostrarHasta(10);
25      getch();
26      return 0;
27  }
28
```

Si ejecutamos este será el resultado:



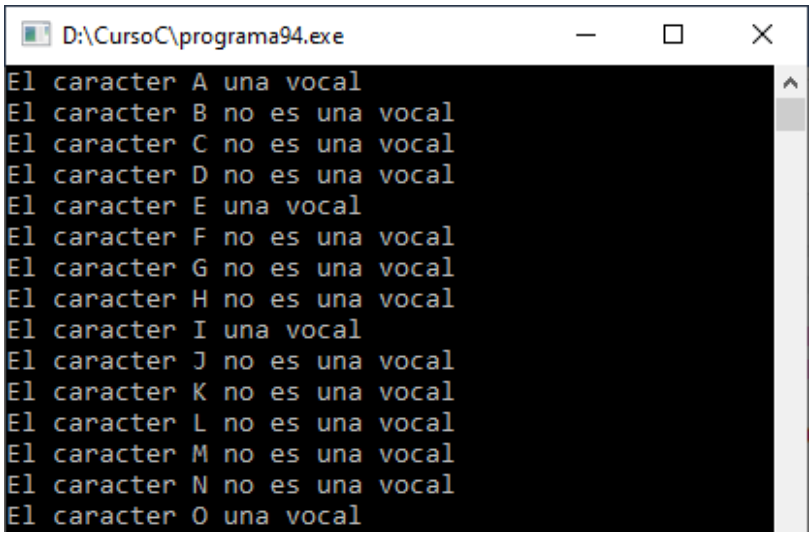
Capítulo 96.- Funciones con parámetros de tipo int, float y char – 7

Problema propuesto

Desarrollar una función que reciba como parámetro un carácter. La función debe mostrar un mensaje si es una vocal o no es una vocal. Debe funcionar tanto con mayúsculas y minúsculas.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   void esVocal(char v)
5   {
6       if(v=='a' || v=='e' || v=='i' || v=='o' ||v=='u' ||
7          v=='A' || v=='E' || v=='I' || v=='O' ||v=='U')
8       {
9           printf("El caracter %c una vocal\n", v);
10      }
11      else
12      {
13          printf("El caracter %c no es una vocal\n", v);
14      }
15  }
16
17  int main()
18  {
19      char c;
20      for (c='A'; c<='z'; c++)
21      {
22          esVocal(c);
23      }
24      getch();
25      return 0;
26  }
27
```

Si ejecutamos este será el resultado:



```
El caracter P no es una vocal
El caracter Q no es una vocal
El caracter R no es una vocal
El caracter S no es una vocal
El caracter T no es una vocal
El caracter U una vocal
El caracter V no es una vocal
El caracter W no es una vocal
El caracter X no es una vocal
El caracter Y no es una vocal
El caracter Z no es una vocal
El caracter [ no es una vocal
El caracter \ no es una vocal
El caracter ] no es una vocal
El caracter ^ no es una vocal
El caracter _ no es una vocal
El caracter ` no es una vocal
El caracter a una vocal
El caracter b no es una vocal
El caracter c no es una vocal
El caracter d no es una vocal
El caracter e una vocal
El caracter f no es una vocal
El caracter g no es una vocal
El caracter h no es una vocal
El caracter i una vocal
El caracter j no es una vocal
El caracter k no es una vocal
El caracter l no es una vocal
El caracter m no es una vocal
El caracter n no es una vocal
El caracter o una vocal
El caracter p no es una vocal
El caracter q no es una vocal
El caracter r no es una vocal
El caracter s no es una vocal
El caracter t no es una vocal
El caracter u una vocal
El caracter v no es una vocal
El caracter w no es una vocal
El caracter x no es una vocal
El caracter y no es una vocal
El caracter z no es una vocal
```

Capítulo 97.- Funciones con retorno de un valor – 1

Estamos viendo en los últimos capítulos la metodología de programación estructurada. Buscamos dividir un problema en subproblemas y plantear funciones que los resuelvan.

Hemos visto que una función la definimos mediante un nombre y puede recibir datos por medio de sus parámetros.

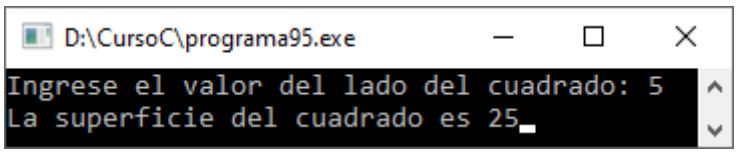
Los parámetros son la forma para que una función reciba datos para ser procesados. Ahora veremos otra característica de las funciones que es la de devolver un dato a quién invocó la función (recordemos que una función la podemos llamar desde la función main o inclusive desde otras funciones de nuestro programa).

Problema

Confeccionar una función que le enviemos como parámetro el valor de lado de un cuadrado y nos retorne su superficie.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   int retornarSuperficie(int lado)
5   {
6       int superficie=lado*lado;
7       return superficie;
8   }
9
10  int main()
11  {
12      int valor, sup;
13      printf("Ingrese el valor del lado del cuadrado: ");
14      scanf("%i", &valor);
15      sup=retornarSuperficie(valor);
16      printf("La superficie del cuadrado es %i", sup);
17      getch();
18      return 0;
19  }
20
```

Si ejecutamos este será el resultado:



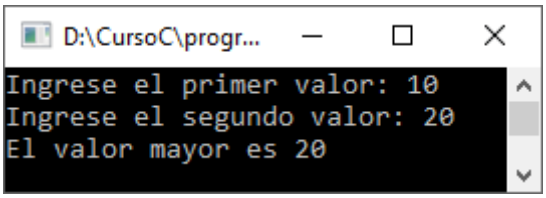
Capítulo 98.- Funciones con retorno de un valor – 2

Problema

Confeccionar una función que defina dos parámetros enteros y nos retorne el mayor.

```c
#include<stdio.h>
#include<conio.h>

int numMayor(int num1, int num2)
{
    if(num1>=num2)
    {
        return num1;
    }
    else
    {
        return num2;
    }
}

int main()
{
    int valor1, valor2, mayor;
    printf("Ingrese el primer valor: ");
    scanf("%i", &valor1);
    printf("Ingrese el segundo valor: ");
    scanf("%i", &valor2);
    mayor=numMayor(valor1, valor2);
    printf("El valor mayor es %i", mayor);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



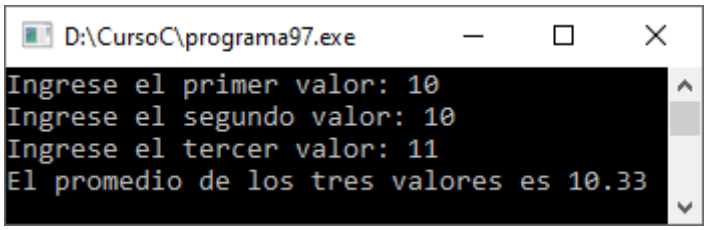
Capítulo 99.- Funciones con retorno de un valor – 3

Problema propuesto

Elaborar una función que reciba tres enteros y nos retorne el valor promedio de los mismos.

```c
#include<stdio.h>
#include<conio.h>

float promedio(int num1, int num2, int num3)
{
    float prome=(float)(num1+num2+num3)/3;
    return prome;
}

int main()
{
    float resultado;
    int nu1, nu2, nu3;
    printf("Ingrese el primer valor: ");
    scanf("%i", &nu1);
    printf("Ingrese el segundo valor: ");
    scanf("%i", &nu2);
    printf("Ingrese el tercer valor: ");
    scanf("%i", &nu3);
    resultado=promedio(nu1, nu2, nu3);
    printf("El promedio de los tres valores es %0.2f", resultado);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



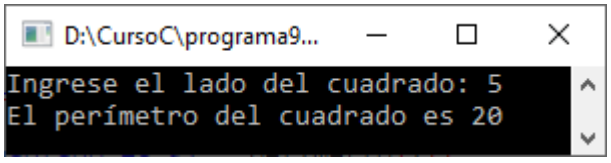
Capítulo 100.- Funciones con retorno de un valor – 4

Problema propuesto

Elaborar una función que nos retorne el perímetro de un cuadrado pasando como parámetro el valor de un lado.

```c
#include<stdio.h>
#include<conio.h>
#include<locale.h>

int perimetro(lado)
{
   return lado*4;
}

int main()
{
    setlocale(LC_ALL, "");
    int l;
    printf("Ingrese el lado del cuadrado: ");
    scanf("%i", &l);
    printf("El perímetro del cuadrado es %i", perimetro(l));
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Otra posible solución:

```c
#include<stdio.h>
#include<conio.h>
#include<locale.h>

int perimetro(lado)
{
   return lado*4;
}

int main()
{
    setlocale(LC_ALL, "");
    int l,per;
    printf("Ingrese el lado del cuadrado: ");
    scanf("%i", &l);
    per=perimetro(l);
    printf("El perímetro del cuadrado es %i", per);
    getch();
    return 0;
}
```

Capítulo 101.- Funciones con retorno de un valor – 5

Problema propuesto

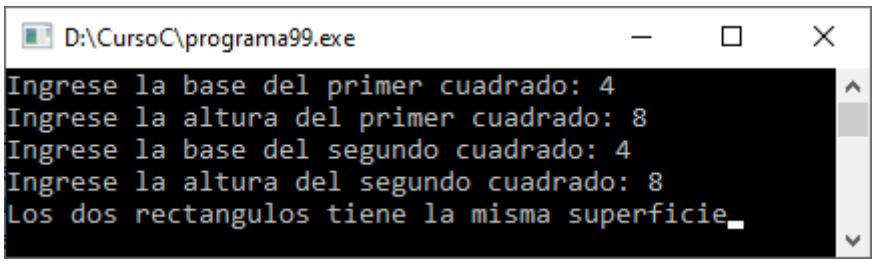
Confeccionar una función que calcule la superficie de un rectángulo y la retorne, la función recibe como parámetro los valore de dos de su lados:

int retornarSuperficie(int lado1, int lado2);

En la función main del programa cargar los lados de dos rectángulos y luego mostrar cual de los dos tiene una superficie mayor.

```
#include<stdio.h>
#include<conio.h>

int retornarSuperficie(int lado1, int lado2)
{
    return lado1*lado2;
}

int main()
{
    int lado1, lado2, lado3, lado4;
    printf("Ingrese la base del primer cuadrado: ");
    scanf("%i", &lado1);
    printf("Ingrese la altura del primer cuadrado: ");
    scanf("%i", &lado2);
    printf("Ingrese la base del segundo cuadrado: ");
    scanf("%i", &lado3);
    printf("Ingrese la altura del segundo cuadrado: ");
    scanf("%i", &lado4);
    if(retornarSuperficie(lado1, lado2)==retornarSuperficie(lado3, lado4))
    {
        printf("Los dos rectangulos tiene la misma superficie");
    }
    else
    {
         if(retornarSuperficie(lado1, lado2)>retornarSuperficie(lado3, lado4))
        {
             printf("El primer cuadrado tiene un superficie superior");
        }
        else
        {
            printf("El segundo cuadrado tiene un superficie superior");
        }
    }
    getch();
    return 0;
}

```

Si ejecutamos este será el resultado:



Capítulo 102.- Funciones con parámetros de tipo vector – 1

Vimos en capítulos anteriores que un vector en el lenguaje C es una estructura de datos que permite almacenar un CONJUNTO de datos del MISMO tipo. Con un único nombre se define el vector y por medio de un subíndice hacemos referencia a cada elemento del mismo (componente).

Ahora veremos que una función puede recibir como parámetro tipos de dato vector.

Importante

Lo más importante de un vector que cuando se pasa como parámetro a una función no se hace una copia como sucede con los tipos de datos char, int y float sino que se para la dirección de memoria donde se almacena el vector original.

Esta forma de pasaje de datos tipo vector hace que si modificamos el parámetro dentro de la función lo que sucede es que se modifica la variable definida en la función main o donde se ha definido.

Problema

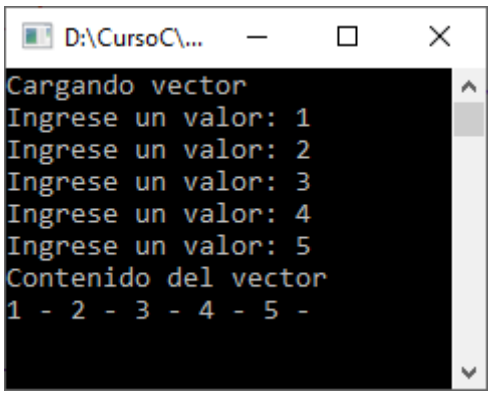
Confeccionar un programa que defina dos funciones, una que permita cargar un vector de 5 elementos enteros y otra que nos muestre un vector de 5 elementos enteros.

En la función main definir una variable de tipo vector y seguidamente llamar a las dos funciones.

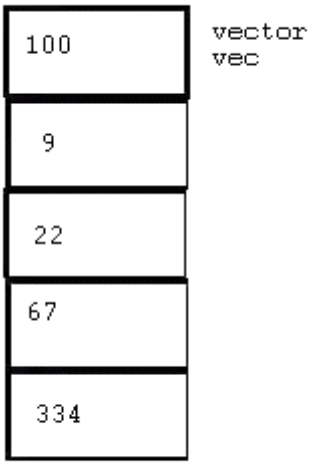
```
1  #include<stdio.h>
2  #include<conio.h>
3
4  void cargarVector(int vec[5])
5      {
6         printf("Cargando vector\n");
7         for (int x=0; x<5; x++)
8         {
9             printf("Ingrese un valor: ");
10            scanf("%i", &vec[x]);
11        }
12    }
13 void leerVector(int vec[5])
14 {
15    printf("Contenido del vector\n");
16    for (int x=0; x<5; x++)
17    {
18        printf("%i - ",vec[x]);
19    }
20    printf("\n");
21 }
22
```

```
23  int main()
24  {
25      int vector[5];
26      cargarVector(vector);
27      leerVector(vector);
28      getch();
29      return 0;
30  }
31
```

Si ejecutamos este será el resultado:



Debe quedar claro que cuando pasamos la variable vector como parámetro vec no recibe una copia del vector sino que guarda la referencia de memoria de la variable vector (cuando cargamos enteros en el parámetro vec estamos cargando la variable vector definida en el main):



Capítulo 103.- Funciones con parámetros de tipo vector – 2

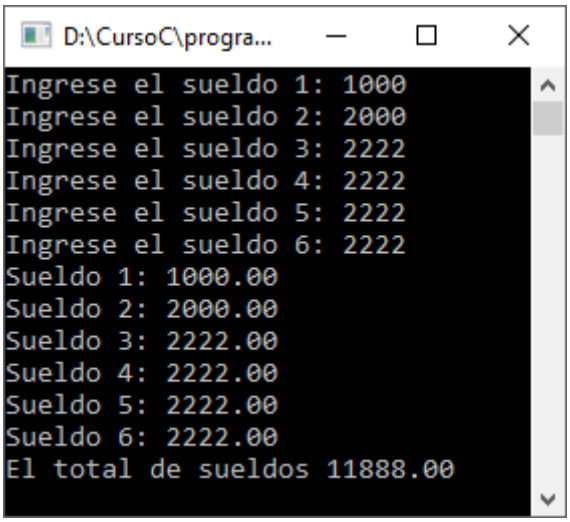
Problema

Guardar los datos de 6 sueldos de empleados en un vector de tipo float. Confeccionar las siguientes funciones:

1. Carga de sueldos.
2. Impresión de sueldos.
3. Gasto total de la empresa en sueldos.

```
#include<stdio.h>
#include<conio.h>

void ingresarSueldos(float sueldos[6])
{
    for (int x=0; x<6; x++)
    {
        printf("Ingrese el sueldo %i: ", x+1);
        scanf("%f", &sueldos[x]);
    }
}

void mostrarSueldos(float sueldos[6])
{
    for (int x=0; x<6; x++)
    {
        printf("Sueldo %i: %.2f\n", (x+1), sueldos[x]);
    }
}

float totalSueldos(float sueldos[6])
{
    float total;
    for (int x=0; x<6; x++)
    {
        total=total+sueldos[x];
    }
    return total;
}

int main()
{
    float sueldo[6];
    ingresarSueldos(sueldo);
    mostrarSueldos(sueldo);
    printf("El total de sueldos %.2f", totalSueldos(sueldo));
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:

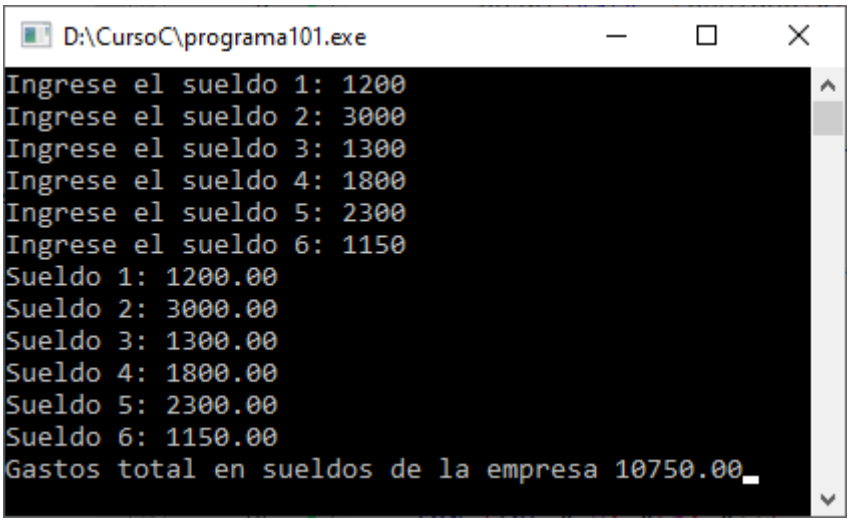


Otra posible solución:

```c
#include<stdio.h>
#include<conio.h>

void ingresarSueldos(float sueldos[6])
{
    for (int x=0; x<6; x++)
    {
        printf("Ingrese el sueldo %i: ", x+1);
        scanf("%f", &sueldos[x]);
    }
}

void mostrarSueldos(float sueldos[6])
{
    for (int x=0; x<6; x++)
    {
        printf("Sueldo %i: %.2f\n", (x+1), sueldos[x]);
    }
}

void totalSueldos(float sueldos[6])
{
    float total;
    for (int x=0; x<6; x++)
    {
        total=total+sueldos[x];
    }
    printf("Gastos total en sueldos de la empresa %0.2f", total);
}

```

```
31  int main()
32  {
33      float sueldo[6];
34      ingresarSueldos(sueldo);
35      mostrarSueldos(sueldo);
36      totalSueldos(sueldo);
37      getch();
38      return 0;
39  }
40
```

Cuando ejecutemos el resultado será el mismo.



Capítulo 104.- Funciones con parámetros de tipo vector – 3

Problema

Definir tres vectores de tipo entero de 5 elementos cada uno. Realizar la carga de los dos primeros por teclado. Definir una única función que realice la carga de un vector y llamar a dicha función dos veces pasando el primer y segundo vector definido.

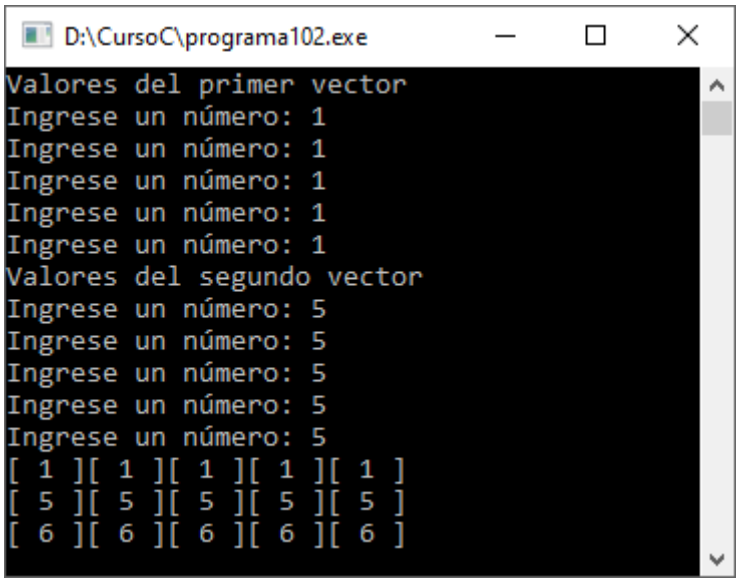
Plantear otra función que reciba tres vectores y proceda a sumar elemento a elemento los dos primeros vectores y se cargue en el tercer vector.

Imprimir los tres vectores.

```c
#include<stdio.h>
#include<conio.h>
#include<locale.h>

void ingresarPrimerVector(int v[5])
{
    printf("Valores del primer vector\n");
    for(int x=0; x<5; x++)
    {
        printf("Ingrese un número: ");
        scanf("%i", &v[x]);
    }
}

void ingresarSegundoVector(int v[5])
{
    printf("Valores del segundo vector\n");
    for(int x=0; x<5; x++)
    {
        printf("Ingrese un número: ");
        scanf("%i", &v[x]);
    }
}

void sumarVector(int v1[5], int v2[5], int v3[5])
{
    for(int x=0; x<5; x++)
    {
        v3[x]=v1[x]+v2[x];
    }
}

void mostrarSumaVector(int v[5])
{
    for (int x=0; x<5; x++)
    {
        printf("[ %i ]", v[x]);
    }
    printf("\n");
}


```

```
43  int main()
44  {
45      setlocale(LC_ALL, "");
46      int vec1[5], vec2[5], sumavec[5];
47      for (int x=0; x<5; x++)
48          sumavec[x]=0;
49      ingresarPrimerVector(vec1);
50      ingresarSegundoVector(vec2);
51      sumarVector(vec1,vec2, sumavec);
52      mostrarSumaVector(vec1);
53      mostrarSumaVector(vec2);
54      mostrarSumaVector(sumavec);
55      getch();
56      return 0;
57  }
58
```

Si ejecutamos este será el resultado:



Capítulo 105.- Funciones con parámetros de tipo vector – 4

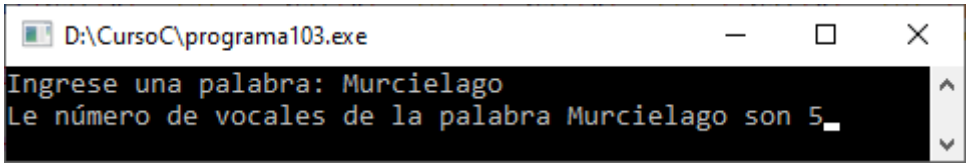
Problema

Confeccionar dos funciones:

1- Permita ingresar por teclado una palabra en un vector de caracteres que llega como parámetro.
2- Retornar la cantidad de vocales que tiene la palabra.

```c
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<locale.h>

void IngresarPalabra(char pal[21])
{
    printf("Ingrese una palabra: ");
    gets(pal);
}

int retornarVocales(char pal[21])
{
    int contador=0;
    int x=0;
    while(pal[x]!='\0')
    {
        if(pal[x]=='a' || pal[x]=='e' || pal[x]=='i' ||pal[x]=='o' ||pal[x]=='u'
           || pal[x]=='A' || pal[x]=='E' || pal[x]=='I' ||pal[x]=='O' ||pal[x]=='U' )
        {
            contador++;
        }
        x++;
    }
    return contador;
}

int main()
{
    setlocale(LC_ALL, "");
    char palabra[21];
    IngresarPalabra(palabra);
    printf("Le número de vocales de la palabra %s son %i",
           palabra, retornarVocales(palabra));
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 106.- Funciones con parámetros de tipo vector – 5

Problema propuesto

Definir un vector de 5 componentes de tipo float en la función main que representa las alturas de 5 personas.

Desarrollar las siguientes funciones:

1- Cargar el vector.
2- Retornar el promedio del vector.

float calcularPromedio(float alturas[5])

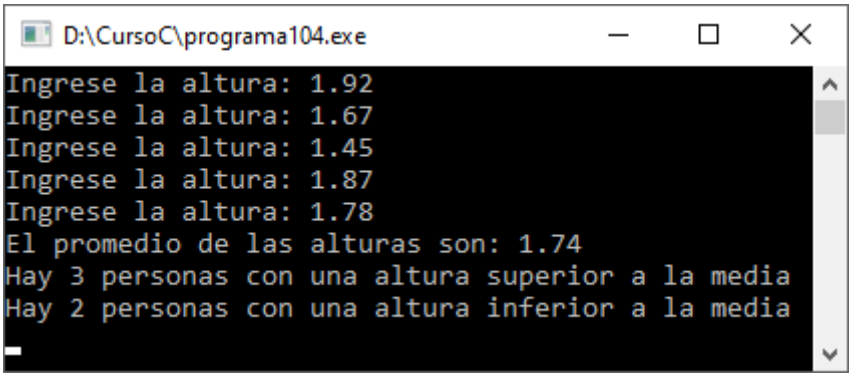
3- Contar y luego imprimir cuántas personas son más altas que el promedio y cuántas más bajas.

void altasBajas(float alturas[5], float pro)

```c
#include<stdio.h>
#include<conio.h>

void cargarAlturas(float altura[5])
{
    for (int x=0; x<5; x++)
    {
        printf("Ingrese la altura: ");
        scanf("%f", &altura[x]);
    }
}

float promedio(float altura[5])
{
    float suma=0;
    for (int x=0; x<5; x++)
    {
        suma=suma+altura[x];
    }
    return suma/5;
}

void altasBajas(float alturas[5], float pro)
{
    int altas=0, bajas=0;
    for(int x=0; x<5; x++)
    {
        if (alturas[x]>pro)
        {
            altas++;
        }
        else
        {
            bajas++;
        }
    }
    printf("Hay %i personas con una altura superior a la media\n", altas);
    printf("Hay %i personas con una altura inferior a la media\n", bajas);
}
```

```
40
41  int main()
42  {
43      float alturas[5];
44      cargarAlturas(alturas);
45      printf("El promedio de las alturas son: %0.2f\n", promedio(alturas));
46      altasBajas(alturas, promedio(alturas));
47      getch();
48      return 0;
49  }
```

Si ejecutamos este será el resultado:



Capítulo 107.- Funciones con parámetros de tipo vector – 6

Problema propuesto

Una empresa tiene dos turnos (mañana y tarde) en los que 8 empleados (4 por la mañana y 4 por la tarde).

Confeccionar un programa que permita almacenar los sueldos de los empleados agrupados por turno.

Imprimir los gastos en sueldos de cada turno.

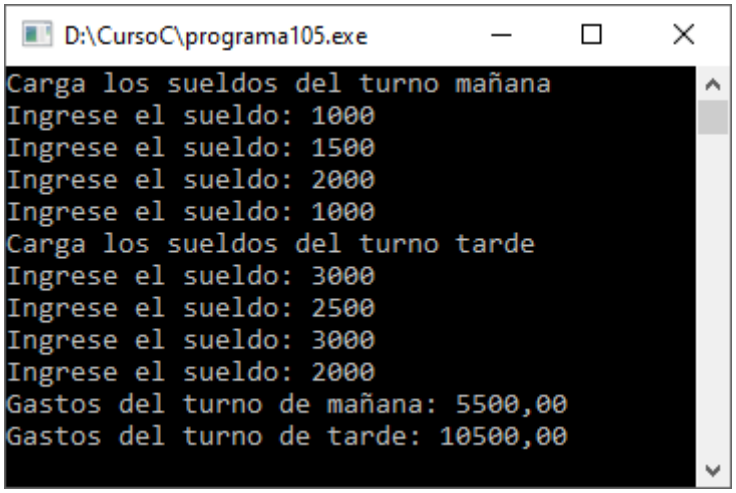
Se deben implementar las siguientes funciones:

```
void cargar(float sueldos[4])
..
float gastosTurno(float sueldos[4])
..
int main()
{
    float sueldosMa[4];
    float sueldosTar[4];
    printf("Carga de sueldos del turno manana\n");
    cargar(sueldosMa);
    printf("Carga de sueldos del turno tarde\n");
    cargar(sueldosTar);
    printf("Gastos del turno de la manana: %0.2f\n", gastosTurno(sueldosMa));
    printf("Gastos del turno de la tarde: %0.2f", gastosTurno(sueldosTar));
    getch();
    return 0;
}
```

```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<locale.h>
4
5   void cargar(float sueldos[4])
6   {
7       for (int x=0; x<4; x++)
8       {
9           printf("Ingrese el sueldo: ");
10          scanf("%f", &sueldos[x]);
11      }
12  }
13
14  float gastosTurno(float sueldos[4])
15  {
16      float total=0;
17      for (int x=0; x<4; x++)
18      {
19          total = total+sueldos[x];
20      }
21      return total;
22  }
23
24  int main()
25  {
26      setlocale(LC_ALL, "");
27      float sueldosMa[4];
28      float sueldosTa[4];
29      printf("Carga los sueldos del turno mañana\n");
30      cargar(sueldosMa);
```

```
31      printf("Carga los sueldos del turno tarde\n");
32      cargar(sueldosTa);
33      printf("Gastos del turno de mañana: %0.2f\n", gastosTurno(sueldosMa));
34      printf("Gastos del turno de tarde: %0.2f\n", gastosTurno(sueldosTa));
35      getch();
36      return 0;
37  }
38
```

Si ejecutamos este será el resultado:



Capítulo 108.- Funciones con parámetros de tipo vector – 7

Problema propuesto

Desarrollar un programa que permita administrar un vector de 8 elementos tipo entero.

Se deben codificar las siguientes funciones:

1. – Cargar un ventor.

```
void cargar(int vector[8])
```

2. Retornar el valor acumulado de todos los elementos del vector.

```
int sumar(int vector[8])
```

3. Retornar el valor acumulado de los elementos del vector que sean mayores a 36.

```
int sumaMayores36(int vector[8])
```

4. Retornar la cantidad de componentes con valores mayores a 50.

```
int cantidadMayores50(int vector[8])
```

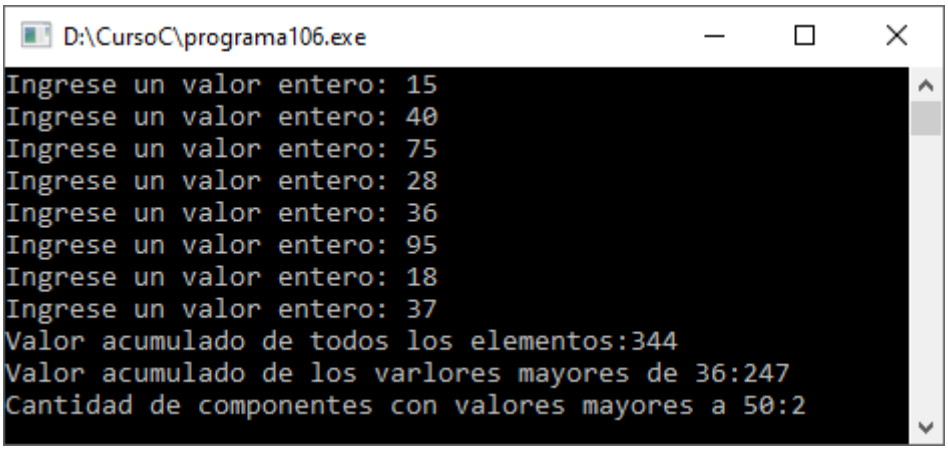
La función main:

```
int main()
{
    int vector[8];
    cargar(vector);
    printf("valor acumulado de todos los elementos:%i\n", sumar(vector));
    printf("valor acumulado de los elementos del vector que sean mayores a 36:
%i\n", sumaMayores36(vector));
    printf("cantidad de componentes con valores mayores a 50: %i",
cantidadMayores50(vector));
    getch();
    return 0;
}
```

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   void cargar(int vector[8])
5   {
6       for(int x=0; x<8; x++)
7       {
8           printf("Ingrese un valor entero: ");
9           scanf("%i", &vector[x]);
10      }
11  }
12
13  int sumar(int vector[8])
14  {
15      int total=0;
16      for (int x=0; x<8; x++)
17      {
18          total=total+vector[x];
19      }
20      return total;
21  }
```

```
21  }
22
23  int sumarMayor36(int vector[8])
24  {
25      int total=0;
26      for (int x=0; x<8; x++)
27      {
28
29          if (vector[x]>36)
30          {
31              total=total+vector[x];
32          }
33      }
34      return total;
35  }
36
37  int cantidadMayor50(int vector[8])
38  {
39      int cantidad=0;
40      for (int x=0; x<8; x++)
41      {
42          if (vector[x]>50)
43          {
44              cantidad++;
45          }
46      }
47      return cantidad;
48  }
49
50  int main()
51  {
52      int vector[8];
53      cargar(vector);
54      printf("Valor acumulado de todos los elementos:%i\n",
55             sumar(vector));
56      printf("Valor acumulado de los varlores mayores de 36:%i\n",
57             sumarMayor36(vector));
58      printf("Cantidad de componentes con valores mayores a 50:%i\n",
59             cantidadMayor50(vector));
60      getch();
61      return 0;
62  }
```

Si ejecutamos este será el resultado:



Capítulo 109.- Funciones con parámetros de tipo vector – 8

Problema propuesto

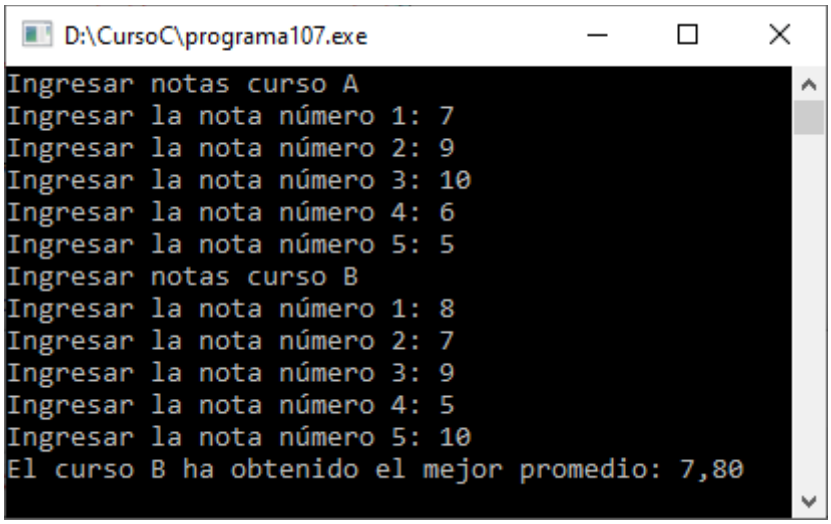
Se tienen las notas del primer parcial de los alumnos de dos curos, el curso A y el curso B, cada curso cuenta con 5 alumnos.

1- Realizar la carga de notas.
2- Mostrar el nombre del curso que obtuvo el mayor promedio general.

```c
#include<stdio.h>
#include<conio.h>
#include<locale.h>

void cargarNotas(int vector[5])
{
    for (int x=0; x<5; x++)
    {
        printf("Ingresar la nota número %i: ", (x+1));
        scanf("%i", &vector[x]);
    }
}

float promedio(int vector[5])
{
    float prom;
    int suma=0;
    for(int x=0; x<5; x++)
    {
        suma=suma+vector[x];
    }
    prom=((float)suma/5);
    return prom;
}

void mayorPromedio(int vectorA[5], int vectorB[5])
{
    if(promedio(vectorA)>promedio(vectorB))
    {
        printf("El curso A ha obtenido el mejor promedio: %0.2f",
               promedio(vectorA));
    }
    else
    {
        if(promedio(vectorB)>promedio(vectorA))
        {
            printf("El curso B ha obtenido el mejor promedio: %0.2f",
                   promedio(vectorB));
        }
        else
        {
            printf("Los dos cursos tienen el mismo promedio: %0.2f",
                   promedio(vectorA));
        }

    }
}
```

```
48
49  int main()
50  {
51      setlocale(LC_ALL, "");
52      int cursoA[5], cursoB[5];
53      printf("Ingresar notas curso A\n");
54      cargarNotas(cursoA);
55      printf("Ingresar notas curso B\n");
56      cargarNotas(cursoB);
57      mayorPromedio(cursoA, cursoB);
58      getch();
59      return 0;
60  }
61
```

Si ejecutamos este será el resultado:



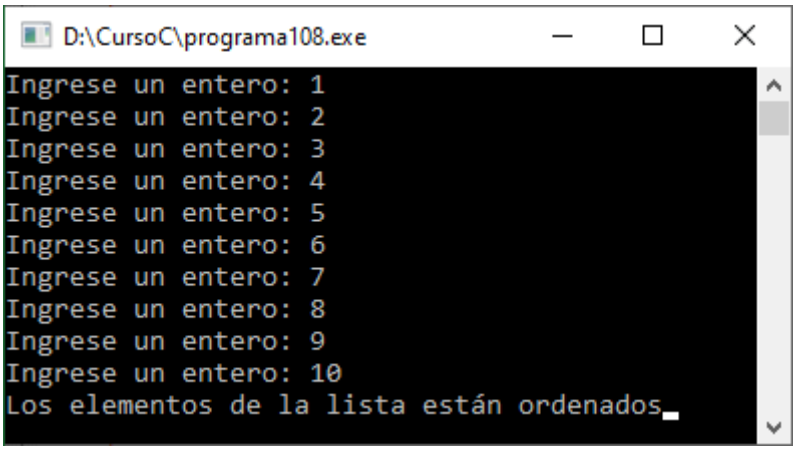
Capítulo 110.- Funciones con parámetros de tipo vector – 9

Problema propuesto

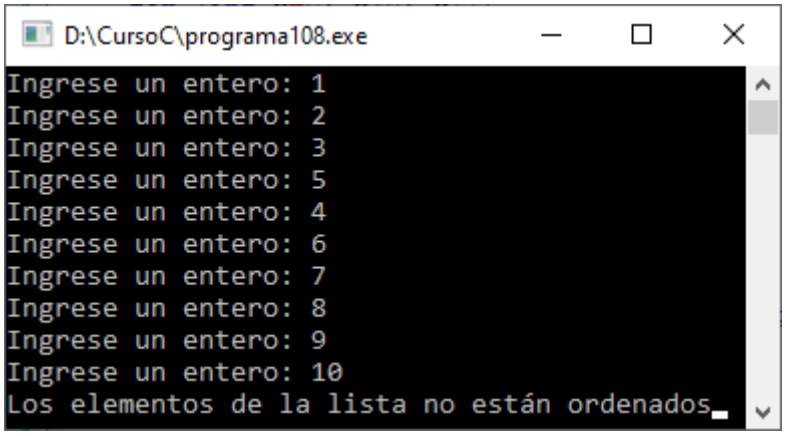
Cargar un vector de 10 elementos y verificar posteriormente si el mismo está ordenado de menor a mayor. Hacer las dos actividades en funciones distintas.

```
#include<stdio.h>
#include<conio.h>
#include<locale.h>

void cargar(int vec[10])
{
    for (int x=0; x<10; x++)
    {
        printf("Ingrese un entero: ");
        scanf("%i", &vec[x]);
    }
}

void ordenado(int vec[10])
{
    int orden=1;
    for (int x=0; x<9; x++)
    {
        if(vec[x]>vec[x+1])
        {
            orden=0;
            break;
        }
    }
    if(orden==1)
        printf("Los elementos de la lista están ordenados");
    else
        printf("Los elementos de la lista no están ordenados");

}

int main()
{
    setlocale(LC_ALL, "");
    int vector[10];
    cargar(vector);
    ordenado(vector);
    getch();
    return 0;
}
```

Vamos a ejecutar y agregamos elementos al vector ordenados:



Ejecutamos de nuevo y agregamos elementos al vector desordenados.

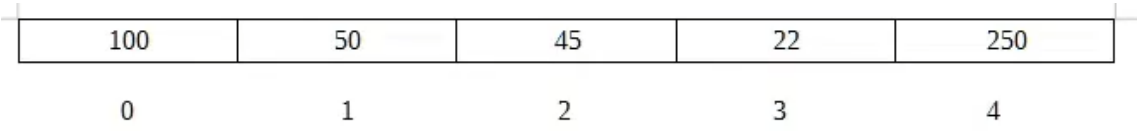


Capítulo 111.- Vectores mayor y menor elemento – 1

Un algoritmo muy común es la búsqueda del mayor y menor elemento de un vector, lo mismo que su posición.

Tiene sentido sobre todo trabajando con vectores que almacenan valores enteros o flotantes.

Si tenemos un vector que almacena los siguientes datos:



Luego el mayor del vector es el número 255 que se encuentra en la posición 4.

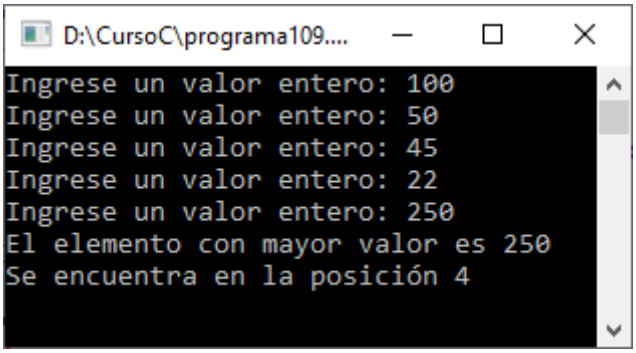
Problema

Confeccionar un programa que defina en el main un vector de 5 elementos de tipo entero. Cargar e imprimir el mayor elemento y su posición.

```
#include<stdio.h>
#include<conio.h>
#include<locale.h>

void cargar(int vec[5])
{
    for (int x=0; x<5; x++)
    {
        printf("Ingrese un valor entero: ");
        scanf("%i", &vec[x]);
    }
}

void mayorPosicion(int vec[5])
{
    int mayor, pos;
    mayor=vec[0];
    pos=0;
    for (int x=1; x<5; x++)
    {
        if (vec[x]>mayor){
            mayor=vec[x];
            pos=x;
        }
    }
    printf("El elemento con mayor valor es %i\n", mayor);
    printf("Se encuentra en la posición %i",pos);
}

int main()
{
    setlocale(LC_ALL, "");
    int vector[5];
    cargar(vector);
    mayorPosicion(vector);
```

```
36      getch();
37      return 0;
38  }
```

Si ejecutamos este será el resultado:



Capítulo 112.- Vectores mayor y menor elemento – 2

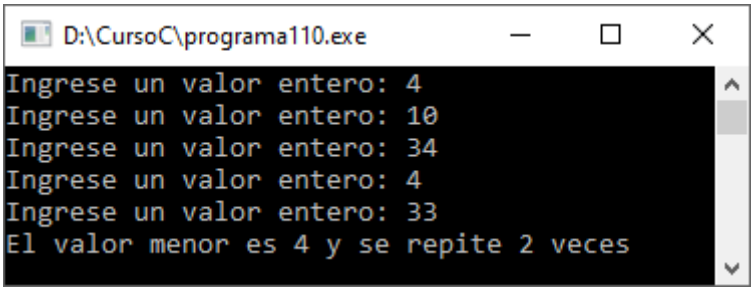
Problema propuesto

Cargar un vector de 5 elementos. Imprimir el menor y un mensaje si se repite dentro del vector.

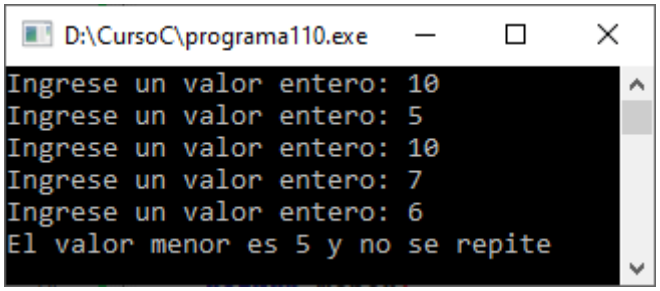
```
#include<stdio.h>
#include<conio.h>
#include<locale.h>

void cargar(int vec[5])
{
    for (int x=0; x<5; x++)
    {
        printf("Ingrese un valor entero: ");
        scanf("%i", &vec[x]);
    }
}

int menor(int vec[5])
{
    int menor=vec[0];
    for (int x=1; x<5; x++)
    {
        if(vec[x]<menor)
        {
            menor=vec[x];
        }
    }
    return menor;
}

void comparar(int vec[5], int menor)
{
    int cant=0;
    for(int x=0; x<5; x++)
        if(vec[x]==menor)
    {
        cant++;
    }
    if(cant>1)
    {
        printf("El valor menor es %i", menor);
        printf( " y se repite %i veces", cant);
    }
    else
    {
        printf("El valor menor es %i", menor);
        printf( " y no se repite");
    }
}
```

```
46
47  int main()
48  {
49      setlocale(LC_ALL, "");
50      int vector[5];
51      cargar(vector);
52      int me=menor(vector);
53      comparar(vector, me);
54      getch();
55      return 0;
56  }
```

Si ejecutamos este será el resultado repitiendo el valor menor:



Ejecutamos de nuevo sin repetir el valor menor:



Capítulo 113.- Vectores ordenamiento – 1

El ordenamiento de un vector en el lenguaje C lo logra intercambiando los elementos de manera que:

vec[0] <= vec[1] <= vec[2] etc.

El contenido del elemento vec[0] sea menor o igual al contenido del elemento vec[1] y así sucesivamente.

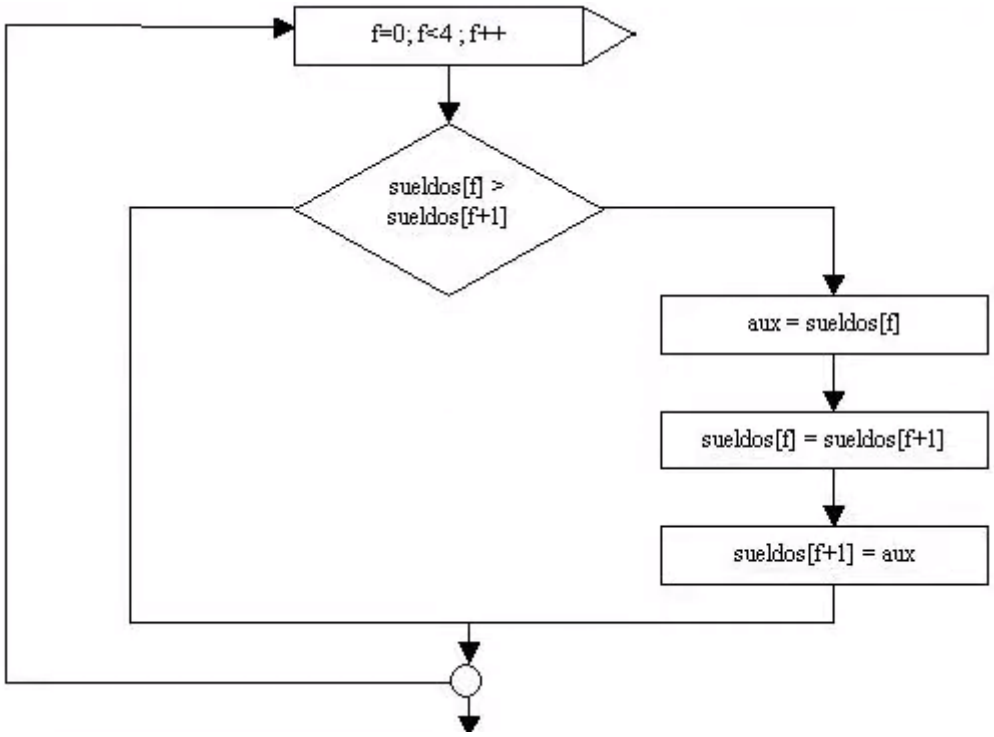
Si se cumple lo dicho anteriormente decimos que el vector está ordenado de menor a mayor.

Igualmente podemos ordenar un vector de mayor a menor.

Se puede ordenar tantos vectores como componentes de tipo int, float y char.

Problema

Se debe crear un vector que contenga 5 sueldos. Ordenar el vector de menor a mayor.



Esta primera aproximación tiene por objetivo analizar los intercambios de elementos dentro del vector.

El algoritmo consiste en comparar si el primer elemento es mayor que el segundo, en caso que la condición sea verdadera, intercambiamos los contenidos de los elementos.

Vamos a suponer que ingresamos los siguientes valores por teclado:

1200
750
820
550
490

En este ejemplo: ¿es 1200 mayor a 750? La respuesta es verdadera, por lo tanto intercambiamos el contenido del elemento 0 con el del elemento 1.

Luego comparamos el contenido del elemento 1 con el contenido del elemento 2: ¿Es 1200 Mayor a 820?

La respuesta es verdadera entonces intercambiamos.

Si hay 5 elementos hay que hacer 4 comparaciones, por eso el for se repite 4 veces.

Generalizamos: si el vector tiene N elementos hay que hacer N-1 comparaciones.

Cuando	f = 0	f = 1	f = 2	f = 3
	750	750	750	750
	1200	820	820	820
	820	1200	550	550
	550	550	1200	490
	490	490	490	1200

Podemos ver cómo el valor más grande del vector desciende a al último elemento. Empleamos una variable auxiliar (aux) para el proceso de intercambio.

```
aux=sueldos[f];
sueldos[f]=sueldos[f+1];
sueldos[f+1]=aux;
```

al salir del for en el ejemplo el contenido del vector es el siguiente:

```
750
820
550
490
1200
```

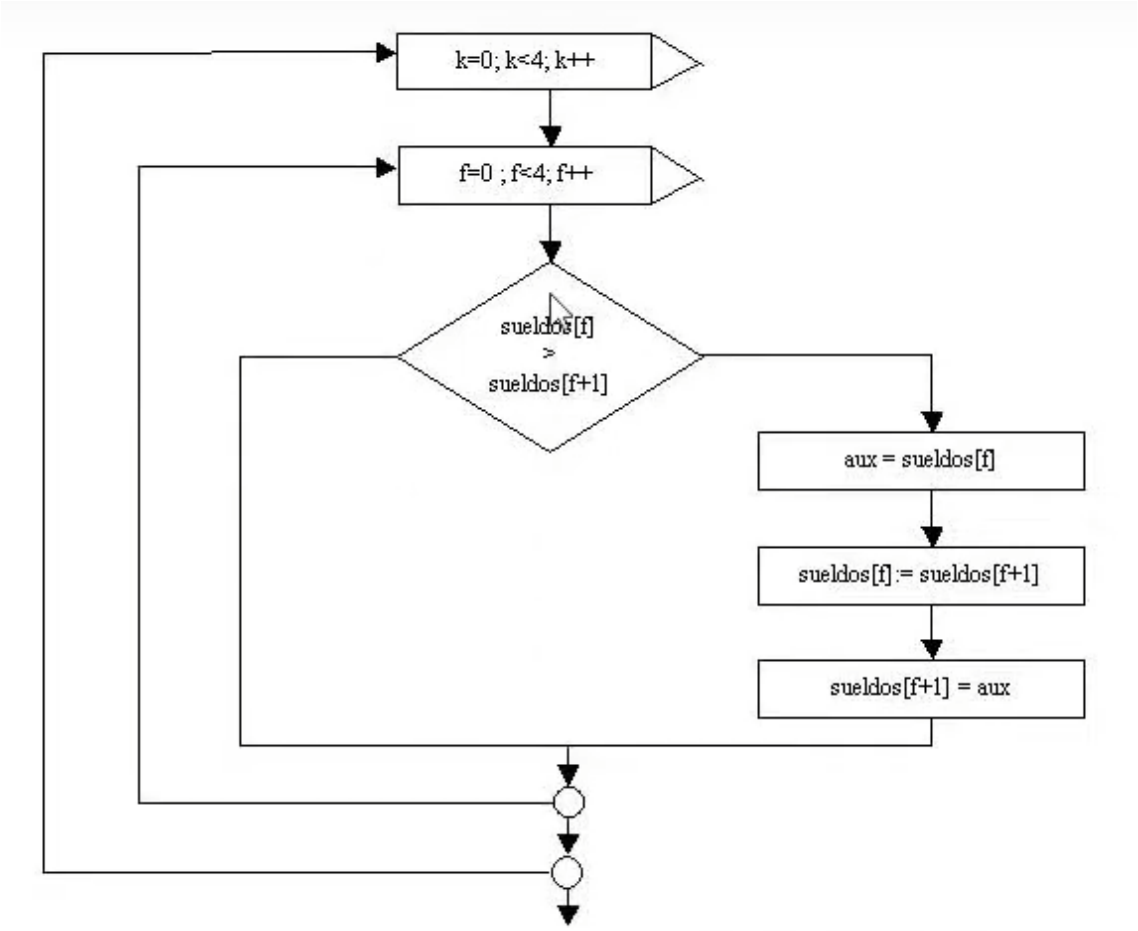
Analizando el algoritmo podemos comprobar que el elemento mayor del vector se ubica ahora en el último lugar.

Podemos definir otros vectores con distintos valores y comprobar que siempre el elemento mayor queda al final.

Pero todavía con este algoritmo no se ordena el vector. Solamente está ordenado el último elemento del vector.

Ahora bien, con los 4 elementos que nos queda podemos hacer el mismo proceso visto anteriormente, con lo cual quedará ordenado otro elemento del vector. Este proceso lo repetiremos hasta que quede ordenado por completo el vector.

Como debemos repetir el mismo algoritmo podemos englobar todo el bloque en otra estructura repetitiva.



Realizamos una prueba del siguiente algoritmo:

Cuando k = 0

f = 0	f = 1	f = 2	f = 3
750	750	750	750
1200	820	820	820
820	1200	550	550
550	550	1200	490
490	490	490	1200

Cuando k = 1

f = 0	f = 1	f = 2	f = 3
750	750	750	750
820	550	550	550
550	820	490	490
490	490	820	820
1200	1200	1200	1200

Cuando k = 2

f = 0	f = 1	f = 2	f = 3
550	550	550	550
750	490	490	490
490	750	750	750
820	820	820	820
1200	1200	1200	1200

Cuando k = 3

f = 0	f = 1	f = 2	f = 3
490	490	490	490
550	550	550	550

```
750             750             750             750
820             820             820             820
1200            1200            1200            1200
```

¿Por qué repetimos 4 veces el for externo?

Como sabemos cada vez que se repite en forma completa el for interno queda ordenado un elemento del vector. A primera vista diríamos que deberíamos repetir el for externo la cantida de componentes del vector, es este ejemplo el vector sueldos tiene 5 elementos.

Si observamos, cuando quedan dos elementos por ordenar, al ordenar uno de ellos que el otro automáticamente ordenado (podemos imaginar que si tenemos un vector con 2 elementos o se requiere el for externo, porque este debería repetirse una única vez).

Una última consideración a este ALGORITMO de ordenamiento es que los elementos que se van ordenando continuamos comparándolos.

ejemplo: En la primera ejecución del for interno el valor 1200 queda ubicado en la posición 4 del vector. En la segunda ejecución comparamos si 820 es mayor a 1200, lo cual seguramente será falso.

Podemos concluir que la primera vez debemos hacer para que este ejemplo 4 comparaciones, en la segunda ejecución del for interno debemos hacer 3 comparaciones y en general debemos ir reduciendo en una la cantidad de comparaciones.

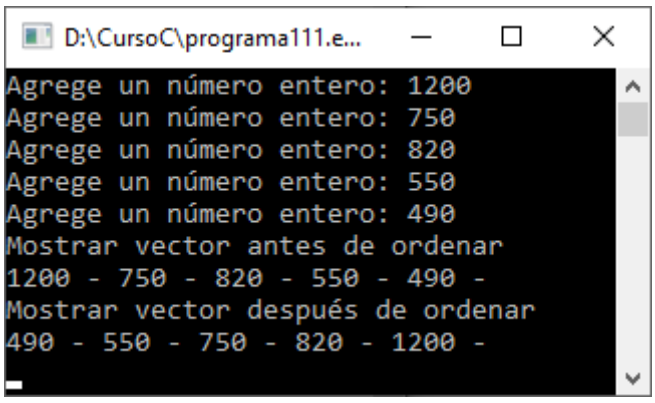
Si bien el algoritmo planteado funciona, un algoritmo más eficiente, que se deriva del anterior es el plantear un for interno con la siguiente estructura: (f=0; f<4-k, f++).

Es decir restarle el valor del contador externo.

```c
#include<stdio.h>
#include<conio.h>
#include<locale.h>

void agregar(int vec[5])
{
    for (int x=0; x<5; x++)
    {
        printf("Agrege un número entero: ");
        scanf("%i", &vec[x]);
    }
}

void imprimir(int vec[5])
{
    for (int x=0; x<5; x++)
    {
        printf("%i - ",vec[x]);
    }
    printf("\n");
}

```

```c
void ordenar(int vec[5])
{
    int aux;
    for (int k=0; k<4; k++)
    {
        for (int f=0; f<4-k; f++)
        {
            if (vec[f]>vec[f+1])
            {
                aux=vec[f];
                vec[f]=vec[f+1];
                vec[f+1]=aux;
            }
        }
    }
}

int main()
{
    setlocale(LC_ALL, "");
    int vector[5];
    agregar(vector);
    printf("Mostrar vector antes de ordenar\n");
    imprimir(vector);
    ordenar(vector);
    printf("Mostrar vector después de ordenar\n");
    imprimir(vector);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 114.- Vectores ordenamiento – 2

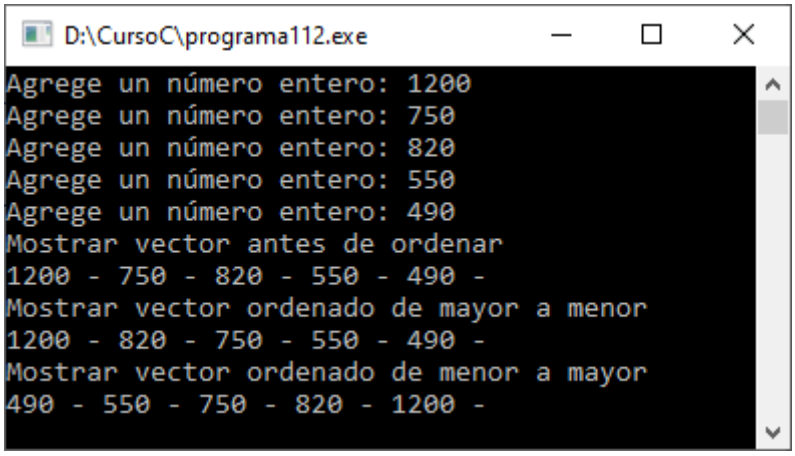
Problema propuesto

Cargar un vector de 5 elementos enteros. Ordenarlo de mayor a menor y postrarlo por pantalla, luego ordenar de menor a mayor e imprimir nuevamente.

```
#include<stdio.h>
#include<conio.h>
#include<locale.h>

void agregar(int vec[5])
{
    for (int x=0; x<5; x++)
    {
        printf("Agrege un número entero: ");
        scanf("%i", &vec[x]);
    }
}

void imprimir(int vec[5])
{
    for (int x=0; x<5; x++)
    {
        printf("%i - ",vec[x]);
    }
    printf("\n");
}

void ordMayorMenor(int vec[5])
{
    int aux;
    for (int k=0; k<4; k++)
    {
        for (int f=0; f<4-k; f++)
        {
            if (vec[f]<vec[f+1])
            {
                aux=vec[f];
                vec[f]=vec[f+1];
                vec[f+1]=aux;
            }
        }
    }
}

void ordMenorMayor(int vec[5])
{
    int aux;
    for (int k=0; k<4; k++)
    {
        for (int f=0; f<4-k; f++)
        {
            if (vec[f]>vec[f+1])
```

```
48              {
49                  aux=vec[f];
50                  vec[f]=vec[f+1];
51                  vec[f+1]=aux;
52              }
53          }
54      }
55  }
56
57  int main()
58  {
59      setlocale(LC_ALL, "");
60      int vector[5];
61      agregar(vector);
62      printf("Mostrar vector antes de ordenar\n");
63      imprimir(vector);
64      ordMayorMenor(vector);
65      printf("Mostrar vector ordenado de mayor a menor\n");
66      imprimir(vector);
67      ordMenorMayor(vector);
68      printf("Mostrar vector ordenado de menor a mayor\n");
69      imprimir(vector);
70      getch();
71      return 0;
72  }
73
```

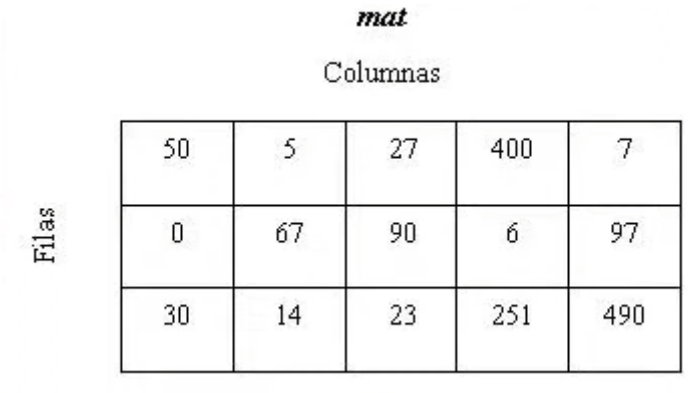
Si ejecutamos este será el resultado:



Capítulo 115.- Estructura de datos tipo matriz (elementos int y float) – 1

Una matriz es una estructura de datos que permite almacenar un CONJUNTO de datos del MISMO tipo.

Con un único nombre se define la matriz y por medio de DOS subíndices hacemos referencia a cada elemento de la misma.



Hemos graficado una matriz de 3 filas y 5 columnas. Para hacer referencia a cada elemento debemos indicar primero la fila y luego la columna, por ejemplo en el elemento 1, 4 se almacena el valor 97.

En este ejemplo almacenamos valores enteros. Todos los elementos de la matriz deben ser del mismo tipo.

Las filas y columnas a numerarse a partir de cero, similar a los vectores.

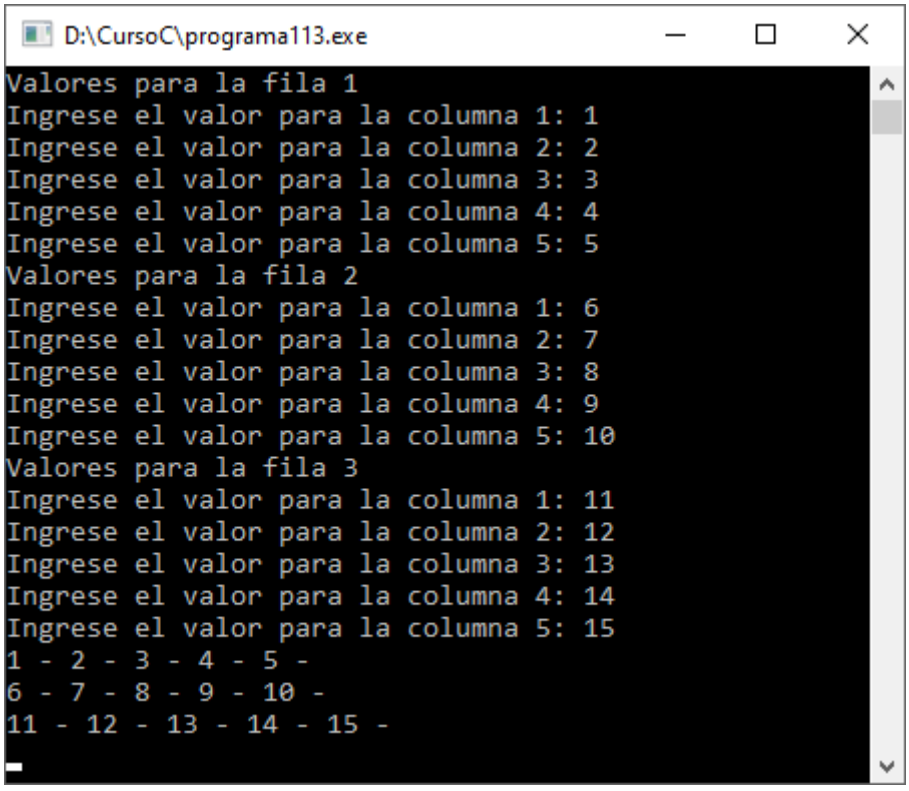
Problema

Crea un matriz de 3 filas por 5 columnas con elementos de tipo int, carga sus elementos y luego imprimirlas.

```
#include<stdio.h>
#include<conio.h>

void cargar(int ma[3][5])
{
    for (int x=0; x<3; x++)
    {
        printf("Valores para la fila %i\n", x+1);
        for (int y=0; y<5; y++)
        {
            printf("Ingrese el valor para la columna %i: ", y+1);
            scanf("%i", &ma[x][y]);
        }
    }
}

```

```
17  void imprimir(int ma[3][5])
18  {
19      for (int x=0; x<3; x++)
20      {
21          for(int y=0; y<5; y++)
22          {
23              printf("%i - ", ma[x][y]);
24          }
25          printf("\n");
26      }
27  }
28
29  int main()
30  {
31      int matriz[3][5];
32      cargar(matriz);
33      imprimir(matriz);
34      getch();
35      return 0;
36  }
37
```

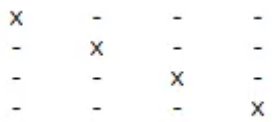
Si ejecutamos este será el resultado:



Capítulo 116.- Estructura de datos tipo matriz (elementos int y float) – 2

Problema

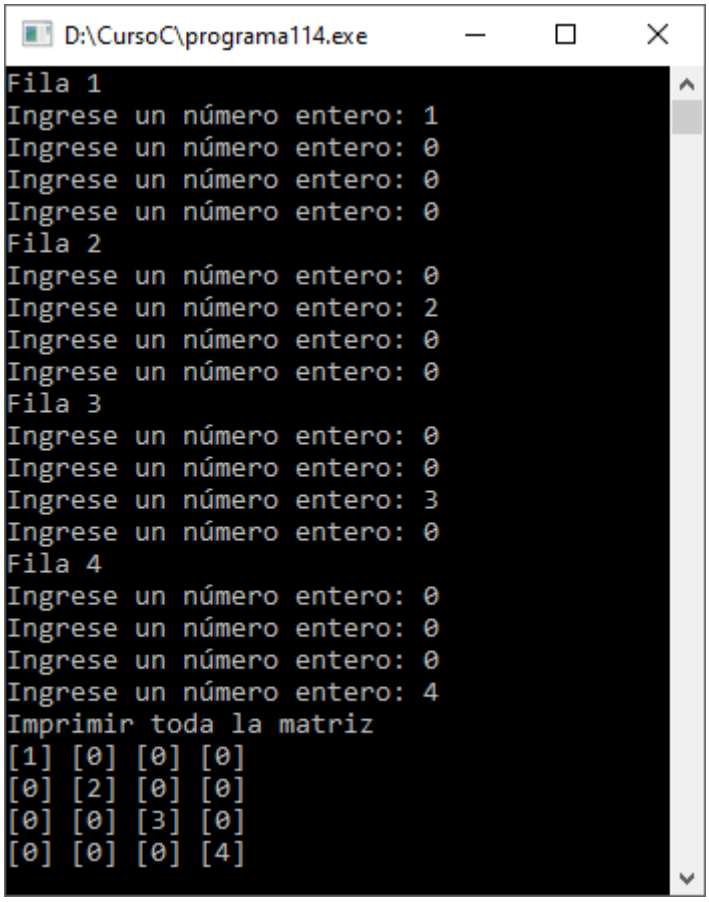
Crear y cargar una matriz de 4 filas por 4 columnas. Imprimir la diagonal principal.




```
#include<stdio.h>
#include<conio.h>
#include<locale.h>

void cargar(int ma[4][4])
{
    for (int x=0; x<4; x++)
    {
        printf("Fila %i\n", x+1);
        for (int y=0; y<4; y++)
        {
            printf("Ingrese un número entero: ");
            scanf("%i",&ma[x][y]);
        }
    }
}

void imprimir(int ma[4][4])
{
    for(int x=0; x<4; x++)
    {
        for (int y=0; y<4; y++)
        {
            printf("[%i] ", ma[x][y]);
        }
        printf("\n");
    }
}

void imprimirDiagonal(int ma[4][4])
{
    for (int x=0; x<4; x++)
    {
        printf("[%i] ", ma[x][x]);
    }
}


```

```c
int main()
{
    setlocale(LC_ALL, "");
    int matriz[4][4];
    cargar(matriz);
    printf("Imprimir toda la matriz\n");
    imprimir(matriz);
    printf("Imprimir la diagonal\n");
    imprimirDiagonal(matriz);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 117.- Estructura de datos tipo matriz (elementos int y float) – 3

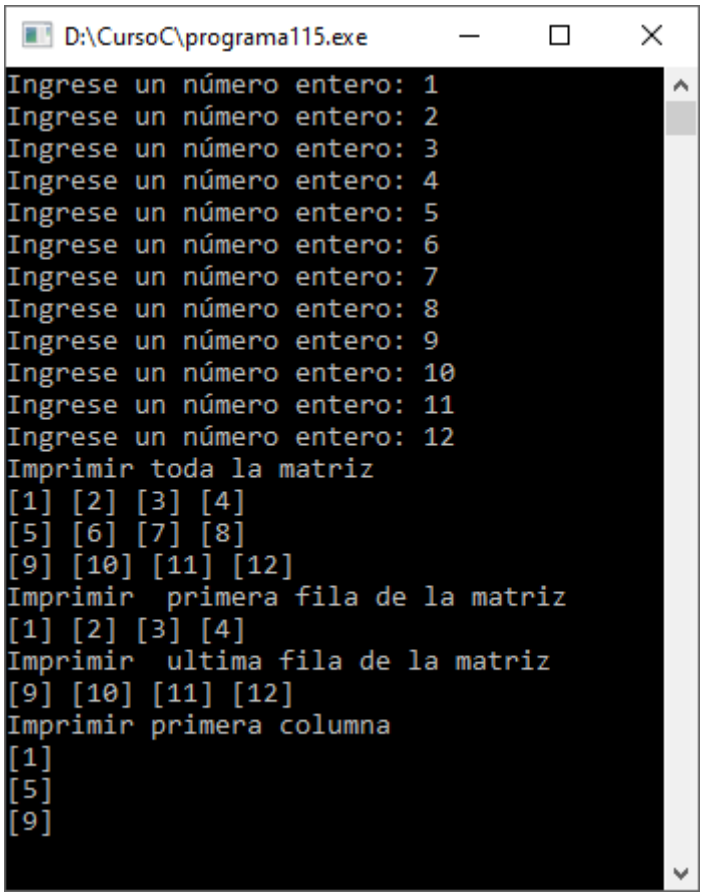
Problema

Crear una matriz de 3 filas por 4 columnas. Imprimir la primera fila. Imprimir la última fila e imprimir la primera columna.

```c
#include<stdio.h>
#include<conio.h>
#include<locale.h>

void cargar(int ma[3][4])
{
    for (int x=0; x<3; x++)
    {
        for (int y=0; y<4; y++)
        {
            printf("Ingrese un número entero: ");
            scanf("%i", &ma[x][y]);
        }
    }
}

void impMatriz(int ma[3][4])
{
    for (int x=0; x<3; x++)
    {
        for (int y=0; y<4; y++)
        {
            printf("[%i] ", ma[x][y]);
        }
        printf("\n");
    }
}

void impPrimeraFila(int ma[3][4])
{
    for (int x=0; x<4; x++)
    {
        printf("[%i] ", ma[0][x]);
    }
    printf("\n");
}

void impUltimaFila(int ma[3][4])
{
    for (int x=0; x<4; x++)
    {
        printf("[%i] ", ma[2][x]);
    }
    printf("\n");
}

```

```c
void impPrimeraColumna(int ma[3][4])
{
    for (int x=0; x<3; x++)
    {
        printf("[%i] \n", ma[x][0]);
    }
    printf("\n");
}

int main()
{
    setlocale(LC_ALL, "");
    int matriz[3][4];
    cargar(matriz);
    printf("Imprimir toda la matriz\n");
    impMatriz(matriz);
    printf("Imprimir  primera fila de la matriz\n");
    impPrimeraFila(matriz);
    printf("Imprimir  ultima fila de la matriz\n");
    impUltimaFila(matriz);
    printf("Imprimir primera columna\n");
    impPrimeraColumna(matriz);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 118.- Estructura de datos tipo matriz (elementos int y float) – 4

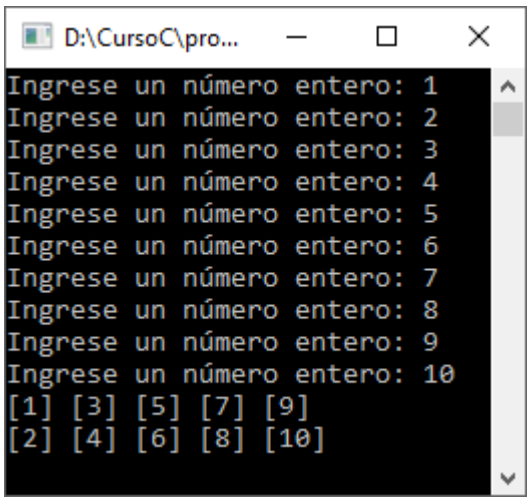
Problema propuesto

Crear una matriz de 2 filas y 5 columnas. Realizar la carga de componentes por columna (es decir primer ingresar toda la primera columna, luego la segunda columna y así sucesivamente).

Imprimir la matriz.

```c
#include<stdio.h>
#include<conio.h>
#include<locale.h>

void cargar(int ma[2][5])
{
    for (int x=0; x<5; x++)
    {
        for (int y=0; y<2; y++)
        {
            printf("Ingrese un número entero: ");
            scanf("%i", &ma[y][x]);
        }
    }
}

void imprimir(int ma[2][5])
{
    for(int x=0; x<2; x++)
    {
        for (int y=0; y<5; y++)
        {
            printf("[%i] ", ma[x][y]);
        }
        printf("\n");
    }
}

int main()
{
    setlocale(LC_ALL, "");
    int matriz[2][5];
    cargar(matriz);
    imprimir(matriz);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 119.- Estructura de datos tipo matriz (elementos int y float) – 5

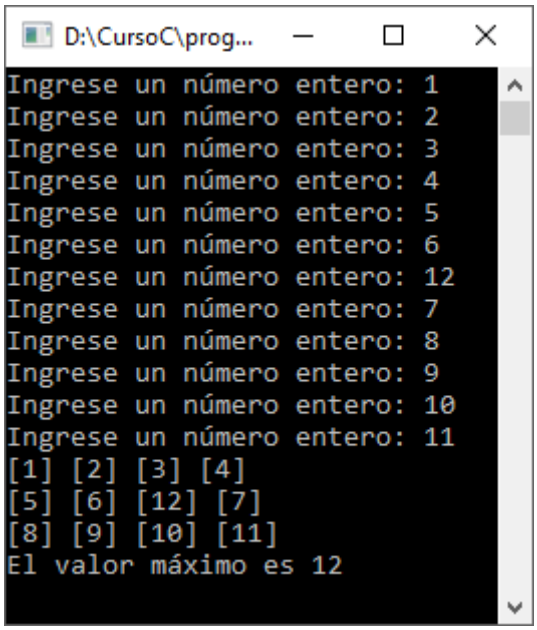
Problema propuesto

Crear un matriz de 3x4. Realizar la carga y luego imprimir el elemento mayor.

```
#include<stdio.h>
#include<conio.h>
#include<locale.h>

void cargar(int ma[3][4])
{
    for (int x=0; x<3; x++)
    {
        for (int y=0; y<4; y++)
        {
            printf("Ingrese un número entero: ");
            scanf("%i", &ma[x][y]);
        }
    }
}

void imprimir(int ma[3][4])
{
    for (int x=0; x<3; x++)
    {
        for (int y=0; y<4; y++)
        {
            printf("[%i] ", ma[x][y]);
        }
        printf("\n");
    }
}

void valorMayor(int ma[3][4])
{
    int max=ma[0][0];
    for (int x=0; x<3; x++)
    {
        for (int y=0; y<4; y++)
        {
            if(ma[x][y]> max)
            {
                max=ma[x][y];
            }
        }
    }
    printf("El valor máximo es %i", max);
}

```

```
45  int main()
46  {
47      setlocale(LC_ALL, "");
48      int matriz[3][4];
49      cargar(matriz);
50      imprimir(matriz);
51      valorMayor(matriz);
52      getch();
53      return 0;
54  }
55
```

Si ejecutamos este será el resultado:



Capítulo 120.- Estructura de datos tipo matriz (elementos int y float) – 6

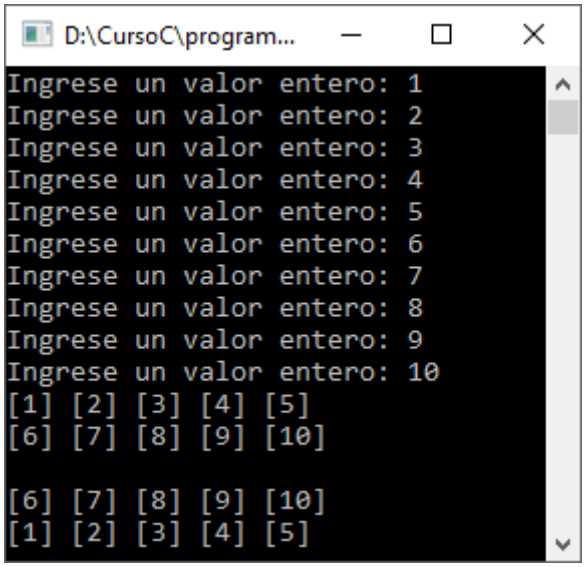
Problema propuesto

Definir una matriz de 2 filas y 5 columnas. Realizar su carga e impresión.
Intercambiar los elementos de la primera fila con la segunda y volver a imprimir la matriz.

```
#include<stdio.h>
#include<conio.h>

void cargar(int ma[2][5])
{
    for (int x=0; x<2; x++)
    {
        for (int y=0; y<5; y++)
        {
            printf("Ingrese un valor entero: ");
            scanf("%i", &ma[x][y]);
        }
    }
}

void imprimir(int ma[2][5])
{
    for (int x=0; x<2; x++)
    {
        for (int y=0; y<5; y++)
        {
            printf("[%i] ", ma[x][y]);
        }
       printf("\n");
    }
    printf("\n");
}

void intercambiar(int ma[2][5])
{
    int aux;
    for (int x=0; x<5; x++)
    {
        aux = ma[0][x];
        ma[0][x]=ma[1][x];
        ma[1][x]=aux;
    }
}

int main()
{
    int matriz[2][5];
    cargar(matriz);
    imprimir(matriz);
    intercambiar(matriz);
    imprimir(matriz);
```

```
47      getch();
48      return 0;
49  }
50
```

Si ejecutamos este será el resultado:



Capítulo 121.- Estructura de datos tipo matriz (elementos char) – 1

Las matrices de tipo de dato char se utilizan fundamentalmente para guardar un conjunto de cadenas de caracteres (varios nombres de personas, nombres de artículos, etc.).

Cuando trabajamos una matriz de tipo char para almacenar cadenas de caracteres cada fila almacena un string:

m	a	n	z	a	n	a	s	\0	
p	e	r	a	s	\0				
n	a	r	a	n	j	a	s	\0	

Para trabajar una matriz de tipo char se parece a trabajar con un vector, debemos indicar un solo subíndice y estaremos accediendo a toda la fila.

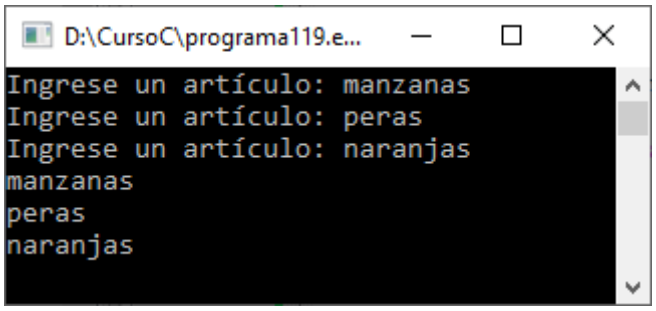
Problema

Confeccionar un programa que permita ingresar en una matriz el tipo char los nombres de artículos para la venta (3 nombres de artículos).

Hacer luego una función que imprima los nombres de dichos artículos.

```
#include<stdio.h>
#include<conio.h>
#include<locale.h>

void cargar(char art[3][10])
{
    for (int x=0; x<3; x++)
    {
        printf("Ingrese un artículo: ");
        gets(art[x]);
    }
}

void imprimir(char art[3][10])
{
    for (int x=0; x<3; x++)
    {
       printf("%s\n",art[x]);
    }
    printf("\n");
}

int main()
{
    setlocale(LC_ALL, "");
    char articulos[3][10];
    cargar(articulos);
    imprimir(articulos);
    getch();
    return 0;
}

```

Si ejecutamos este será el resultado:



Capítulo 122.- Estructura de datos tipo matriz (elementos char) – 2

Problema

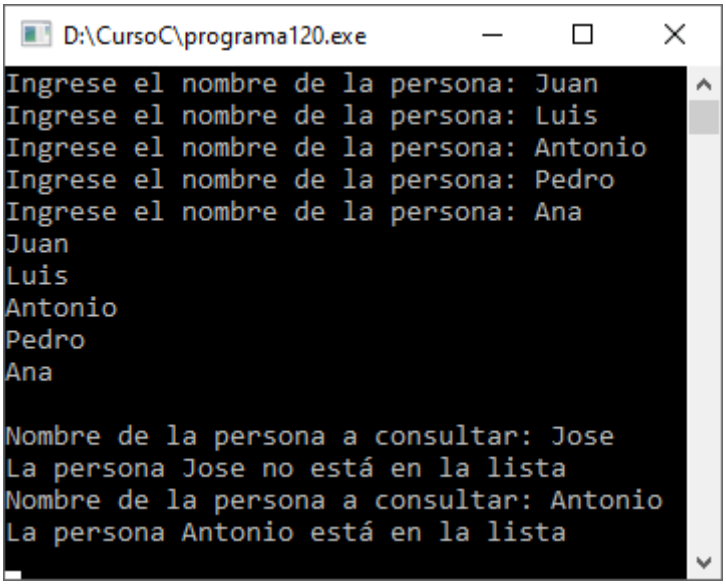
Confeccionar un programa que permita:

1- Almacenar en una matriz los datos de 5 personas.
2- Imprimir los nombres.
3- Ingresar otro nombre y verificar si se encuentra almacenado en la matriz.

```
#include<stdio.h>
#include<conio.h>
#include<locale.h>
#include<string.h>

void cargar(char per[5][31])
{
    for (int x=0; x<5; x++)
    {
        printf("Ingrese el nombre de la persona: ");
        gets(per[x]);
    }
}

void imprimir(char per[5][31])
{
    for (int x=0; x<5; x++)
    {
        printf("%s\n", per[x]);
    }
    printf("\n");
}

void buscar(char per[5][31])
{
    char perso[31];
    int encontrado=0;
    printf("Nombre de la persona a consultar: ");
    gets(perso);
    for (int x=0; x<5; x++)
    {
        if (strcmp(perso, per[x])==0)
            encontrado=1;
    }
    if (encontrado==1)
    {
        printf("La persona %s está en la lista\n",perso);
    }
    else
    {
        printf("La persona %s no está en la lista\n", perso);
    }
}
```

```
44
45  int main()
46  {
47      setlocale(LC_ALL, "");
48      char personas[5][31];
49      cargar(personas);
50      imprimir(personas);
51      buscar(personas);
52      buscar(personas);
53      getch();
54      return 0;
55  }
56
```

Si ejecutamos este será el resultado:



Capítulo 123.- Estructura de datos tipo matriz (elementos char) – 3

Problema

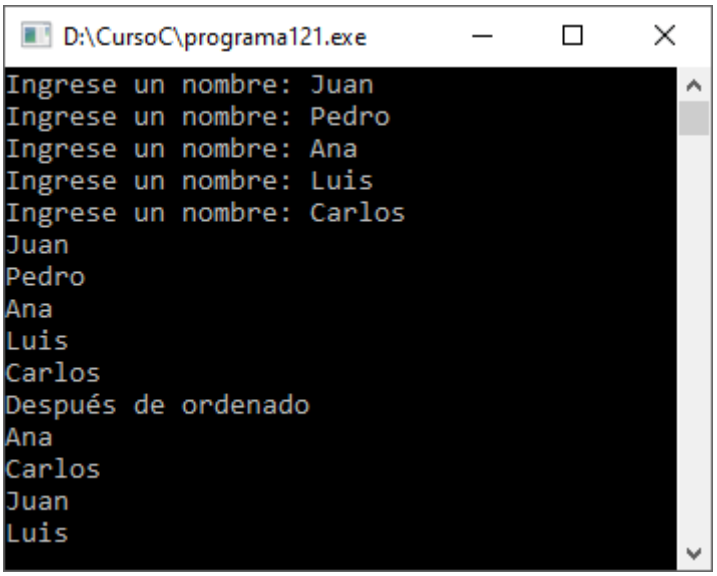
Confeccionar un programa que permita:

1- Almacenar en una matriz los datos de 5 personas.
2- Imprimir los nombres.
3- Ordenar alfabéticamente los nombres.

```
#include<stdio.h>
#include<conio.h>
#include<locale.h>
#include<string.h>

void cargar(char per[5][31])
{
    for (int x=0; x<5; x++)
    {
        printf("Ingrese un nombre: ");
        gets(per[x]);
    }
}

void imprimir(char per[5][31])
{
    for (int x=0; x<5; x++)
    {
        printf("%s\n", per[x]);
    }
}

void ordenar(char per[5][31])
{
    char aux[31];
    for (int x=0; x<4; x++)
    {
        for (int y=0; y<4-x; y++)
        {
            if (strcmp(per[y], per[y+1])>0)
            {
                strcpy(aux, per[y]);
                strcpy(per[y],per[y+1]);
                strcpy(per[y+1], aux);
            }
        }
    }
}
```

```
39
40  int main()
41  {
42      setlocale(LC_ALL, "");
43      char personas[5][31];
44      cargar(personas);
45      imprimir(personas);
46      ordenar(personas);
47      printf("Después de ordenado\n");
48      imprimir(personas);
49      getch();
50      return 0;
51  }
52
```

Si ejecutamos este será el resultado:



Capítulo 124.- Estructura de datos tipo matriz (elementos char) – 4

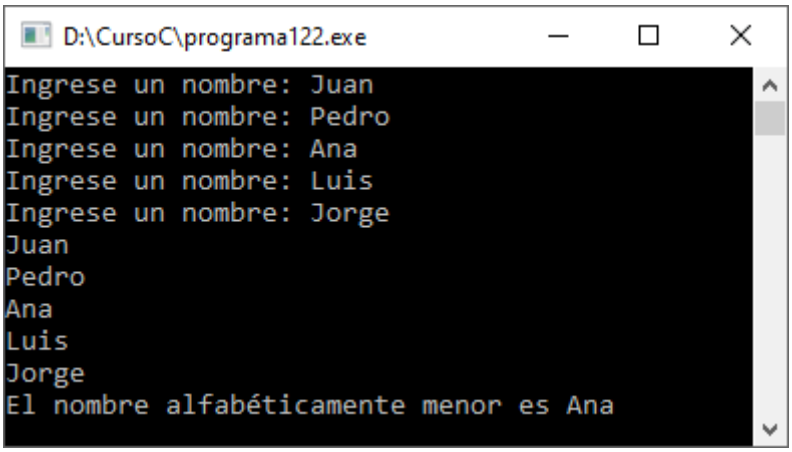
Problema propuesto

Confeccionar un programa que permita:

1- Almacenar un matriz los datos de 5 personas.
2- Imprimir el nombre alfabéticamente menor.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<locale.h>

void cargar(char per[5][31])
{
    for (int x=0; x<5; x++)
    {
        printf("Ingrese un nombre: ");
        gets(per[x]);
    }
}

void imprimir(char per[5][31])
{
    for (int x=0; x<5; x++)
    {
        printf("%s\n", per[x]);
    }
}

void menor(char per[5][31])
{
    char menor[31];
    strcpy(menor, per[1]);
    for(int x=1; x<5; x++)
    {
        if(strcmp(per[x], menor)<0)
        {
            strcpy(menor, per[x]);
        }
    }
    printf("El nombre alfabéticamente menor es %s", menor);
}

int main()
{
    setlocale(LC_ALL, "");
    char personas[5][31];
    cargar(personas);
    imprimir(personas);
    menor(personas);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 125.- Vectores y matrices paralelas – 1

Este concepto se da cuando hay una relación entre los elementos de igual subíndice (misma posición) de un vector y otro, o de un vector con una matriz de caracteres.

nombres	Juan	Ana	Marcos	Pablo	Laura
edades	12	21	27	14	21

Si tenemos una matriz de caracteres de 5 filas y 30 columnas en la que se almacena los nombres de personas y un vector de 5 enteros en la que se almacenan las edades de dichas personas, decimos que la matriz nombre es paralelo al vector edades si en el elemento 0 del vector y la fila 0 de la matiz se almacena información relacionada a una persona (Juan – 12 años).

Es decir hay una relación entre cada elemento del vector y la fila de la matriz.

Esta relación la conoce únicamente el programador y se hace para facilitar el desarrollo de algoritmos que procesan los datos almacenados en las estructuras de datos.

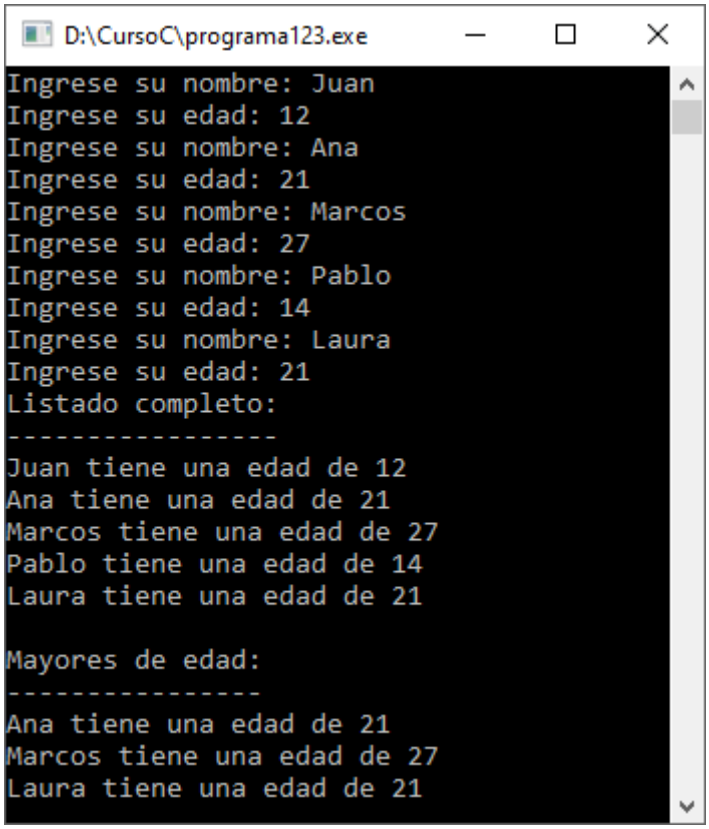
Problema

Desarrollar un programa que permita cargar 5 nombres de personas y sus edades respectivas. Luego de realizar la carga por teclado de todos los datos imprimir los nombres de las personas mayores de edad (mayores o iguales a 18 años).

```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<locale.h>
4
5   void cargar(char per[5][31], int ed[5])
6   {
7       for(int x=0; x<5; x++)
8       {
9           printf("Ingrese su nombre: ");
10          fflush(stdin);
11          gets(per[x]);
12          printf("Ingrese su edad: ");
13          scanf("%i", &ed[x]);
14      }
15  }
16
17  void imprimir(char per[5][31], int ed[5])
18  {
19      printf("Listado completo:\n");
20      printf("-----------------\n");
21      for (int x=0; x<5; x++)
22      {
23          printf("%s tiene una edad de %i\n", per[x], ed[x]);
24      }
25      printf("\n");
26  }
27
```

```c
void mayorEdad(char per[5][31], int ed[5])
{
    printf("Mayores de edad:\n");
    printf("----------------\n");
    for (int x=0; x<5; x++)
    {
        if (ed[x]>=18)
        {
            printf("%s tiene una edad de %i\n", per[x], ed[x]);
        }
    }
    printf("\n");
}

int main()
{
    setlocale(LC_ALL, "");
    char persona[5][31];
    int edad[5];
    cargar(persona, edad);
    imprimir(persona, edad);
    mayorEdad(persona, edad);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 126.- Vectores y matrices paralelas – 2

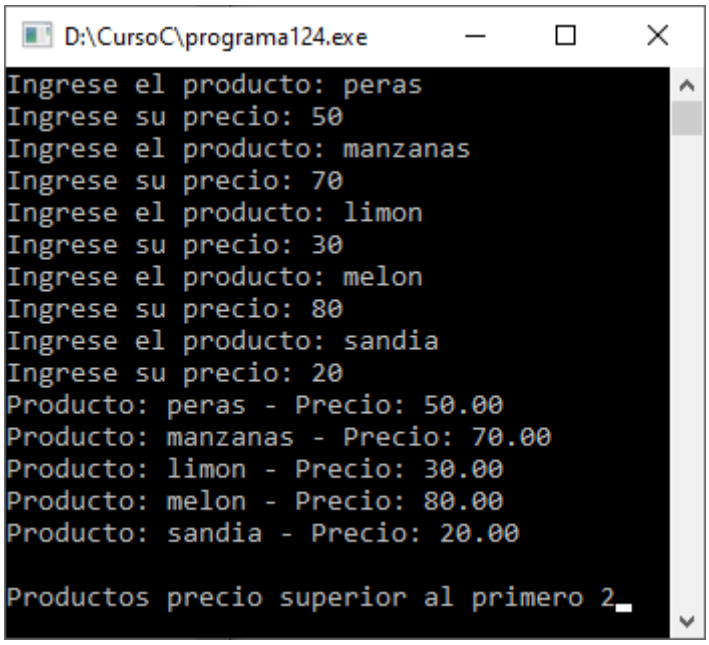
Problema propuesto

Ingresar el nombre de 5 productos en una matriz de caracteres y sus respectivos precios en un vector paralelo de tipo float.

Mostrar cuantos productos tiene un precio mayor al primer producto ingresado (se debe contar)

```
#include<stdio.h>
#include<conio.h>

void cargar(char producto[5][31], float imp[5])
{
    for(int x=0; x<5; x++)
    {
        printf("Ingrese el producto: ");
        fflush(stdin);
        gets(producto[x]);
        printf("Ingrese su precio: ");
        scanf("%f", &imp[x]);
    }
}

void imprimir(char producto[5][31], float imp[5])
{
    for (int x=0; x<5; x++)
    {
        printf("Producto: %s - Precio: %0.2f\n", producto[x], imp[x]);
    }
    printf("\n");
}

void precioMayor(char producto[5][31], float imp[5])
{
    int cant=0;
    for (int x=1; x<5; x++)
    {
        if(imp[x]>imp[0])
            cant++;
    }
    printf("Productos precio superior al primero %i", cant);
}

int main()
{
    char productos[5][31];
    float importe[5];
    cargar(productos, importe);
    imprimir(productos, importe),
    precioMayor(productos, importe);
    getch();
    return 0;
}

```

Si ejecutamos este será el resultado:


D:\CursoC\programa124.exe
Ingrese el producto: peras
Ingrese su precio: 50
Ingrese el producto: manzanas
Ingrese su precio: 70
Ingrese el producto: limon
Ingrese su precio: 30
Ingrese el producto: melon
Ingrese su precio: 80
Ingrese el producto: sandia
Ingrese su precio: 20
Producto: peras - Precio: 50.00
Producto: manzanas - Precio: 70.00
Producto: limon - Precio: 30.00
Producto: melon - Precio: 80.00
Producto: sandia - Precio: 20.00
Productos precio superior al primero 2

Capítulo 127.- Vectores y matrices paralelas – 3

Problema propuesto

En un curso de 4 alumnos se registran las notas de sus exámenes y se debe procesar de acuerdo a lo siguiente:

a) Ingresar Nombre y Nota de cada alumno (almacenar los datos en estructuras paralelas).
b) Realizar un listado que muestre los nombres, notas y condición del alumno. En la condición, colocar "Muy bueno" si la nota es mayor o igual a 8, "Bueno" si la nota está entre 4 y 7, y colocar "Insuficiente" si la nota es menor a 4.
c) Imprimir cuantos alumnos tienen la leyenda "Muy Bueno".

```c
#include<stdio.h>
#include<conio.h>
#include<locale.h>

void cargar(char nom[4][31], int no[4])
{
    for (int x=0; x<4; x++)
    {
        printf("Ingrese el nombre del alumno: ");
        fflush(stdin);
        gets(nom[x]);
        printf("Ingrese su nota: ");
        scanf("%i", &no[x]);
    }
}

void imprimir(char nom[4][31], int no[4])
{
    printf("Relación de notas por alumno\n");
    printf("----------------------------\n");
    for (int x=0; x<4; x++)
    {
        printf("El alumno: %s ha obtenido una calificación de %i", nom[x], no[x]);
        printf(" su valoración es ");
        if (no[x]>=8)
        {
            printf("'Muy bueno'\n");
        }
        else
        {
            if(no[x]>=4)
            {
               printf("'Bueno'\n");
            }
            else
            {
                printf("'Insuficiente'\n");
            }
        }
    }
    printf("----------------------------\n");
}
```

```c

void cantidadAlumnnosMuyBuenos(int no[4])
{
    int cant=0;
    for (int x=0; x<4; x++)
    {
        if(no[x]>=8)
           {
              cant++;
           }
    }
    printf("Cuantos alumnos con calificación 'Muy Bueno': %i", cant);
}

int main()
{
    setlocale(LC_ALL, "");
    char nombre[4][31];
    int nota[4];
    cargar(nombre,nota);
    imprimir(nombre, nota);
    cantidadAlumnnosMuyBuenos(nota);
    getch();
    return 0;
}

```

Si ejecutamos este será el resultado:

```
D:\CursoC\programa125.exe
Ingrese el nombre del alumno: Juan
Ingrese su nota: 2
Ingrese el nombre del alumno: Ana
Ingrese su nota: 4
Ingrese el nombre del alumno: Luis
Ingrese su nota: 8
Ingrese el nombre del alumno: Carlos
Ingrese su nota: 9
Relación de notas por alumno
----------------------------
El alumno: Juan ha obtenido una calificación de 2 su valoración es 'Insuficiente'
El alumno: Ana ha obtenido una calificación de 4 su valoración es 'Bueno'
El alumno: Luis ha obtenido una calificación de 8 su valoración es 'Muy bueno'
El alumno: Carlos ha obtenido una calificación de 9 su valoración es 'Muy bueno'
----------------------------
Cuantos alumnos con calificación 'Muy Bueno': 2
```

Capítulo 128.- Vectores y matrices paralelas (ordenamiento) – 1

Cuando se tienen vectores y matrices paralelas y se ordena uno de ellos hay que tener la precaución de intercambiar los elementos de los vectores o matrices paralelas.

Problema

Confeccionar un programa que permita cargar los nombres de 5 alumnos y sus notas respectivas. Luego ordenar las notas de mayor a menor.

Imprimir las notas y los nombres de los alumnos.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<locale.h>

void cargar(char nom[5][31], int no[5])
{
    for (int x=0; x<5; x++)
    {
        printf("Ingrese el nombre del alumno: ");
        gets(nom[x]);
        printf("Ingrese su nota: ");
        scanf("%i", &no[x]);
        fflush(stdin);
    }
}

void imprimir(char nom[5][31], int no[5])
{
    printf("Listado de calificaciones\n");
    printf("-------------------------\n");
    for (int x=0; x<5; x++)
    {
        printf("El alumno: %s ha obtenido ",nom[x]);
        printf("una calificacion de %i\n", no[x]);
    }
}

void ordenar(char nom[5][31], int no[5])
{
    int auxNotas;
    char auxNombres[31];
    for (int x=0; x<4; x++)
    {
        for (int y=0; y<4-x; y++)
        {
            if(no[y]<no[y+1])
            {
                auxNotas=no[y];
                no[y]=no[y+1];
                no[y+1]=auxNotas;
                strcpy(auxNombres, nom[y]);
                strcpy(nom[y],nom[y+1]);
```

```
44                  strcpy(nom[y+1],auxNombres);
45              }
46          }
47      }
48  }
49
50  int main()
51  {
52      setlocale(LC_ALL, "");
53      char nombres[5][31];
54      int notas[5];
55      cargar(nombres, notas);
56      imprimir(nombres, notas);
57      ordenar(nombres, notas);
58      printf("Después de ordenar de mayor a menor\n");
59      imprimir(nombres, notas);
60      getch();
61      return 0;
62  }
63
```

Si ejecutamos este será el resultado:

```
D:\CursoC\programa126.exe
Ingrese el nombre del alumno: Juan
Ingrese su nota: 2
Ingrese el nombre del alumno: Carlos
Ingrese su nota: 9
Ingrese el nombre del alumno: Laura
Ingrese su nota: 7
Ingrese el nombre del alumno: Maria
Ingrese su nota: 5
Ingrese el nombre del alumno: Diego
Ingrese su nota: 9
Listado de calificaciones
------------------------
El alumno: Juan ha obtenido una calificacion de 2
El alumno: Carlos ha obtenido una calificacion de 9
El alumno: Laura ha obtenido una calificacion de 7
El alumno: Maria ha obtenido una calificacion de 5
El alumno: Diego ha obtenido una calificacion de 9
Después de ordenar de mayor a menor
Listado de calificaciones
------------------------
El alumno: Carlos ha obtenido una calificacion de 9
El alumno: Diego ha obtenido una calificacion de 9
El alumno: Laura ha obtenido una calificacion de 7
El alumno: Maria ha obtenido una calificacion de 5
El alumno: Juan ha obtenido una calificacion de 2
```

Capítulo 129.- Vectores y matrices paralelas (ordenamiento) – 2

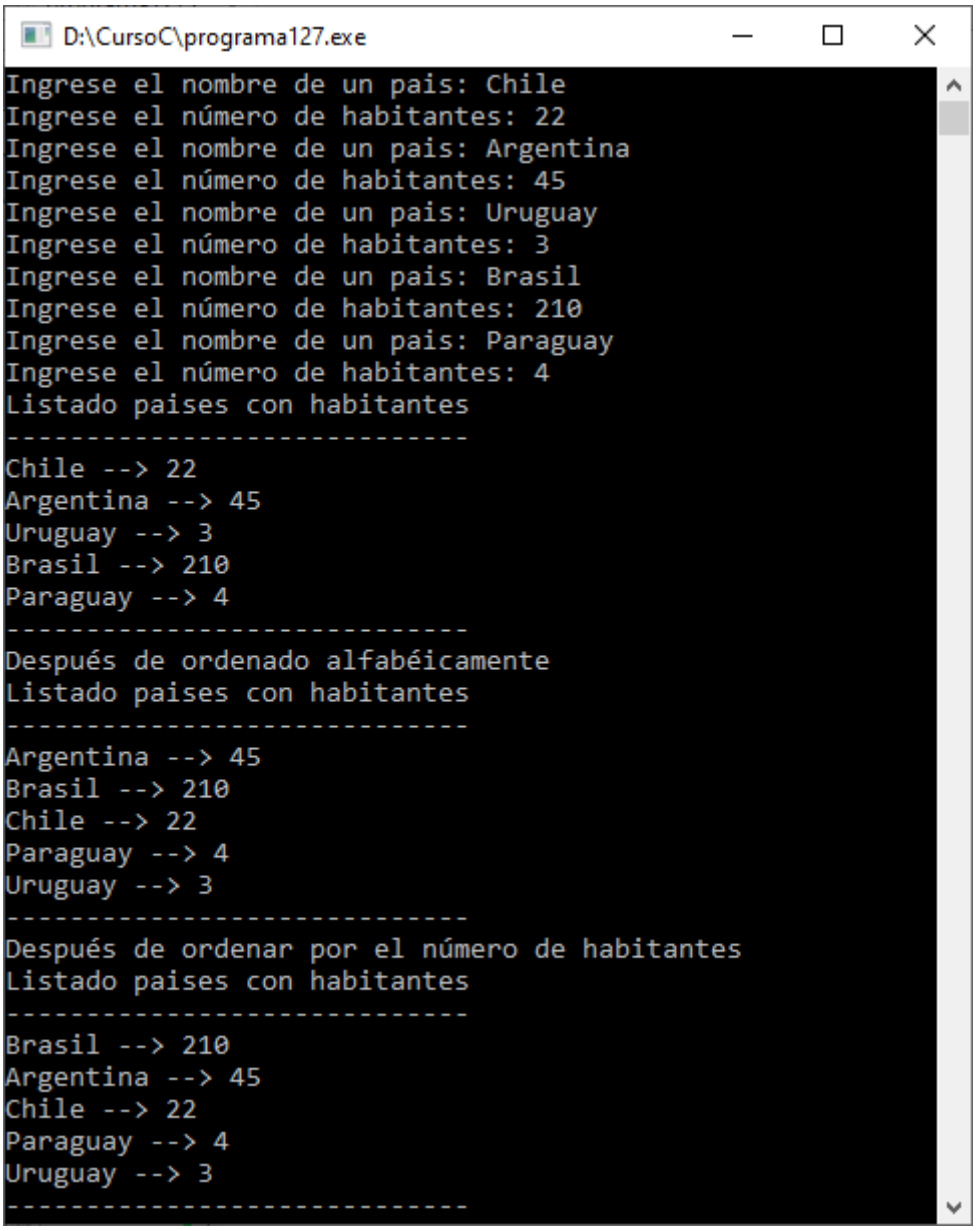
Problema propuesto

Cargar en una matriz los nombres de 5 países y en un vector paralelo la cantidad de habitantes del mismo. Ordenar alfabéticamente e imprimir el resultado. Por último ordenar con respecto a la cantidad de habitantes (de mayor a menor) e imprimir nuevamente.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<locale.h>

void cargar(char pais[5][31], int habi[5])
{
    for (int x=0; x<5; x++)
    {
        printf("Ingrese el nombre de un pais: ");
        gets(pais[x]);
        printf("Ingrese el número de habitantes: ");
        scanf("%i", &habi[x]);
        fflush(stdin);
    }
}

void imprimir(char pais[5][31], int habi[5])
{
    printf("Listado paises con habitantes\n");
    printf("-----------------------------\n");
    for (int x=0; x<5; x++)
    {
        printf("%s --> %i\n", pais[x], habi[x]);
    }
    printf("-----------------------------\n");
}

void ordenarAlfabeticamente(char pais[5][31], int habi[5])
{
    char auxpais[31];
    int auxhabi;
    for (int x=0; x<4; x++)
    {
       for (int y=0; y<4-x; y++)
       {
            if(strcmp(pais[y],pais[y+1])>0)
            {
                strcpy(auxpais, pais[y]);
                strcpy(pais[y], pais[y+1]);
                strcpy(pais[y+1], auxpais);

                auxhabi=habi[y];
                habi[y]=habi[y+1];
                habi[y+1]=auxhabi;
            }
```

```
47          }
48      }
49  }
50
51  void ordenarNumeroHabitantes(char pais[5][31], int habi[5])
52  {
53       char auxpais[31];
54      int auxhabi;
55      for (int x=0; x<4; x++)
56      {
57         for (int y=0; y<4-x; y++)
58         {
59              if(habi[y]<habi[y+1])
60              {
61                  auxhabi=habi[y];
62                  habi[y]=habi[y+1];
63                  habi[y+1]=auxhabi;
64
65                  strcpy(auxpais, pais[y]);
66                  strcpy(pais[y], pais[y+1]);
67                  strcpy(pais[y+1], auxpais);
68              }
69         }
70      }
71  }
72
73
74  int main()
75  {
76      setlocale(LC_ALL, "");
77      char paises[5][31];
78      int habitantes[5];
79      cargar(paises, habitantes);
80      imprimir(paises, habitantes);
81      ordenarAlfabeticamente(paises, habitantes);
82      printf("Después de ordenado alfabéicamente\n");
83      imprimir(paises, habitantes);
84      ordenarNumeroHabitantes(paises, habitantes);
85      printf("Después de ordenar por el número de habitantes\n");
86      imprimir(paises, habitantes);
87      getch();
88      return 0;
89  }
90
```

Si ejecutamos este será el resultado:


D:\CursoC\programa127.exe
Ingrese el nombre de un pais: Chile
Ingrese el número de habitantes: 22
Ingrese el nombre de un pais: Argentina
Ingrese el número de habitantes: 45
Ingrese el nombre de un pais: Uruguay
Ingrese el número de habitantes: 3
Ingrese el nombre de un pais: Brasil
Ingrese el número de habitantes: 210
Ingrese el nombre de un pais: Paraguay
Ingrese el número de habitantes: 4
Listado paises con habitantes
Chile --> 22
Argentina --> 45
Uruguay --> 3
Brasil --> 210
Paraguay --> 4
Después de ordenado alfabéicamente
Listado paises con habitantes
Argentina --> 45
Brasil --> 210
Chile --> 22
Paraguay --> 4
Uruguay --> 3
Después de ordenar por el número de habitantes
Listado paises con habitantes
Brasil --> 210
Argentina --> 45
Chile --> 22
Paraguay --> 4
Uruguay --> 3

Capítulo 130.- Directiva #define – 1

La directiva #define especifica un nombre que será reemplazado por un cierto valor en todos los lugares del programa donde se haga referencia a dicho nombre.

Se crean fuera de cualquier función normalmente en la parte inicial del archivo luego de las directivas #include.

La sintaxis es la siguiente:

```
#define [nombre de la macro]  [valor de la macro]
```

Un ejemplo concreto es:

```
#define TAMANO 20
#define MENSAJEFIN "Presione una tecla para finalizar"
```

El nombre de la macro es común que se escriba en mayúsculas (es una regla que utilizan mucho los programadores pero no es obligatoria).

Luego de especificar el nombre de la macro hay uno o más espacios en blanco y debemos especificar el valor de la macro.

El primer paso del compilador es reemplazar todas las partes del programa donde hay declaradas macros por los valores de dichas macros.

El uso excesivo de macros puede tornar muy difícil el mantenimiento del programa.

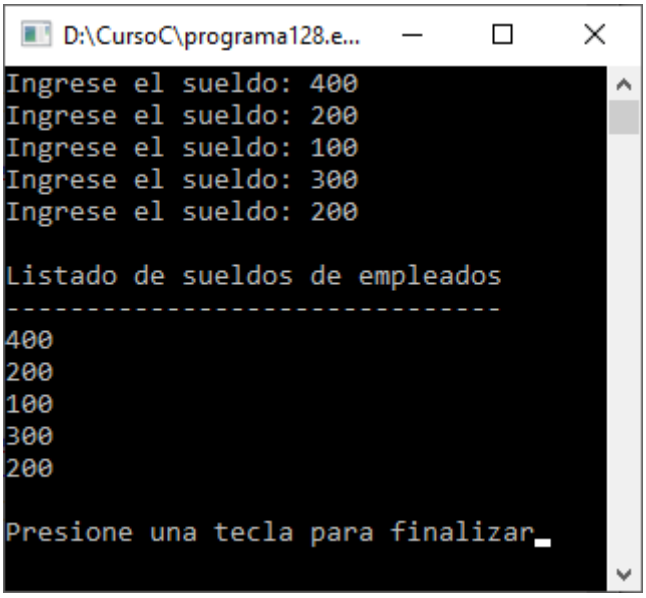
Problema

Se desea guardar los sueldos de 5 operarios.
Desarrollar dos funciones una donde los ingrese por teclado y otra función donde se los imprima.
Definir una macro para indicar el tamaño del vector.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   #define TAMANO 5
5   #define MENSAJEFIN "Presione una tecla para finalizar"
6
7   void cargar(int su[TAMANO])
8   {
9       for (int x=0; x<TAMANO; x++)
10      {
11          printf("Ingrese el sueldo: ");
12          scanf("%i", &su[x]);
13      }
14      printf("\n");
15  }
16
```

```
17  void imprimir(int su[TAMANO])
18  {
19      printf("Listado de sueldos de empleados\n");
20      printf("------------------------------\n");
21      for (int x=0; x<TAMANO; x++)
22      {
23          printf("%i\n", su[x]);
24      }
25      printf("\n");
26  }
27
28  int main()
29  {
30      int sueldo[TAMANO];
31      cargar(sueldo);
32      imprimir(sueldo);
33      printf(MENSAJEFIN);
34      getch();
35      return 0;
36  }
37
```

Si ejecutamos este será el resultado:



Capítulo 131.- Directiva #define – 2

Problema propuesto

Se tiene la siguiente información:

- Nombre de 4 empleados (matriz tipo char)
- Ingresos en conceptos de sueldos, cobrado por cada empleado, en los últimos 3 meses (matriz de tipo float).

Confeccionar el programa para:

a) Realizar la carga de la información mencionada.
b) Generar un vector que contenga el ingreso acumulado en los últimos 3 meses para cada empleado.
c) Mostrar por pantalla el total pagado en sueldos a todos los empleados en los últimos 3 meses.
d) Obtener el nombre del empleado que tuvo el mayor ingreso acumulado.

Utilizar macros para definir la cantidad de filas y columnas de las estructuras de datos.

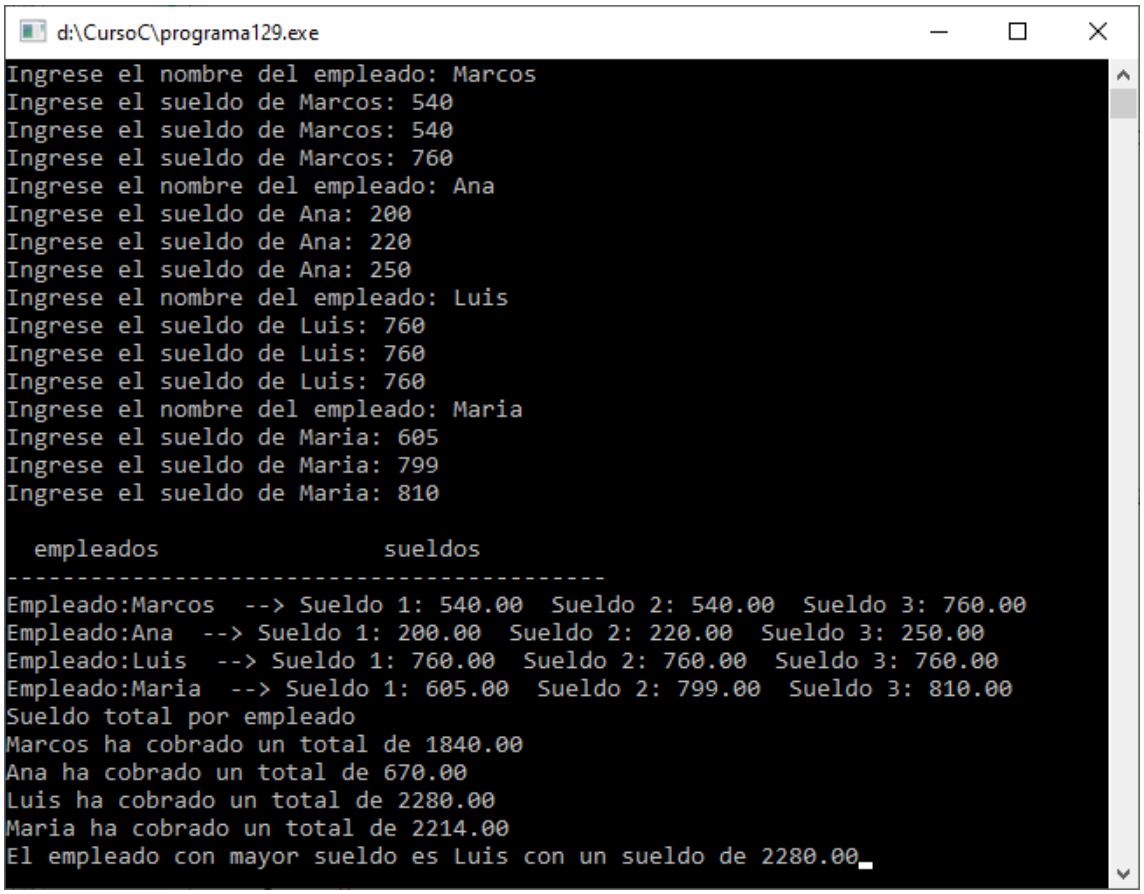
empleados	sueldos			sueldostot
Marcos	540	540	760	
Ana	200	220	250	
Luis	760	760	760	
Maria	605	799	810	

```
#include<stdio.h>
#include<conio.h>

#define COLUMNAS 3
#define FILAS 4
#define CARACTERES 21

void cargar(char emp[FILAS][CARACTERES], float suel[FILAS][COLUMNAS])
{
    for (int x=0; x<FILAS; x++)
    {
        printf("Ingrese el nombre del empleado: ");
        gets(emp[x]);
        for (int y=0; y<COLUMNAS; y++){
            printf("Ingrese el sueldo de %s: ", emp[x]);
            scanf("%f", &suel[x][y]);
        }
        fflush(stdin);
    }
}
```

```
21
22  void imprimir(char emp[FILAS][CARACTERES], float suel[FILAS][COLUMNAS])
23  {
24      printf("\n");
25      printf("  empleados                sueldos\n");
26      printf("-------------------------------------------\n");
27      for(int x=0; x<FILAS; x++)
28      {
29          printf("Empleado:%s  --> Sueldo 1: %.2f  Sueldo 2: %.2f  Sueldo 3: %.2f \n",
30                 emp[x], suel[x][0], suel[x][1], suel[x][2]);
31      }
32  }
33
34  void cargarTotal(float suel[FILAS][COLUMNAS],float sueltot[FILAS])
35  {
36      for (int x=0; x<FILAS; x++)
37      {
38          sueltot[x]=suel[x][0]+suel[x][1]+suel[x][2];
39      }
40  }
41
42  void imprimirTotalSueldos(char emp[FILAS][CARACTERES],float sueltot[FILAS])
43  {
44      printf("Sueldo total por empleado\n");
45      for (int x=0; x<FILAS; x++)
46      {
47          printf("%s ha cobrado un total de %.2f\n", emp[x], sueltot[x]);
48      }
49  }
50
```

```
51  void sueldoMayorPagado(char emp[FILAS][CARACTERES],float sueltot[FILAS])
52  {
53      float max=sueltot[0];
54      int posicion=0;
55      for (int x=1; x<FILAS; x++)
56      {
57          if (sueltot[x]>max)
58          {
59              max=sueltot[x];
60              posicion=x;
61          }
62      }
63      printf("El empleado con mayor sueldo es %s con un sueldo de %.2f", emp[posicion], max);
64  }
65
66  int main()
67  {
68      char empleados[FILAS][CARACTERES];
69      float sueldos[FILAS][COLUMNAS];
70      float sueldosTot[FILAS];
71      cargar(empleados, sueldos);
72      imprimir(empleados, sueldos);
73      cargarTotal(sueldos, sueldosTot);
74      imprimirTotalSueldos(empleados, sueldosTot);
75      sueldoMayorPagado(empleados, sueldosTot);
76      getch();
77      return 0;
78  }
79
```

Si ejecutamos este será el resultado:


d:\CursoC\programa129.exe
Ingrese el nombre del empleado: Marcos
Ingrese el sueldo de Marcos: 540
Ingrese el sueldo de Marcos: 540
Ingrese el sueldo de Marcos: 760
Ingrese el nombre del empleado: Ana
Ingrese el sueldo de Ana: 200
Ingrese el sueldo de Ana: 220
Ingrese el sueldo de Ana: 250
Ingrese el nombre del empleado: Luis
Ingrese el sueldo de Luis: 760
Ingrese el sueldo de Luis: 760
Ingrese el sueldo de Luis: 760
Ingrese el nombre del empleado: Maria
Ingrese el sueldo de Maria: 605
Ingrese el sueldo de Maria: 799
Ingrese el sueldo de Maria: 810
empleados sueldos
Empleado:Marcos --> Sueldo 1: 540.00 Sueldo 2: 540.00 Sueldo 3: 760.00
Empleado:Ana --> Sueldo 1: 200.00 Sueldo 2: 220.00 Sueldo 3: 250.00
Empleado:Luis --> Sueldo 1: 760.00 Sueldo 2: 760.00 Sueldo 3: 760.00
Empleado:Maria --> Sueldo 1: 605.00 Sueldo 2: 799.00 Sueldo 3: 810.00
Sueldo total por empleado
Marcos ha cobrado un total de 1840.00
Ana ha cobrado un total de 670.00
Luis ha cobrado un total de 2280.00
Maria ha cobrado un total de 2214.00
El empleado con mayor sueldo es Luis con un sueldo de 2280.00

Capítulo 132.- Estructura de datos tipo registro struct – 1

Un registro es una estructura de datos que permite almacenar un conjunto de elementos no necesariamente del mismo tipo.

Vimos que vectores y matrices son estructura de datos que permiten almacenar un conjunto de datos del mismo tipo.

Un registro normalmente almacena un conjunto de datos que están relacionados entre si.

Ejemplo de registros podrían ser los datos de un alumno (nº de expediente, apellido, nombre, carrera que cursa), una historia clínica de un paciente (nº de documento, obra social que tiene, enfermedades), etc.

A un vector o matriz accedemos a sus elementos por medio de subíndices, a los elementos de un registro se los llama campos y tienen cada uno un nombre.

Por ejemplo si definimos el registro "producto" sus campos pueden ser el "codigo", "descripcion" y "precio".

A diferencia de vectores y matrices los registros deben declarase y luego definir variables de dicho tipo.

Veamos la sintaxis para declarar un registro en lenguaje C:

```
struct producto {
    int codigo;
    char descripcion[41];
    float precio;
}; //obligatorio el punto y coma
```

Hemos declarado un registro llamado "producto" con tres campos uno de tipo int llamado codigo, otro de tipo cadena de caracteres llamado descripcion y finalmente el campo "precio" de tipo float.

Como vemos los tres campos están relacionados y hacen referencia a las características que puede tener todo producto que vende una empresa.

Pero si solo declaramos el registro no nos sirve de nada, debemos definir una o más variables de dicho tipo.

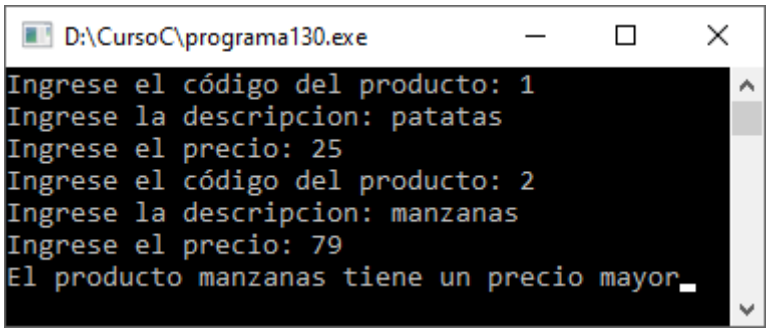
Problema

Declarar un registro que permita almacenar el código, descripcion y precio de un producto. Luego definir dos variables de dicho tipo, cargarlas e imprimir el nombre del producto que tiene mayor precio.

```
1  #include<stdio.h>
2  #include<conio.h>
3  #include<locale.h>
4
5  struct producto {
6      int codigo;
7      char descripcion[41];
8      float precio;
9  };
```

```c

int main()
{
    setlocale(LC_ALL, "");
    struct producto pro1, pro2;
    printf("Ingrese el código del producto: ");
    scanf("%i", &pro1.codigo);
    fflush(stdin);
    printf("Ingrese la descripcion: ");
    gets(pro1.descripcion);
    printf("Ingrese el precio: ");
    scanf("%f", &pro1.precio);

    printf("Ingrese el código del producto: ");
    scanf("%i", &pro2.codigo);
    fflush(stdin);
    printf("Ingrese la descripcion: ");
    gets(pro2.descripcion);
    printf("Ingrese el precio: ");
    scanf("%f", &pro2.precio);

    if (pro1.precio>pro2.precio)
    {
        printf("El producto %s tiene un precio mayor", pro1.descripcion);
    }
    else
    {
           if (pro2.precio>pro1.precio)
            {
                printf("El producto %s tiene un precio mayor", pro2.descripcion);
            }
            else
            {
                printf("Los dos productos tienen el mismo precio");
            }
    }
    getch();
    return 0;
}

```

Si ejecutamos este será el resultado:



Capítulo 133.- Estructura de datos tipo registro struct – 2

Problema propuesto

Se tiene la siguiente declaración de registro:

```
struct pais {
    char nombre[40];
    int cantidadhab;
};
```

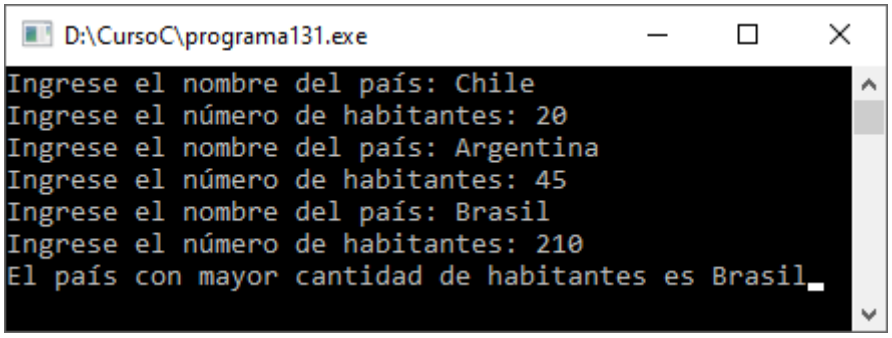
Definir tres variables de tipo país y almacenar los nombres de los países y la cantidad de habitantes de dichos países.

Mostrar seguidamente el nombre del país con mayor cantidad de habitantes (considerar que los tres países tienen cantidades distintas).

```
#include<stdio.h>
#include<conio.h>
#include<locale.h>

struct pais {
    char nombre[40];
    int cantidadhab;
};

int main()
{
    setlocale(LC_ALL, "");
    struct pais pais1, pais2, pais3;

    printf("Ingrese el nombre del país: ");
    gets(pais1.nombre);
    printf("Ingrese el número de habitantes: ");
    scanf("%i", &pais1.cantidadhab);
    fflush(stdin);

    printf("Ingrese el nombre del país: ");
    gets(pais1.nombre);
    printf("Ingrese el número de habitantes: ");
    scanf("%i", &pais2.cantidadhab);
    fflush(stdin);

    printf("Ingrese el nombre del país: ");
    gets(pais3.nombre);
    printf("Ingrese el número de habitantes: ");
    scanf("%i", &pais3.cantidadhab);

    if(pais1.cantidadhab>pais2.cantidadhab && pais1.cantidadhab>pais3.cantidadhab)
    {
        printf("El país con mayor cantidad de habitantes es %s", pais1.nombre);
    }
   else
    {
        if (pais2.cantidadhab>pais3.cantidadhab)
        {
            printf("El país con mayor cantidad de habitantes es %s", pais2.nombre);
        }
        else
        {
           printf("El país con mayor cantidad de habitantes es %s", pais3.nombre);
        }
    }

```

```
48      getch();
49      return 0;
50  }
51
```

Si ejecutamos este será el resultado:



Capítulo 134.- Funciones con parámetros de tipo struct – 1

Hemos visto anteriormente que una función puede recibir tipos de datos simples como int, char, y float y lo que sucede es que se hace una copia en el parámetro.

También hemos visto y trabajado con parámetros de tipo vector y matriz y funcionamiento es muy distinto a los tipos de datos simples. Si en la función modificamos el parámetro lo que sucede es que se modifica la variable que le pasamos desde la función main.

Ahora veremos como trata el lenguaje C los parámetros de tipo struct. Su funcionamiento es idéntico a los tipos de datos simples, es decir que se hace una copia del registro en el parámetro. No podemos modificar el registro que le enviamos a la función sino solo accederlo para consultarlo.

Problema

Se tiene la siguiente declaración de registro:

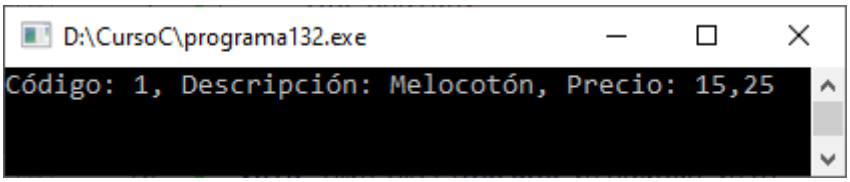
```
struct producto {
    int codigo;
    char descripcion[41];
    float precio;
}; //obligatorio el punto y coma
```

Definir una variable en la función main e inicializar por asignación los tres campos.
Plantear una función que reciba el registro y lo imprima.

```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<string.h>
4   #include<locale.h>
5
6   struct producto {
7       int codigo;
8       char descripcion[41];
9       float precio;
10  };
11
12  void imprimir(struct producto pro)
13  {
14      printf("Código: %i, Descripción: %s, Precio: %0.2f",
15             pro.codigo, pro.descripcion, pro.precio);
16  }
17
18  int main()
19  {
20      setlocale(LC_ALL, "");
21      struct producto pro1;
22      pro1.codigo=1;
23      strcpy(pro1.descripcion, "Melocotón");
24      pro1.precio=15.25;
25      imprimir(pro1);
```

```
26      getch();
27      return 0;
28  }
29
```

Si ejecutamos este será el resultado:



Capítulo 135.- Funciones con parámetros de tipo struct – 2

Problema propuesto

Se tiene la siguiente declaración de registro:

```
struct pais {
    char nombre[40];
    int cantidadhab;
};
```

Define tres variables de tipo país e inicializarlas con asignación con la sintaxis:

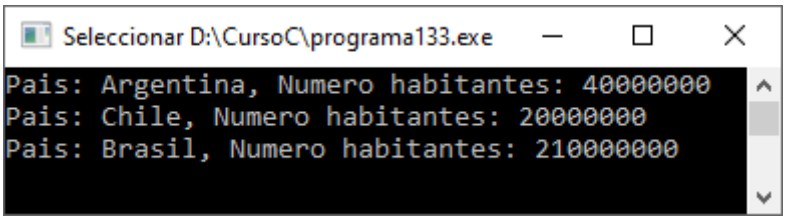
```
struct pais pais1={"Argentina",40000000};
```

Elabora una función que reciba un parámetro de tipo país y muestre por pantalla sus dos campos.

Llamar a dicha función desde la main pasando en forma sucesiva las tres variables definidas.

```
#include<stdio.h>
#include<conio.h>

struct pais{
    char nombre[40];
    int cantidadhab;
};

void imprimir(struct pais pa)
{
    printf("Pais: %s, Numero habitantes: %i\n", pa.nombre, pa.cantidadhab);
}

int main()
{
    struct pais pais1={"Argentina", 40000000};
    struct pais pais2={"Chile", 20000000};
    struct pais pais3={"Brasil", 210000000};

    imprimir(pais1);
    imprimir(pais2);
    imprimir(pais3);

    getch();
    return 0;
}

```

Si ejecutamos este será el resultado:



Capítulo 136.- Funciones con retorno de un struct – 1

Hemos visto que una función cuando no retorna dato definimos la palabra clave void previo al nombre de la función:

```
void cargar()
```

Cuando devuelve una función un tipo de dato simple puede ser int, float o char:

```
int cargar()
```

En este concepto veremos que una función puede retornar un registro:

```
struct producto cargar()
```

Problema

Se tiene la siguiente declaración de registro:

```
struct producto {
    int codigo;
    char descripcion[41];
    float precio;
}; //obligatorio el punto y coma
```

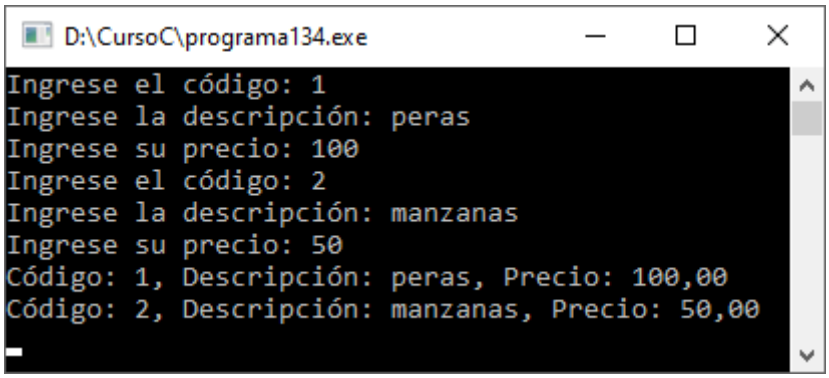
Plantear dos funciones una que cargue un registro de tipo producto y lo retorne, y otra que lo imprima.

En la función main definir dos variables de tipo producto llamar a las funciones anteriores.

```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<locale.h>
4
5   struct producto {
6       int codigo;
7       char descripcion[41];
8       float precio;
9   };
10
11  struct producto cargar()
12  {
13      struct producto pro;
14      printf("Ingrese el código: ");
15      scanf("%i", &pro.codigo);
16      fflush(stdin);
17      printf("Ingrese la descripción: ");
18      gets(pro.descripcion);
19      fflush(stdin);
20      printf("Ingrese su precio: ");
21      scanf("%f", &pro.precio);
22      fflush(stdin);
23      return pro;
24  }
25
```

```c
void imprimir(struct producto pro)
{
    printf("Código: %i, Descripción: %s, Precio: %.2f\n",
           pro.codigo, pro.descripcion, pro.precio);
}

int main()
{
    setlocale(LC_ALL, "");
    struct producto pro1, pro2;
    pro1=cargar();
    pro2=cargar();
    imprimir(pro1);
    imprimir(pro2);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 137.- Funciones con retorno de un struct – 2

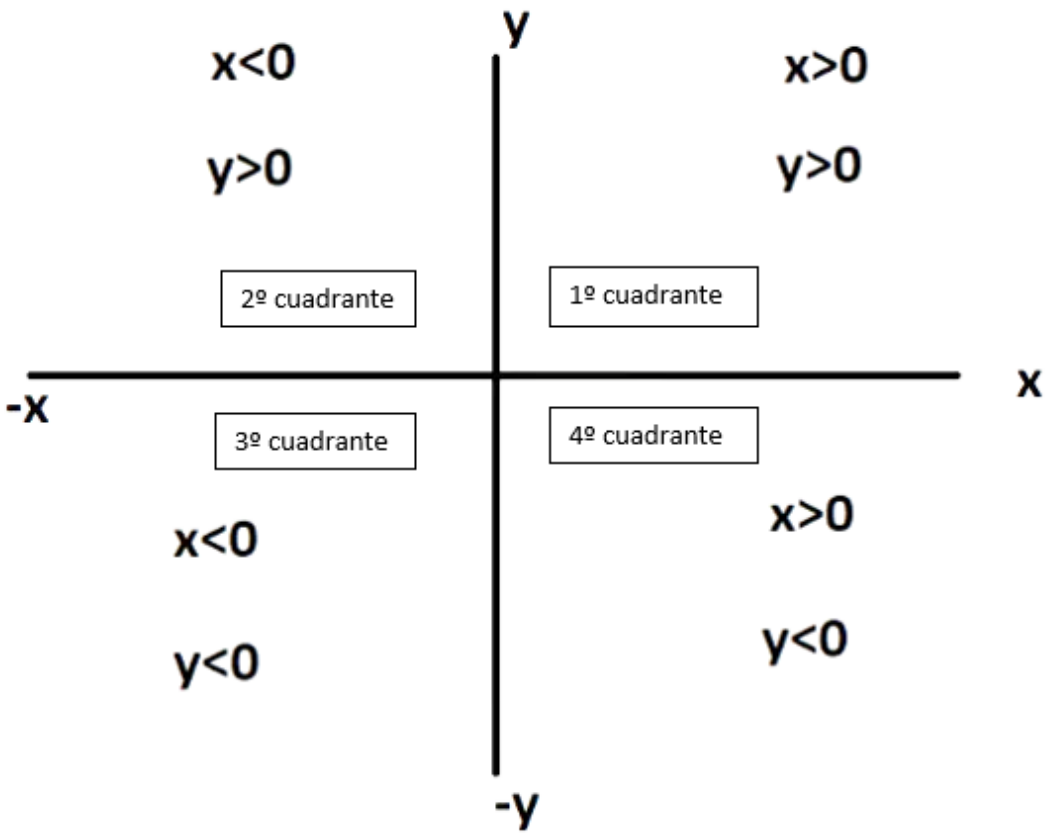
Problema propuesto

Se tiene la siguiente declaración de registro:

```
struct punto {
    int x;
    int y;
};
```

Definir 3 variables de tipo punto y cargarlas llamando a una función que retorne valores de tipo punto.

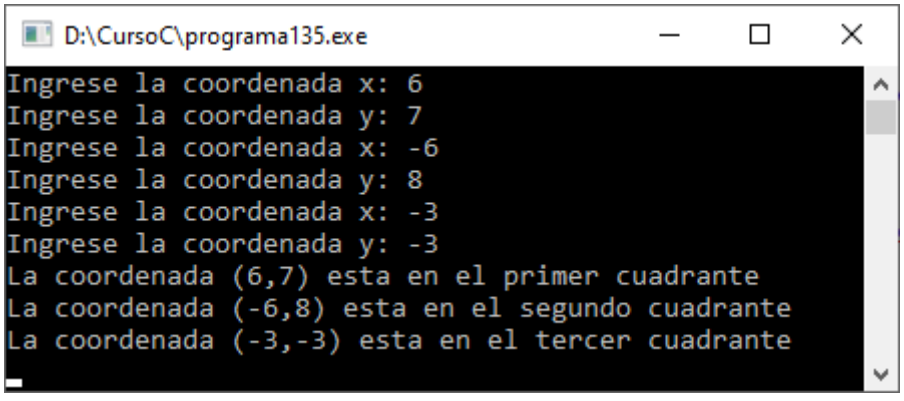
Finalmente crear otra función que imprima en que cuadrante se encuentra cada punto (tener en cuenta que si x>0 e y>0 se encuentra en el primer cuadrante, si x<0 e y>0 se encuentra en el segundo cuadrante y así sucesivamente).




```
1  #include<stdio.h>
2  #include<conio.h>
3  #include<locale.h>
4
5  struct punto{
6      int x;
7      int y;
8  };
9
```

```c
struct punto cargar()
{
    struct punto pun;
    printf("Ingrese la coordenada x: ");
    scanf("%i", &pun.x);
    printf("Ingrese la coordenada y: ");
    scanf("%i", &pun.y);
    return pun;
}

void cuadrante(struct punto pun)
{
    if (pun.x>0 && pun.y>0)
    {
        printf("La coordenada (%i,%i) esta en el primer cuadrante\n", pun.x, pun.y);
    }
    else
    {
        if(pun.x<0 && pun.y>0)
        {
            printf("La coordenada (%i,%i) esta en el segundo cuadrante\n", pun.x, pun.y);
        }
        else
        {
            if(pun.x<0 && pun.y<0)
            {
                printf("La coordenada (%i,%i) esta en el tercer cuadrante\n", pun.x, pun.y);
            }
            else
            {
                if(pun.x>0 && pun.y<0)
                {
                    printf("La coordenada (%i,%i) esta en el cuarto cuadrante\n", pun.x, pun.y);
                }
                else
                {
                    printf("La coordenada (%i,%i) no se encuentra en ningún cuadrante\n", pun.x, pun.y);
                }
            }
        }
    }
}

int main()
{
    setlocale(LC_ALL, "");
    struct punto punto1, punto2, punto3;
    punto1=cargar();
    punto2=cargar();
    punto3=cargar();
    cuadrante(punto1);
    cuadrante(punto2);
    cuadrante(punto3);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 138.- Estructura de datos tipo vector elementos de tipo struct – 1

Hemos trabajado bastante con la estructura de datos tipo vector definiendo elementos de tipo int, float y char.

Ahora veremos que podemos definir vectores cuyos elementos sean de tipo registro.

Problema

Se tiene la siguiente declaración de registro:

```
struct producto {
    int codigo;
    char descripcion[41];
    float precio;
};
```

Definir un vector de 4 elementos de tipo producto.

Implementar las funciones:

- Cargar el vector.
- Impresión del vector.
- Mostrar el nombre del articulo con precio mayor.

```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<locale.h>
4
5   struct producto{
6       int codigo;
7       char descripcion[41];
8       float precio;
9   };
10
11  void cargar(struct producto produ[4])
12  {
13      for (int x=0; x<4; x++)
14      {
15          printf("Ingrese el código del producto: ");
16          scanf("%i", &produ[x].codigo);
17          fflush(stdin);
18          printf("Ingrese la descripción del producto: ");
19          gets(produ[x].descripcion);
20          fflush(stdin);
21          printf("Ingrese el precio del producto: ");
22          scanf("%f", &produ[x].precio);
23          fflush(stdin);
24          printf("\n");
25      }
26  }
27
```

```c
void imprimir(struct producto produ[4])
{
    for (int x=0; x<4; x++)
    {
        printf("Código: %i, Descripción: %s, Precio: %.2f\n",
               produ[x].codigo, produ[x].descripcion, produ[x].precio);
    }
    printf("\n");
}
void precioMayor(struct producto produ[4])
{
    float max=produ[0].precio;
    int pos=0;
    for (int x=1; x<4; x++)
    {
        if(produ[x].precio>max)
        {
            max=produ[x].precio;
            pos=x;
        }
    }
    printf("El producto con un precio mayor es:\n %s con un precio de %.2f\n",
           produ[pos].descripcion, produ[pos].precio);
}

int main()
{
    setlocale(LC_ALL,"");
    struct producto pro[4];
    cargar(pro);
    imprimir(pro);
    precioMayor(pro);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:

```
D:\CursoC\programa136.exe
Ingrese el código del producto: 1
Ingrese la descripción del producto: patatas
Ingrese el precio del producto: 34

Ingrese el código del producto: 2
Ingrese la descripción del producto: manzanas
Ingrese el precio del producto: 79

Ingrese el código del producto: 3
Ingrese la descripción del producto: bananas
Ingrese el precio del producto: 56

Ingrese el código del producto: 4
Ingrese la descripción del producto: kiwis
Ingrese el precio del producto: 120

Código: 1, Descripción: patatas, Precio: 34,00
Código: 2, Descripción: manzanas, Precio: 79,00
Código: 3, Descripción: bananas, Precio: 56,00
Código: 4, Descripción: kiwis, Precio: 120,00

El producto con un precio mayor es:
 kiwis con un precio de 120,00
```

Capítulo 139.- Estructura de datos tipo vector elementos de tipo struct – 2

Problema propuesto

Se tiene la siguiente declaración de registro:

```
struct libro{
    int codigo;
    char titulo[40];
    char autor[40];
};
```

Definir un vector de 4 elementos de tipo libro.

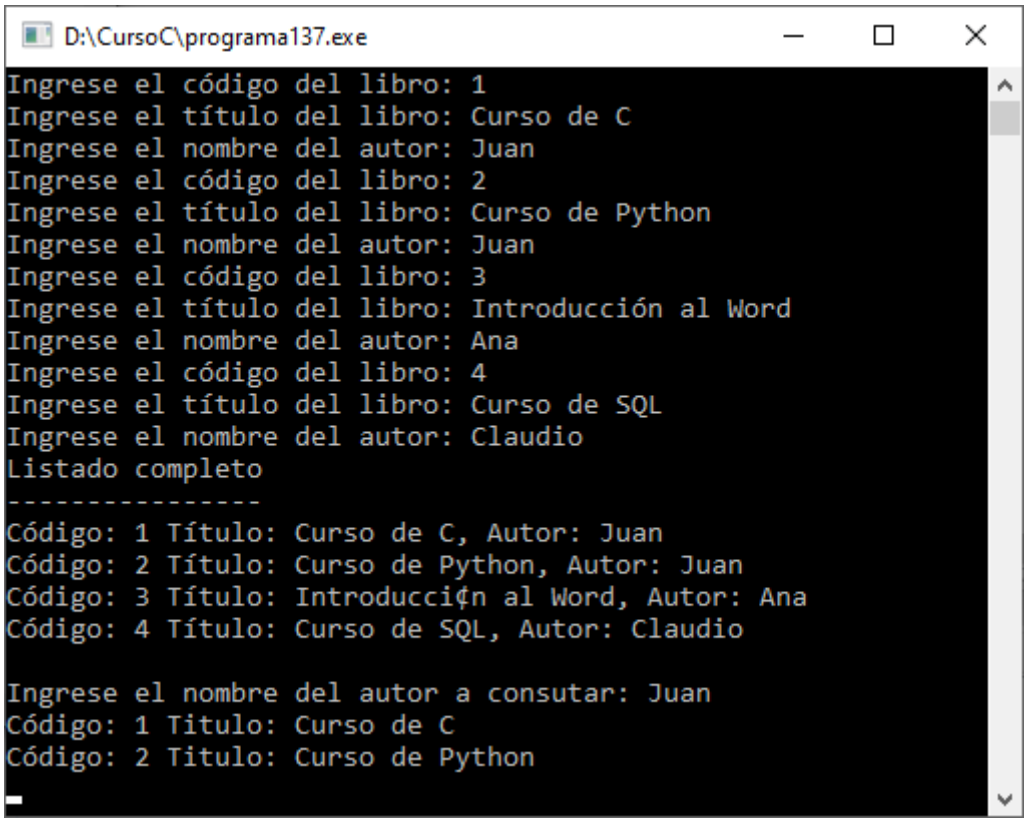
Codificar las funciones:

1- Cargar el vector.
2- Listado completo.
3- Ingresar por teclado el nombre del autor y mostrar todos los títulos que ha escrito o un mensaje si no tiene.

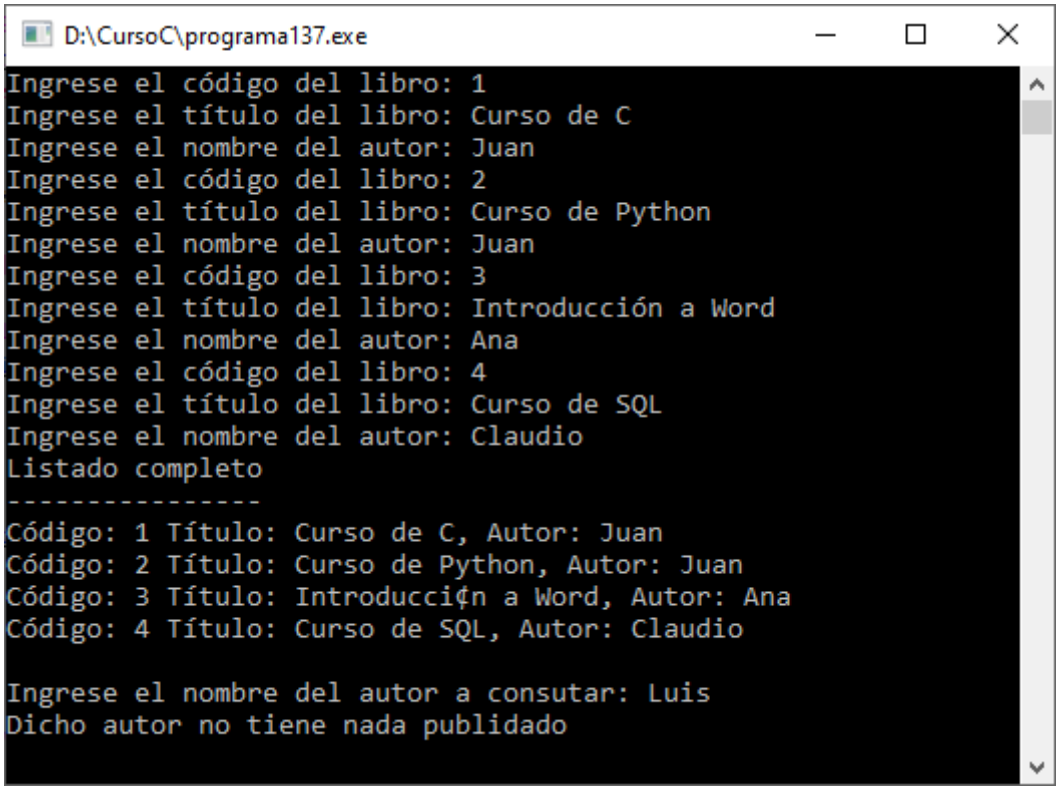
```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<locale.h>

struct libro{
    int codigo;
    char titulo[40];
    char autor[40];
};

void cargar(struct libro lib[4])
{
    for (int x=0; x<4; x++)
    {
        printf("Ingrese el código del libro: ");
        scanf("%i", &lib[x].codigo);
        fflush(stdin);
        printf("Ingrese el título del libro: ");
        gets(lib[x].titulo);
        printf("Ingrese el nombre del autor: ");
        gets(lib[x].autor);
        fflush(stdin);
    }
}
```

```
26
27  void imprimir(struct libro lib[4])
28  {
29      printf("Listado completo\n");
30      printf("----------------\n");
31      for (int x=0; x<4; x++)
32      {
33          printf("Código: %i Título: %s, Autor: %s\n",
34                 lib[x].codigo, lib[x].titulo, lib[x].autor);
35      }
36      printf("\n");
37  }
38
39  void consultarAutor(struct libro lib[4])
40  {
41      char consultaAutor[40];
42      printf("Ingrese el nombre del autor a consutar: ");
43      gets(consultaAutor);
44      fflush(stdin);
45      int encontrado=0;
46      for (int x=0; x<4; x++)
47      {
48          if(strcmp(lib[x].autor, consultaAutor)==0)
49          {
50              encontrado=1;
51              printf("Código: %i Titulo: %s\n", lib[x].codigo, lib[x].titulo);
52          }
53      }
54      if(encontrado==0)
55      {
56          printf("Dicho autor no tiene nada publidado\n");
57      }
58  }
59
60  int main()
61  {
62      setlocale(LC_ALL, "");
63      struct libro li[4];
64      cargar(li);
65      imprimir(li);
66      consultarAutor(li);
67      getch();
68      return 0;
69  }
70
```

Si ejecutamos este será el resultado:



Vamos a ejecutar y consultaremos por un autor que no está.



Capítulo 140.- Estructura de datos tipo registro (con campos int, float, vector, registros anidados, etc.) – 1

Hemos visto ya varias estructuras de datos fundamentales en lenguaje C:

```
vectores

matrices

registros
```

También hemos visto que podemos definir vectores y matrices cuyos elementos sean de tipo registro.

En este concepto veremos que los campos de un registro pueden ser también registro, es decir anidar un registro dentro de otro registro.

Problema

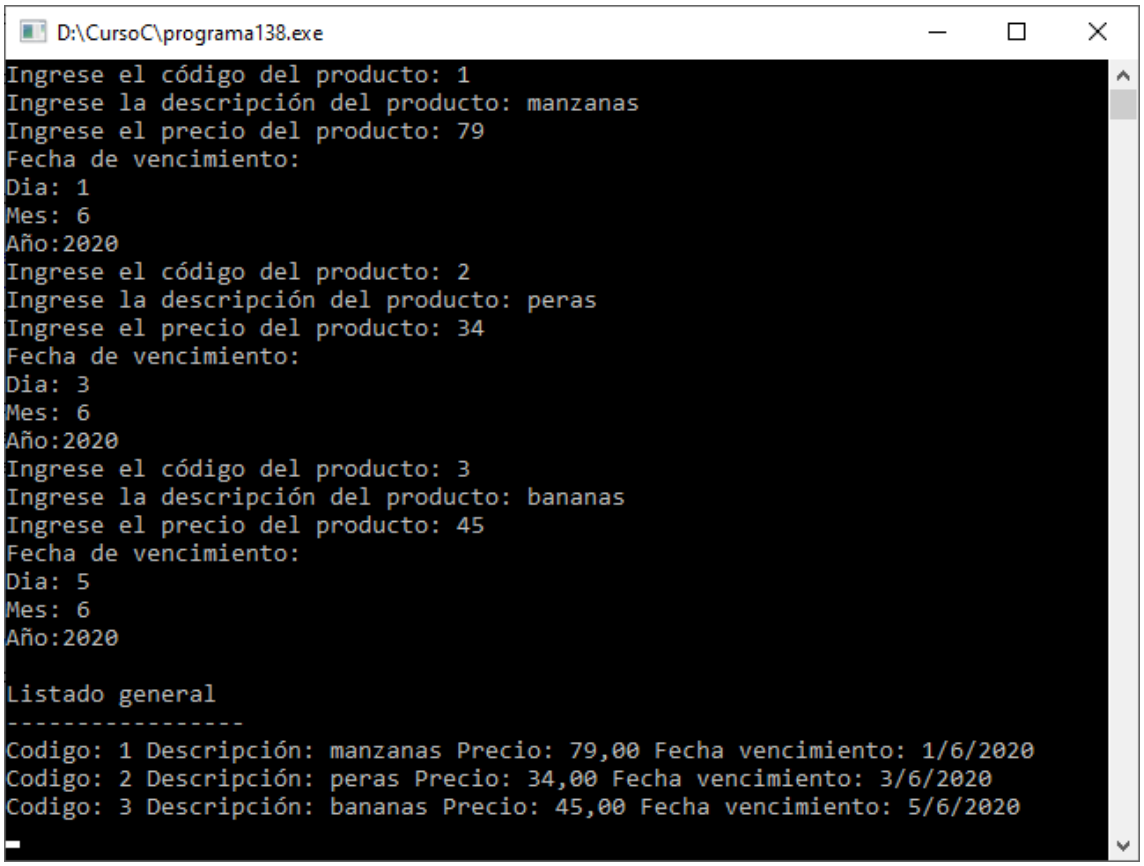
Se tiene las siguientes declaraciones de registros:

```
struct fecha {
    int dd;
    int mm;
    int aa;
};

struct producto {
    int codigo;
    char descripcion[41];
    float precio;
    struct fecha fechavencimiento;
};
```

Definir un vector de 3 elementos de tipo producto, realizar su carga e impresión.

```c
#include<stdio.h>
#include<conio.h>
#include<locale.h>

#define LONG 3

struct fecha{
    int dd;
    int mm;
    int aa;
};

struct producto{
    int codigo;
    char descripcion[41];
    float precio;
    struct fecha fechavencimiento;
};

void cargar(struct producto pro[LONG])
{
    for (int x=0; x<LONG; x++)
    {
        printf("Ingrese el código del producto: ");
        scanf("%i", &pro[x].codigo);
        fflush(stdin);
        printf("Ingrese la descripción del producto: ");
        gets(pro[x].descripcion);
        fflush(stdin);
        printf("Ingrese el precio del producto: ");
        scanf("%f", &pro[x].precio);
        fflush(stdin);
        printf("Fecha de vencimiento:\n");
        printf("Dia: ");
        scanf("%i", &pro[x].fechavencimiento.dd);
        printf("Mes: ");
        scanf("%i", &pro[x].fechavencimiento.mm);
        printf("Año:");
        scanf("%i", &pro[x].fechavencimiento.aa);
    }
}

void imprimir(struct producto pro[LONG])
{
    printf("\nListado general\n");
    printf("-----------------\n");
    for (int x=0; x<LONG; x++)
    {
        printf("Codigo: %i Descripción: %s Precio: %.2f Fecha vencimiento: %i/%i/%i\n",
               pro[x].codigo, pro[x].descripcion, pro[x].precio,
               pro[x].fechavencimiento.dd, pro[x].fechavencimiento.mm,
               pro[x].fechavencimiento.aa);
    }
}

int main()
{
    setlocale(LC_ALL, "");
    struct producto produc[LONG];
    cargar(produc);
    imprimir(produc);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 141.- Estructura de datos tipo registro (con campos int, float, vector, registros anidados, etc.) – 2

Problema propuesto

Se tienen las siguientes declaraciones de registros:

```
struct punto {
    int x;
    int y;
};

struct triangulo {
    struct punto punto1;
    struct punto punto2;
    struct punto punto3;
};
```

Definir en la main un registro de tipo triangulo.

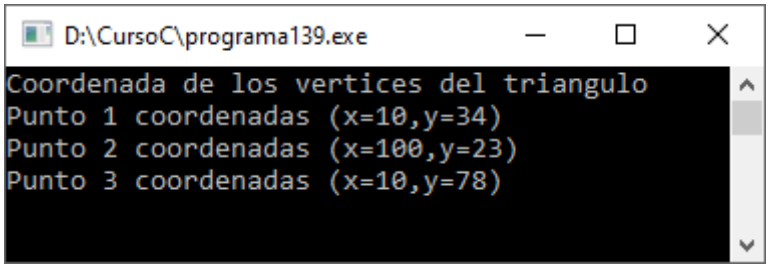
Codificar las fundiones:

1- Una función que retorne un registro de tipo triangulo.
2- Impresión del registro.

```
#include<stdio.h>
#include<conio.h>

struct punto {
    int x;
    int y;
};

struct triangulo {
    struct punto punto1;
    struct punto punto2;
    struct punto punto3;
};

struct triangulo cargar()
{
    struct triangulo tri;
    tri.punto1.x=10;
    tri.punto1.y=34;
    tri.punto2.x=100;
    tri.punto2.y=23;
    tri.punto3.x=10;
    tri.punto3.y=78;
    return tri;
};

void imprimir (struct triangulo t)
{
    printf("Coordenada de los vertices del triangulo\n");
    printf("Punto 1 coordenadas (x=%i,y=%i)\n", t.punto1.x, t.punto1.y);
    printf("Punto 2 coordenadas (x=%i,y=%i)\n", t.punto2.x, t.punto2.y);
    printf("Punto 3 coordenadas (x=%i,y=%i)\n", t.punto3.x, t.punto3.y);
}
```

```
34
35  int main()
36  {
37      struct triangulo t;
38      t=cargar();
39      imprimir(t);
40      getch();
41      return 0;
42  }
43
```

Si ejecutamos este será el resultado:



Capítulo 142.- Variables de tipo puntero – 1

Un tema fundamental del lenguaje C es el manejo del concepto de punteros.

Los punteros en el lenguaje C son esenciales:

- Para que un programa sea más eficiente.
- Modificar variables de tipo int, float, struct, etc. en otras funciones.
- Poder requerir y liberar memoria durante la ejecución de programas (hay muchas situaciones donde no sabemos cuánto espacio reservar).

Definición de puntero

Una variable de tipo puntero almacena la dirección de memoria de otra variable que puede ser int, char, float, struct, etc.

Es difícil en un principio entender y ver las ventajas que trae trabajar con punteros pero su uso es fundamental si vamos a trabajar con el lenguaje C. Recordemos que el lenguaje C se utiliza fundamentalmente para el desarrollo de software de base (sistemas operativos, drives,etc.).

Por ejemplo si seleccionamos una función al azar del código fuente del sistema operativo Linux veremos que siempre utiliza punteros:

```
struct sk_buff *audit_make_reply(__u32 portid, int seq, int type,
                                 int multi, const void *payload,
{
        struct sk_buff *skb;
        struct nlmsghdr *nlh;
        void           *data;
        int             flags = multi ? NLM_F_MULTI : 0;
        int             t     = done  ? NLMSG_DONE  : type;

        skb = nlmsg_new(size, GFP_KERNEL);
        if (!skb)
                return NULL;

        nlh     = nlmsg_put(skb, portid, seq, t, size, flags);
        if (!nlh)
                goto out_kfree_skb;
        data = nlmsg_data(nlh);
        memcpy(data, payload, size);
        return skb;

 out_kfree_skb:
        kfree_skb(skb);
        return NULL;
}
```

Los punteros son aquellas variables antecedidas por el carácter *.

Definición de punteros a tipos de datos int y float

La sintaxis para definir una variable de tipo puntero se logra antecediendo el carácter * al nombre de la variable:

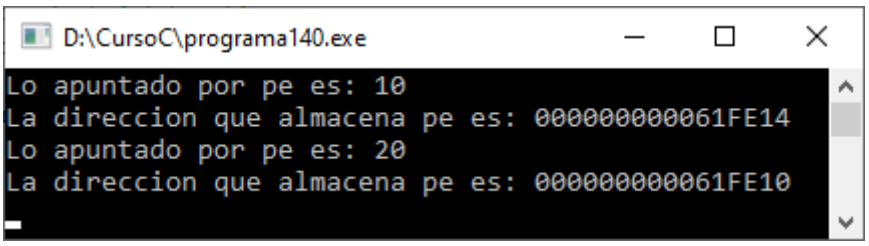
```
int *pe;
float *pf;
```

Problema

Definir dos variables enteras y almacenar valores por asignación. Definir una variable puntero a entero y guardar sucesivamente las direcciones de dichas dos variables y acceder a sus valores.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   int main()
5   {
6       int valor1=10;
7       int valor2=20;
8       int *pe;
9       pe=&valor1;
10      printf("Lo apuntado por pe es: %i\n", *pe);
11      printf("La direccion que almacena pe es: %p\n", pe);
12      pe=&valor2;
13      printf("Lo apuntado por pe es: %i\n", *pe);
14      printf("La direccion que almacena pe es: %p\n", pe);
15      getch();
16      return 0;
17  }
18
```

La ejecución de este programa tiene una salida similar a:



Haremos una serie de problemas para entender el concepto de una variable de tipo puntero que en principio no le encontramos mayores ventajas a como veníamos resolviendo problemas. Lo más conveniente es empezar con problemas sencillos para luego ver la verdadera potencia que nos proporcionan las variables de tipo puntero.

En este problema definimos dos variables de tipo entera y las inicializamos con los valores 10 y 20:

```
int valor1=10;
int valor2=20;
```

Nosotros ya hemos visto que si queremos acceder e imprimir el contenido de la variable "valor1" lo hacemos por medio de su nombre:

```
printf("%i",valor1);
```

Hay otra forma de acceder al contenido de la variable "valor1" mediante el concepto de una variable de tipo puntero. Definimos una tercera variable llamada pe (una variable puntero a entero):

```
int *pe;
```

La variable "pe" puede almacenar una dirección de memoria, es incorrecto (no podemos asignar valor1 a pe):

```
pe=valor1;
```

pe y valor1 son variables de distinto tipo, valor1 es una variable entera y pe es una variable que apunta a una dirección de memoria donde se almacena un entero.

Para almacenar en la variable pe la dirección de memoria donde se almacena valor1 utilizamos el carácter &:

```
pe=&valor1;
```

La línea anterior la podemos leer como: "en la variable pe almacenamos la dirección de memoria donde se almacena valor1".

Si vemos el contenido del la memoria podemos identificar dos variables "valor1" y "pe":

Dirección	Memoria	Variable
1000	10	valor1
1004	1000	pe

La variable "valor1" almacena el número entero 10. La variable "valor1" se almacena por ejemplo en la dirección de memoria 1000 (este valor es a modo de ejemplo y la dirección real sucede cuando se ejecuta el programa).

La variable "pe" puede almacenar la dirección de una variable entera, cuando hacemos la asignación:

```
pe=&valor1;
```

Estamos guardando la dirección 1000 en la variable "pe".

Para imprimir lo apuntado por el puntero pe utilizamos la sintaxis *pe, en nuestro caso accedemos al valor 10 almacenado en valor1:

```
printf("Lo apuntado por pe es:%i\n",*pe);
```

Imprimir el contenido de pe puede tener poco sentido ya que veremos la dirección que almacena pe, en el caso que lo hagamos debemos indicar %p:

```
printf("La direccion que almacena pe es:%p\n",pe);
```

Una variable de tipo puntero puede cambiar la dirección que almacena a lo largo de la ejecución del programa, luego si hacemos:

```
pe=&valor2;
```

Estamos almacenando la dirección de la variable valor2. Si imprimimos lo apuntado por pe tendremos el número 20:

```
printf("Lo apuntado por pe es:%i\n",*pe);
```

Capítulo 143.- Variables de tipo puntero – 2

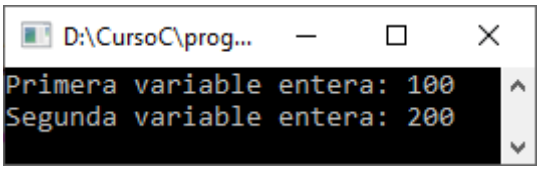
Definir dos variables enteras y no inicializarlas.

Definir una variable puntero a entero, hacer que apunte sucesivamente a las dos variables enteras definidas previamente y cague sus contenidos.

Imprimir las dos variables enteras.

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int x1, x2;
    int *pe;
    pe=&x1;
    *pe=100;
    pe=&x2;
    *pe=200;
    printf("Primera variable entera: %i \n", x1);
    printf("Segunda variable entera: %i \n", x2);
    getch();
    return 0;
}

```

Si ejecutamos este será el resultado:



Si no utilizamos una variable de tipo puntero este sería el ejemplo:

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int x1=100, x2=200;
    printf("Primera variable entera: %i \n", x1);
    printf("Segunda variable entera: %i \n", x2);
    getch();
    return 0;
}
```

Cuando ejecutemos el resultado será el mismo.

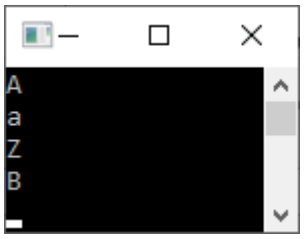
Capítulo 144.- Variables de tipo puntero – 3

Problema propuesto

Se tiene el siguiente programa:

```
#include<stdio.h>
#include<conio.h>

int main()
{
    char c1='A';
    char c2='B';
    char *pc;
    pc=&c1;
    printf("%c\n",c1); //se imprime: ?
    *pc='a';
    printf("%c\n",c1); //se imprime: ?
    c1='Z';
    printf("%c\n",*pc); //se imprime: ?
    pc=&c2;
    printf("%c\n",*pc); //se imprime: ?
    getch();
    return 0;
}
```

1000	A	c1
1001	B	c2
1002		pc



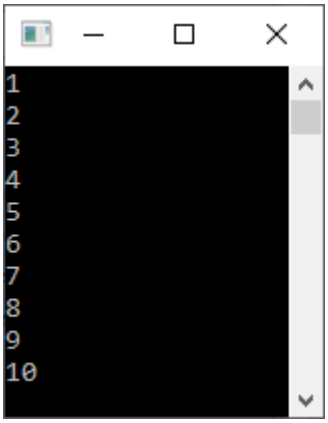
Indicar que valor se imprime en cada llamada a printf.

- Se tiene el siguiente programa:

```
#include<stdio.h>

int main()
{
    int f;
    int *pe;
    pe=&f;
    for(*pe=1;*pe<=10;*pe=*pe+1)
    {
        printf("%i\n",f); //se imprime ?
    }
    getch();
    return 0;
}
```

1000		f
1004		pe

Indicar que valor se imprime en cada llamada printf.

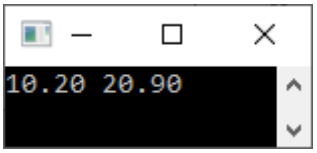


- Se tiene el siguiente programa:

```
#include <stdio.h>
#include<conio.h>

int main()
{
  float z1,z2;
  float *pf;
  pf=&z1;
  *pf=10.2;
  pf=&z2;
  *pf=20.90;
  printf("%0.2f %0.2f",z1,z2); // Se imprime ?
  getch();
  return 0;
}
```

Indicar que valor se imprime en la llamada printf.

1000		z1
1004		z2
1008		pf



Capítulo 145.- Parámetros de una función de tipo punteros a int, float y char – 1

Hemos visto que cuando llamamos a una función y le pasamos un tipo de dato char, int o float lo que sucede es que se hace una copia de dicha variable en el parámetro de la función.
Ahora veremos el primer uso real de punteros en el lenguaje C. Cuando uno quiere modificar una variable char, int o float en una función lo que hacemos es pasar la dirección de la variable y que lo reciba un puntero.

Problema

Confeccionar una función que reciba como parámetro las direcciones de dos variables enteras y le cargue a lo apuntado por dichas variables dos enteros.

Importante

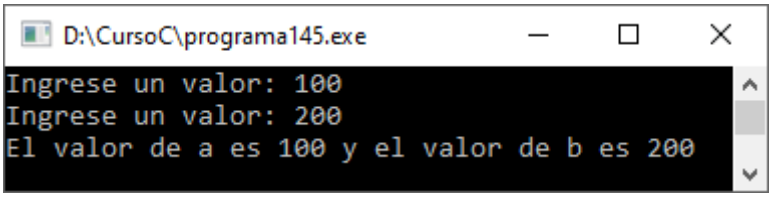
Después de mucho avanzar en este curso del lenguaje C podemos entender completamente porque es la siguiente sintaxis para cargar un entero por teclado:

```
int x1;
scanf("%i",&x1);
```

El & significa que le pasamos la dirección de la variable x1 para que la función por medio de un puntero cargue un entero en x1.

```
1  #include<stdio.h>
2  #include<conio.h>
3
4  void cargar(int *puntero)
5  {
6      int valor;
7      printf("Ingrese un valor: ");
8      scanf("%i", &valor);
9      *puntero=valor;
10 }
11
12 int main()
13 {
14     int a, b;
15     int *p;
16     p=&a;
17     cargar(p);
18     p=&b;
19     cargar(p);
20     printf("El valor de a es %i y el valor de b es %i", a, b);
21     getch();
22     return 0;
23 }
```

Si ejecutamos este será el resultado:



Otro ejemplo:

```c
#include<stdio.h>
#include<conio.h>

void cargar(int *pe1, int *pe2)
{
   *pe1=100;
   *pe2=200;
}

int main()
{
    int x1, x2;
    cargar(&x1, &x2);
    printf("%i    %i",x1,x2);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:

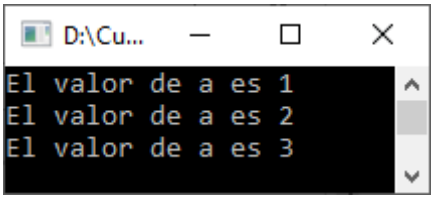
Capítulo 146.- Parámetros de una función de tipo punteros a int, float y char – 2

Problema

Elaborar una función que se le pase la dirección de una variable entera e incremente en 1 lo apuntado por dicha variable.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   void incrementar(int *puntero)
5   {
6       *puntero=*puntero+1;
7   }
8
9   int main()
10  {
11      int a=0;
12      incrementar(&a);
13      printf("El valor de a es %i\n", a);
14      incrementar(&a);
15      printf("El valor de a es %i\n", a);
16      incrementar(&a);
17      printf("El valor de a es %i\n", a);
18      getch();
19      return 0;
20  }
```

Si ejecutamos este será el resultado:

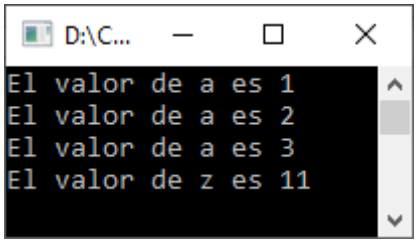


Modificamos el código:

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   void incrementar(int *puntero)
5   {
6       *puntero=*puntero+1;
7   }
8
9   int main()
10  {
11      int a=0;
12      incrementar(&a);
13      printf("El valor de a es %i\n", a);
14      incrementar(&a);
```

```
15      printf("El valor de a es %i\n", a);
16      incrementar(&a);
17      printf("El valor de a es %i\n", a);
18      int z=10;
19      incrementar(&z);
20      printf("El valor de z es %i\n", z);
21      getch();
22      return 0;
23  }
24
```

Si ejecutamos este será el resultado:



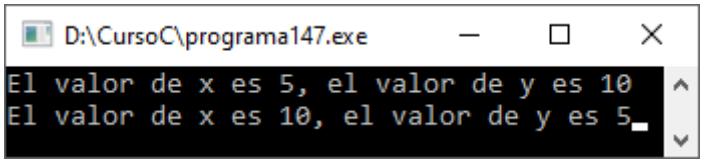
Capítulo 147.- Parámetros de una función de tipo punteros a int, float y char – 3

Problema

Implementar una función que intercambie el contenido de dos variables enteras, utilizar punteros para solucionarlo.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   void cambio(int *p1, int *p2)
5   {
6       int aux;
7       aux=*p1;
8       *p1=*p2;
9       *p2=aux;
10  }
11
12  int main()
13  {
14      int x=5, y=10;
15      printf("El valor de x es %i, el valor de y es %i\n",x,y);
16      cambio(&x, &y);
17      printf("El valor de x es %i, el valor de y es %i",x,y);
18      getch();
19      return 0;
20  }
21
```

Si ejecutamos este será el resultado:



Capítulo 148.- Parámetros de una función de tipo punteros a int, float y char – 4

Problema propuesto

Confeccionar un programa que permita cagar un vector de 5 enteros y obtenga el mayor y el menor.

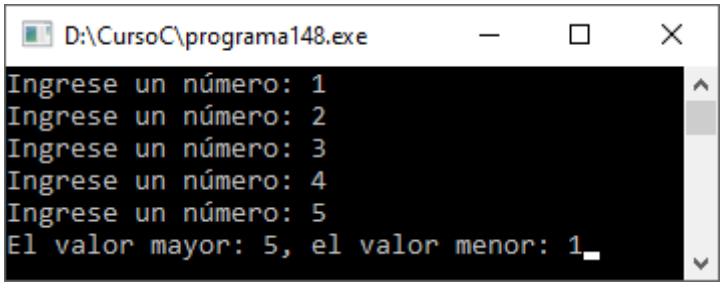
1- Carga del vector.
2- Otra función que reciba el vector y retorne el mayor y menor elemento del vector por medio de dos parámetros de tipo puntero.

void mayorMenor(int vec[TAMANO], int *pmayor, int *pmenor)

```
#include<stdio.h>
#include<conio.h>
#include<locale.h>

#define TAMANO 5

void cargarVector(int vec[TAMANO])
{
    for (int x=0; x<TAMANO; x++)
    {
        printf("Ingrese un número: ");
        scanf("%i", &vec[x]);
    }
}

void mayorMenor(int vec[TAMANO], int *pmayor, int *pmenor)
{
    *pmayor=vec[0];
    *pmenor=vec[0];
    for(int x=1; x<TAMANO; x++)
    {
        if (vec[x]>*pmayor)
        {
            *pmayor=vec[x];
        }
        else
        {
           if (vec[x]<*pmenor)
            {
                *pmenor=vec[x];
            }
        }
    }
}

```

```c
int main()
{
    setlocale(LC_ALL, "");
    int vector[TAMANO];
    int mayor, menor;
    cargarVector(vector);
    mayorMenor(vector, &mayor, &menor);
    printf( "El valor mayor: %i, el valor menor: %i", mayor, menor);
    getch();
    return 0;
}

```

Si ejecutamos este será el resultado:



Capítulo 149.- Parámetros de una función de tipo punteros a struct – 1

Vimos que cuando pasamos una variable de tipo registro a una función lo que sucede es que se hace una copia en el parámetro.

Ahora veremos que si queremos modificar el registro en la función lo que debemos hacer es pasar la dirección del registro y que lo reciba el puntero.

Problema

Se tiene la siguiente declaración de registro:

```
struct producto {
    int codigo;
    char descripcion[41];
    float precio;
};
```

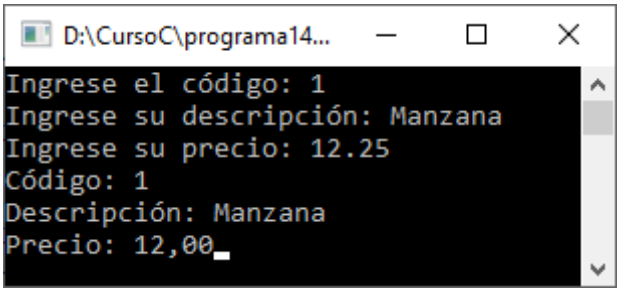
Plantear una función que reciba la dirección de un registro y mediante esta modificar los campos de la variable que le pasamos desde la main.

Imprimir el registro definido en la main.

```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<locale.h>
4
5   struct producto{
6       int codigo;
7       char descripcion[41];
8       float precio;
9   };
10
11  void cargar(struct producto *pro)
12  {
13      printf("Ingrese el código: ");
14      scanf("%i", &pro->codigo);
15      fflush(stdin);
16      printf("Ingrese su descripción: ");
17      gets(pro->descripcion);
18      printf("Ingrese su precio: ");
19      scanf("%f", &pro->precio);
20  }
21
22  void imprimir(struct producto prod)
23  {
24      printf("Código: %i\n", prod.codigo);
25      printf("Descripción: %s\n", prod.descripcion);
26      printf("Precio: %.2f", prod.precio);
27  }
28
```

```
29  int main()
30  {
31      setlocale(LC_ALL, "");
32      struct producto produ;
33      cargar(&produ);
34      imprimir(produ);
35      getch();
36      return 0;
37  }
38
```

Si ejecutamos este será el resultado:



Esta es la sintaxis más usada para acceder a los campos de un registro mediante un puntero:

```
void cargar(struct producto *pprod)
{
    printf("Ingrese codigo:");
    scanf("%i",&pprod->codigo);
    fflush(stdin);
    printf("Ingrese descripcion:");
    gets(pprod->descripcion);
    printf("Ingrese precio:");
    scanf("%f",&pprod->precio);
}
```

Nuevamente si analizamos algunas de las miles de funciones que contiene el sistema operativo Linux veremos que aparece el empleo de punteros a struct utilizando el operador -> para acceder a los campos del registro:

```
static int check_free_space(struct bsd_acct_struct *acct)
{
        struct kstatfs sbuf;

        if (time_is_before_jiffies(acct->needcheck))
                goto out;

        /* May block */
        if (vfs_statfs(&acct->file->f_path, &sbuf))
                goto out;

        if (acct->active) {
                u64 suspend = sbuf.f_blocks * SUSPEND;
                do_div(suspend, 100);
                if (sbuf.f_bavail <= suspend) {
                        acct->active = 0;
                        pr_info("Process accounting paused\n");
                }
```

Capítulo 150.- Parámetros de una función de tipo punteros a struct – 2

Problema propuesto

Se tiene la siguiente declaración de registro:

```
struct pais {
    char nombre[40];
    int cantidadhab;
};
```

Definir tres variables de tipo país en la función main.

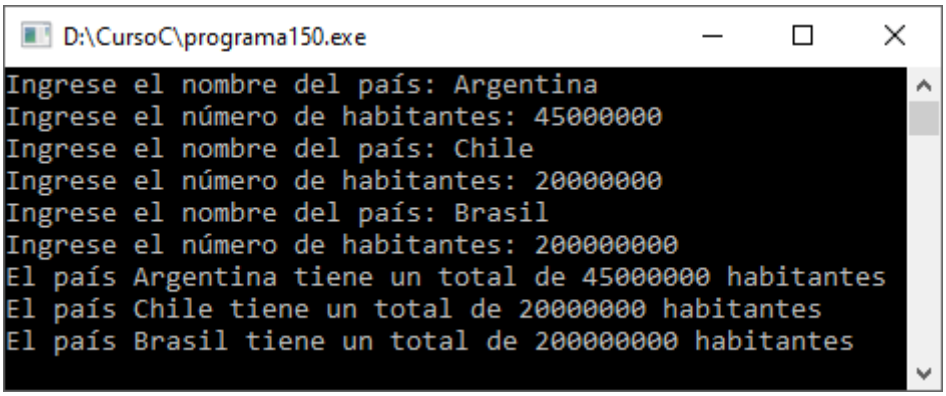
Crear una función que reciba un puntero de tipo país y cargue por teclado el nombre del país y la cantidad de habitantes.

Mostrar en otra función los datos cargados en cada país.

```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<locale.h>
4
5   struct pais{
6       char nombre[40];
7       int cantidadhab;
8   };
9
10  void cargar(struct pais *pa)
11  {
12      printf("Ingrese el nombre del país: ");
13      gets(pa->nombre);
14      printf("Ingrese el número de habitantes: ");
15      scanf("%i", &pa->cantidadhab);
16      fflush(stdin);
17  }
18
19  void imprimir(struct pais pa)
20  {
21      printf("El país %s tiene un total de %i habitantes\n",
22             pa.nombre, pa.cantidadhab);
23  }
24
25  int main()
26  {
27      setlocale(LC_CTYPE,"spanish");
28      struct pais pais1, pais2, pais3;
29      cargar(&pais1);
30      cargar(&pais2);
31      cargar(&pais3);
```

```
32          imprimir(pais1);
33          imprimir(pais2);
34          imprimir(pais3);
35          getch();
36          return 0;
37      }
38
```

Si ejecutamos este será el resultado:



Capítulo 151.- relación entre punteros y vectores – 1

Es importante entender que un vector es una puntero que contiene la dirección de la primer elemento.

```
int vec[5]={10,20,30,40,50};

int *pe;
```

Si luego igualamos pe con vec:

```
pe=vec;
```

Luego tenemos en memoria:

Dirección	Contenido	Nombre de la variable
1000	1004	vec
1004	10	
1008	20	
1012	30	
1016	40	
1020	50	
1024	1004	pe

La variable pe almacena la misma dirección de memoria que la variable vec.

Ahora podemos trabajar perfectamente con la variable pe para acceder a los elementos del vector.

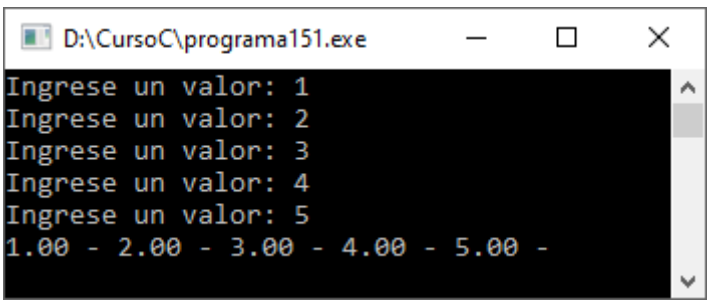
```
#include<stdio.h>

int main()
{
    int vec[5]={10,20,30,40,50};
    int *pe;
    int f;
    pe=vec;
    for(f=0;f<5;f++)
    {
        printf("%i ",pe[f]);
    }
    getch();
    return 0;
}
```

como vemos después de la asignación de pe=vec podemos acceder a los elementos del vector con cualquiera de las dos variables.
Es decir desde que presentamos el concepto de vectores estábamos trabajando con punteros.
Ahora podemos entender porque cuando pasamos un vector o matriz a una función se modifica la variable que le pasamos desde la función main, esto debido a que le pasamos la dirección donde se almacenan los elementos del vector o matriz.
Podemos inclusive utilizar la sintaxis de punteros en los parámetros de una función que recibe un vector.

Problema
Confeccionar un programa que permita cargar e imprimir un vector de 5 elementos de tipo float. Utilizar la sintaxis de punteros en los parámetros de las funciones.

```c
#include<stdio.h>
#include<conio.h>

void cargar(float *vec)
{
    for (int x=0; x<5; x++)
    {
        printf("Ingrese un valor: ");
        scanf("%f", &vec[x]);
    }
}

void imprimir(float *vec)
{
    for (int x=0; x<5; x++)
    {
        printf("%.2f - ", vec[x]);
    }
}

int main()
{
    float vector[5];
    cargar(vector);
    imprimir(vector);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 152.- relación entre punteros y vectores – 2

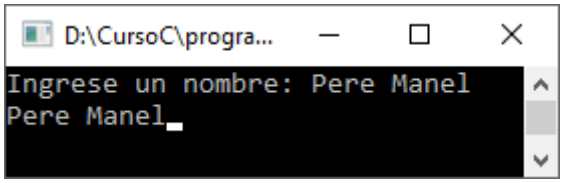
Problema propuesto

Se define en la main un vector de tipo char de 40 caracteres.

Implementar una función de carga e impresión de dicho vector utilizando la sintaxis de punteros cuando definamos los parámetros de las funciones.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   void cargar(char *nombre)
5   {
6       printf("Ingrese un nombre: ");
7       gets(nombre);
8   }
9
10  void imprimir(char *nombre)
11  {
12      printf("%s", nombre);
13  }
14
15  int main()
16  {
17      char nombre[40];
18      cargar(nombre);
19      imprimir(nombre);
20      getch();
21      return 0;
22  }
23
```

Si ejecutamos este será el resultado:



Capítulo 153.- Operadores ++ y – con variables de tipo puntero – 1

Los operadores ++ y – con tipo de dato puntero tiene un funcionamiento distinto que cuando lo incrementamos o decrementamos una variable entera.

Cuando utilizamos el operador ++ con un puntero incrementa la variable según el tamaño del tipo de dato que apunta. Si es un puntero entero incrementa la dirección en 4 bytes (en un sistema operativo de 32 bits) , si es un puntero de tipo char incrementa la dirección en 1 byte, etc.

Un ejemplo de cómo utilizar la sintaxis de punteros:

```
#include<stdio.h>
#include<conio.h>

int main()
{
    int vec[5]={10,20,30,40,50};
    int *pe;
    pe=vec;
    printf("%i\n",*pe);  // 10
    pe++;
    printf("%i\n",*pe);  // 20
    pe++;
    printf("%i\n",*pe);  // 30
    pe--;
    printf("%i\n",*pe);  // 20
    getch();
    return 0;
}
```

Como vec y pe son punteros a enteros luego podemos asignar vec a la variable pe:

```
pe=vec;
```

En este momento se tiene la dirección del primer elemento del vector, luego si imprimimos lo que apunta pe tenemos el valor 10:

```
printf("%i\n",*pe);  // 10
```

El operador ++ incrementa el contenido de la variable pe tantos bytes como sea el tamaño de las variables int en nuestro sistema operativo, por eso si accedemos a lo apuntado por pe tenemos el valor 20:

```
pe++;
printf("%i\n",*pe);  // 20
```

Un punto importante es que el vector vec si bien es un puntero no podemos utilizar el operador ++ o --, siempre apuntará al primer elemento del vector.

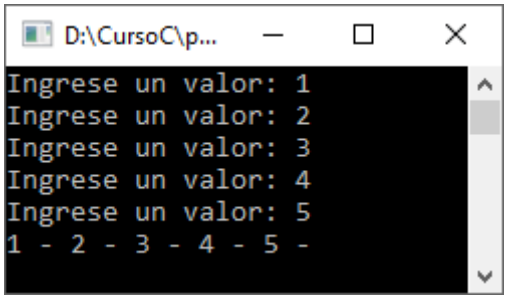
Problema

Desarrollar un programa para administrar un vector de 5 enteros.

En la función de carga e impresión utilizar la sintaxis de punteros para acceder a sus elementos (no utilizar la sintaxis de subíndice.

```c
#include<stdio.h>
#include<conio.h>

void cargar(int *pun)
 {
      for (int x=0; x<5; x++)
     {
         printf("Ingrese un valor: ");
         scanf("%i", &*pun);
         pun++;
     }
 }

 void imprimir(int *pun)
 {
     for (int x=0; x<5; x++)
      {
         printf("%i - ", *pun);
         pun++;
      }
      printf("\n");
 }

int main()
{
    int vector[5];
    cargar(vector);
    imprimir(vector);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 154.- Operadores ++ y – con variables de tipo puntero – 2

Problema propuesto

Implementar la función:

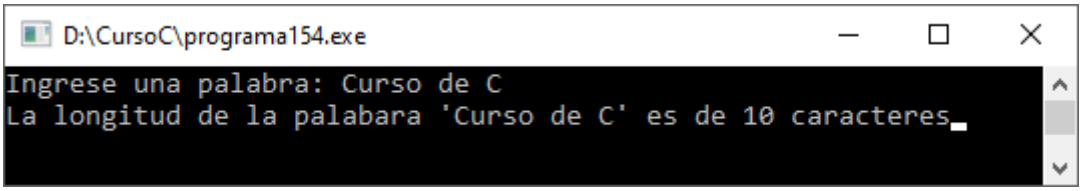
int largo(char *cadena)

Debe retornar el largo de la cadena utilizando la sintaxis de punteros para acceder a sus componentes. Recordar que el carácter '\0' indica el fin de la parte de información de la cadena.

No podemos utilizar la función strlen, ya que en realidad estamos pidiendo implementar el algoritmo de dicha función.

```
#include<stdio.h>
#include<conio.h>
#include<locale.h>

int largo(char *cadena)
{
    int cant=0;
    while (*cadena!='\0')
    {
        cant++;
        cadena++;
    }
    return cant;
}


int main()
{
    setlocale(LC_CTYPE, "spanish");
    char palabra[41];
    printf("Ingrese una palabra: ");
    gets(palabra);
    printf("La longitud de la palabara '%s' es de %i caracteres",
           palabra, largo(palabra));
    getch();
    return 0;
}

```

Si ejecutamos este será el resultado:



Capítulo 155.- Asignación dinámica de memoria (malloc y free) – 1

Una característica esencial del lenguaje C es la capacidad de requerir bloques de memoria variable durante la ejecución del programa.

Hasta ahora hemos visto que se reserva espacio para variables en el momento de definirlos:

```
int x;       // reservamos espacio para almacenar un entero
float z;     //reservamos espacio para almacenar un valor real
char c;      //reservamos espacio para almacenar un caracter
int vec[10]; //reservamos espacio para almacenar 10 enteros
int *pe;     //reservamos espacio para almacenar un puntero
```

El lenguaje nos permite en tiempo de ejecución solicitar espacio mediante la función malloc (**m**emory **alloc**ate = Asignar memoria) y luego de usarla en forma obligatoria debemos devolverla llamando a la función free.

Estas dos funciones se encuentran en el archivo de inclusión:

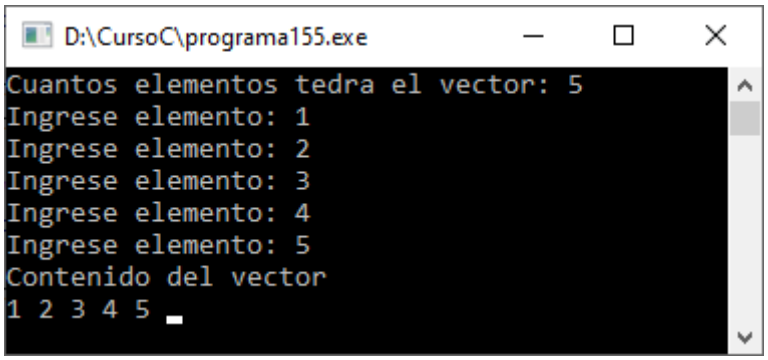
```
#include<stdlib.h>
```

Problema

Ingresar por teclado un entero que represente la cantidad de elementos que debe crearse un vector .
Crear el vector en forma dinámica, cargar e imprimir sus datos. Hacer todo en la main.

```
1  #include<stdio.h>
2  #include<conio.h>
3  #include<stdlib.h>
4
5  int main()
6  {
7      int *pe;
8      int tam;
9      printf("Cuantos elementos tedra el vector: ");
10     scanf("%i", &tam);
11     pe=malloc(tam*sizeof(int));
12     for (int f=0; f<tam; f++)
13     {
14         printf("Ingrese elemento: ");
15         scanf("%i", &pe[f]);
16     }
17     printf("Contenido del vector\n");
18     for (int f=0; f<tam; f++)
19     {
20         printf("%i ",pe[f]);
21     }
22     free(pe);
23     getch();
24     return 0;
25 }
```

Si ejecutamos este será el resultado:



Si nosotros intentamos definir un vector de la siguiente forma:

```
int tam;
printf("Cuantos elementos tedra el vector: ");
scanf("%i", &tam);

int vector[tam];
```

Si intentamos definir un vector partiendo del valor de una variable, esto nos dará un error.

Capítulo 156.- Asignación dinámica de memoria (malloc y free) – 2

Problema

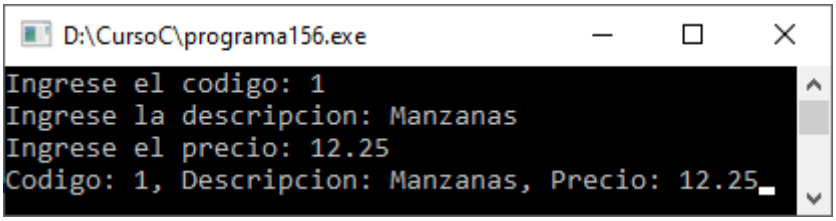
Se tiene la siguiente declaración de registro:

```
struct producto {
    int codigo;
    char descripcion[41];
    float precio;
};
```

Definir un puntero de tipo producto y luego mediante la función malloc crea un registro en la pila dinámica. Cargar el registro, imprimirlo y finalmente liberar el espacio reservado mediante la función free.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

struct producto{
    int codigo;
    char descripcion[41];
    float precio;
};

int main()
{
    struct producto *pro;
    pro=malloc(sizeof(struct producto));
    printf("Ingrese el codigo: ");
    scanf("%i", &pro->codigo);
    fflush(stdin);
    printf("Ingrese la descripcion: ");
    gets(pro->descripcion);
    printf("Ingrese el precio: ");
    scanf("%f", &pro->precio);
    printf("Codigo: %i, Descripcion: %s, Precio: %.2f",
           pro->codigo, pro->descripcion, pro->precio);
    free(pro);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



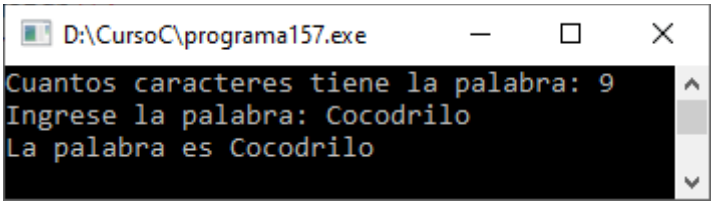
Capítulo 157.- Asignación dinámica de memoria (malloc y free) – 3

Problema propuesto

Pedir ingresar por teclado cuantas letras tiene una palabra. Seguidamente crear un vector en forma dinámica que reserve el espacio mínimo para ingresar dicha palabra.
Cargar por teclado la palabra, mostrarla y finalmente liberar el espacio requerido.

```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<stdlib.h>
4
5   int main()
6   {
7       char *puntero;
8       int caracteres;
9       printf("Cuantos caracteres tiene la palabra: ");
10      scanf("%i", &caracteres);
11      fflush(stdin);
12      puntero=malloc((caracteres+1)*sizeof(char));
13      printf("Ingrese la palabra: ");
14      gets(puntero);
15      printf("La palabra es %s", puntero);
16      free(puntero);
17      getch();
18      return 0;
19  }
20
```

Si ejecutamos este será el resultado:



Capítulo 158.- Estructuras dinámicas en C: Concepto de listas

Conocemos algunas estructuras de datos como son los vectores, matrices y registros. No son las únicas. Hay muchas situaciones donde utilizar alguna de estas estructuras nos proporcionarán una solución muy ineficiente (cantidad de espacio que ocupa en memoria, velocidad de acceso a la información, etc.)

Ejemplo 1. Imaginemos que debemos realizar un editor de texto, debemos elegir la estructura de datos para almacenar en memoria las distintas líneas que el operador irá escribiendo. Un solución factible es utilizar una matriz de caracteres. Pero como sabemos debemos especificar la cantidad de filas y columnas que ocupará de antemano. Podría ser por ejemplo 2000 filas y 200 columnas. Con esta definición estamos reservando de antemano 800000 bytes de memoria, no importa si el operador después de cargar una línea de 20 caracteres, igualmente ya se ha reservado una cantidad de espacio que permanecerá ociosa.

Tiene que existir alguna estructura de datos que pueda hacer más eficiente la solución del problema anterior.

Ejemplo 2. ¿Cómo estarán codificadas las plantillas de cálculo? ¿Reservarán espacio para cada casilla de la plantilla al principio? Si no la lleno, ¡lo mismo habrá reservado espacio?.

Utilizar una matriz para almacenar todas las casillas de una plantilla de cálculo seguro será ineficiente.

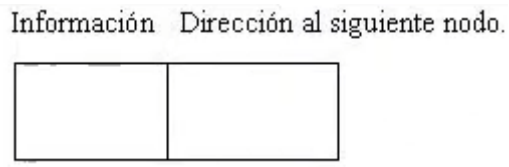
Bien, todos estos problemas y muchos más podrían ser resueltos en forma eficiente cuando conozcamos esta nueva estructura de datos (Listas, árboles).

Para trabajar con estructuras dinámicas en C debemos conocer y manejar muy bien el concepto de punteros que hemos ido presentando.

Concepto de listas

Una lista es un conjunto de nodos, cada uno de los cuales tiene dos campos: uno de información y un apuntador al siguiente nodo de la lista. Además un apuntador externo señala el primer nodo de la lista.

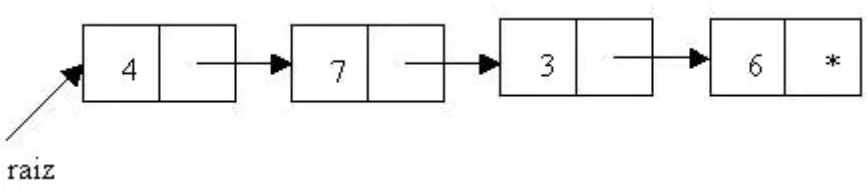
Representación gráfica de un nodo:



La información puede ser cualquier tipo de dato simple (int, char, float, etc.) o estructura de datos (registro, vector, etc.).

La dirección al siguiente nodo es un puntero.

Representación gráfica de una lista:



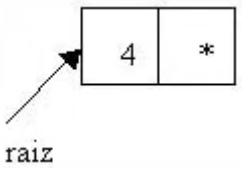
Como decíamos, una lista es una secuencia de nodos (en este caso cuatro nodos). La información de los nodos es este caso es un entero y siempre contiene un puntero que guarda la dirección del siguiente nodo.

Raiz es otro puntero externo a la lista que contiene la dirección del primer nodo.

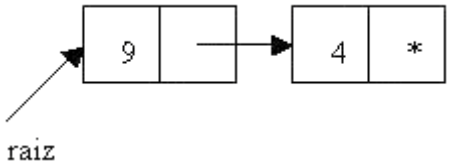
El estado de una lista varía durante la ejecución del programa:

De esta forma representamos gráficamente una lista vacía.

Si insertamos un nodo en la lista quedaría luego:



Si insertamos otro nodo al principio con el valor 9 tenemos:



Lo mismo podemos borrar nodos de cualquier parte de la lista.

Esto nos trae a la mente el primer problema planteado: el desarrollo del procesador de textos.

Podríamos utilizar una lista que inicialmente estuviera vacía e introdujéramos un nuevo nodo con cada línea que escribe el operador. Con esta estructura haremos un uso muy eficiente de la memoria.

Tipos de listas

- Listas tipo pila.
- Listas tipo cola.
- Listas genéricas.

Una lista se comporta como una pila si las inserciones y extracciones las hacemos por un lismo lado de la lista (por ejemplo donde apunta raiz). También se las llama listas LIFO (Last IN First Out – último en entrar primero en salir).

Una lista se comporta como una cola si las inserciones las hacemos al final y las extracciones las hacemos por el frente de la lista. También se las llama listas FOFO (First In First Out – primero en entrar primero en salir).

Una lista se comporta como genérica cuando las inserciones y extracciones se realizan en cualquier parte de la lista.

Podemos en algún momento insertar un nodo en medio de la lista, en otro momento al final, borrar uno del frente, borrar uno del fondo o uno interior, etc.

Capítulo 159.- Estructuras dinámicas en C: Listas tipo Pila – 1

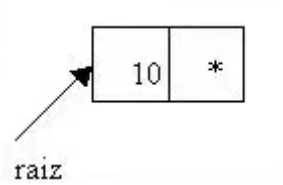
Una lista se comporta como una pila si las inserciones y extracciones las hacemos por el mismo lado de la lista. También se las llama LIFO (Last In First Out – último en entrar primero en salir).

Importante: Una pila al ser una lista puede almacenar en el campo de información cualquier tipo de valor (int, char, float, vector de caracteres, un registro, etc.).

Para estudiar el mecanismo de utilización de una pila supondremos que n el campo de información almacena un entero (para una fácil interpretación y codificación).

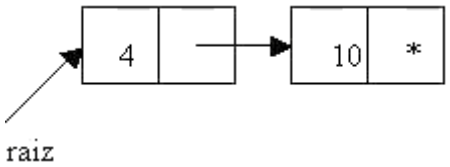
Inicialmente la PILA está vacía y decimos que el puntero raíz apunta a NULL (Si apunta a NULL decimos que no tiene una dirección de memoria, en realidad este valor NULL es el valor cero):

Insertamos un valor entero en la pila: insertar (10).



Luego de realizar la inserción la lista tipo pila que de esta manera: un nodo con el valor 10 y raíz apunta a dicho nodo. El puntero del nodo apunta a NULL ya que no hay otro nodo después de este.

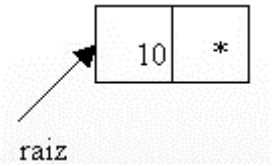
Insertamos luego el valor 4: Insertar (4):



Ahora el primer nodo de la pila es el que almacena el valor cuatro, raíz apunta a dicho nodo. Recordemos que raíz es el puntero externo a la lista que almacena la dirección del primer nodo. El nodo que acabamos de insertar en el campo puntero guarda la dirección del nodo que almacena el valor 10.

ahora qué sucede si extraemos un nodo de la pila. ¿Cuál se extrae? Como sabemos en una pila se extrae el último en entrar.

al extraer de la pila tenemos: extraer().



La pila ha quedado con un nodo.

Hay que tener cuidado que si al extraer un nuevo nodo la pila quedara vacía y no se podrá extraer otros valores (avisar que la pila está vacía).

Problema

Confeccionar un programa que administre una lista de tipo pila (se debe poder insertar, extraer e imprimir los datos de la pila).

```
1  #include<stdio.h>
2  #include<conio.h>
3  #include<stdlib.h>
4
5  struct nodo{
6      int info;
7      struct nodo *sig;
8  };
9
10 struct nodo *raiz=NULL;
11
12 void insertar(int x)
13 {
14     struct nodo *nuevo;
15     nuevo = malloc(sizeof(struct nodo));
16     nuevo->info=x;
17     if(raiz == NULL)
18     {
19         raiz = nuevo;
20         nuevo->sig = NULL;
21     }
22     else
23     {
24         nuevo->sig = raiz;
25         raiz = nuevo;
26     }
27 }
28
29
30 int main()
31 {
32     insertar(10);
33     insertar(4);
34     insertar(5);
35 }
```

Al método llega la información a insertar, en este caso en particular es un valor entero.

La creación de un nodo requiere de los siguientes pasos:

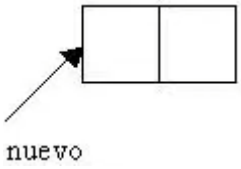
- Definición de un puntero o referencia a un tipo de nodo:

```
struct nodo *nuevo;
```

- Creación del nodo (reserva de espacio en la memoria dinámica);

```
nuevo = malloc(sizeof(struct nodo));
```

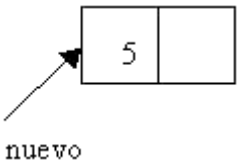
Cuando se ejecuta la función malloc se reserva espacio para el nodo. Realmente se crea el nodo cuando se ejecuta la función malloc.



Paso seguido debemos guardar la información del nodo (el campo info lo accedemos mediante el operador ->:

```
nuevo->info = x;
```

En el campo info almacenamos lo que llega en el parámetro x. por ejemplo si llega un 5 el nodo queda:

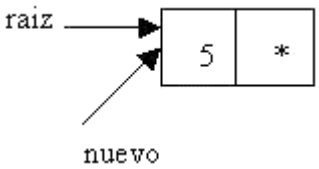


Por último queda enlazar el nodo que acabamos de crear al principio de la lista.

Si la lista está vacía debemos guardar en el campo sig del nodo el valor NULL para indicar que no hay otro nodo después de este, y hacer que raíz apunte al nodo creado (sabemos si una lista está vacía si raiz almacena un NULL, es decir el valor que le almacenamos cuando definimos la variable).

Tener en cuenta que podemos acceder a raiz sin haberla pasado como parámetro ya que es una variable global y puede ser accedida por cualquier función luego de su definición:

```
if (raiz == NULL)
{
    raiz = nuevo;
    nuevo->sig = NULL;
}
```



Gráficamente podemos observar que cuando indicamos raiz=nuevo, el puntero raiz guarda la dirección del nodo apuntado nuevo.

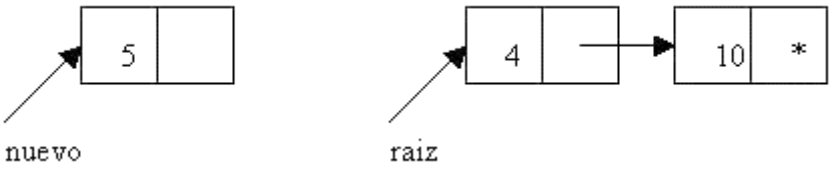
Tener en cuenta que cuando finaliza la ejecución de la función el puntero =nuevo" desaparece, pero no es nodo creado mediante la función malloc.

En caso que la lista no esté vacía, el puntero sig nodo que acabamos de crear debe apuntar al que es hasta este momento el primer nodo, es decir al nodo que apunta la raiz actualmente.

```
else
{
    nuevo->sig = raiz;
    raiz = nuevo;
}
```

Como primera actividad cargamos en el puntero sig el nodo apuntado por nuevo la dirección de raiz, y posteriormente raiz apunta al nodo que acabamos de crear, que será ahora el primero de la lista.

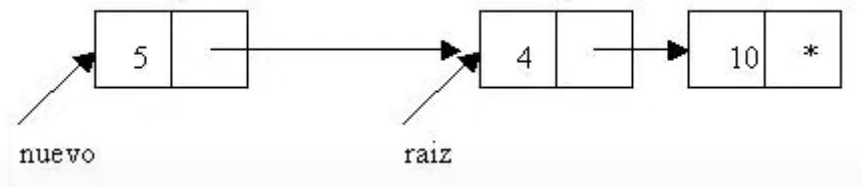
Antes de los enlaces tenemos:



Luego de ejecutar la línea:

```
nuevo->sig = raiz;
```

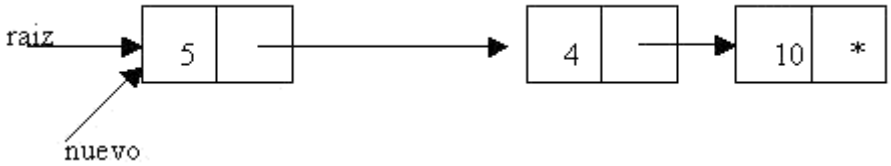
ahora tenemos:



Por último asignamos a raiz la dirección que almacena el puntero nuevo.

```
raiz = nuevo;
```

La lista queda:



La función extraer:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

struct nodo{
    int info;
    struct nodo *sig;
};

struct nodo *raiz=NULL;

```

```
12  void insertar(int x)
13  {
14      struct nodo *nuevo;
15      nuevo = malloc(sizeof(struct nodo));
16      nuevo->info=x;
17      if(raiz == NULL)
18      {
19          raiz = nuevo;
20          nuevo->sig = NULL;
21      }
22      else
23      {
24          nuevo->sig = raiz;
25          raiz = nuevo;
26      }
27  }
28
29  int extraer()
30  {
31      if (raiz != NULL)
32      {
33          int informacion = raiz->info;
34          struct nodo *bor = raiz;
35          raiz=raiz->sig;
36          free(bor);
37          return informacion;
38      }
39      else
40      {
41          return -1;
42      }
43  }
44
45  int main()
46  {
47      insertar(10);
48      insertar(4);
49      insertar(5);
50      printf("extraemos de la pila: %i\n", extraer());
51  }
```

El objetivo de la función extraer es retornar la información del primer nodo y además borrarlo de la lista.

Si la lista no está vacía (es decir raiz tiene un valor distinto de NULL) gurdamos en una variable local la información del primer nodo.

```
if (raiz != NULL)
{
    int informacion = raiz->info;
```

Creamos un puntero auxiliar y hacemos que apunte al nodo que vamos a borrar:

```
struct nodo *bor = raiz;
```

Avanzamos raiz al segundo nodo de la lista, ya que borraremos el primero (si no hay otro nodo más adelante en raiz se gurda NULL ya que el último nodo de la lista tiene en el puntero sig dicho valor):

```
raiz = raiz->sig;
```

Procedemos a eliminar el primer nodo de la lista llamando la función free (si no lo hacemos esto no genera error en el programa pero dicho espacio de memoria no se podrá asignar cuando hagamos otra llamada a la función malloc):

```
free(bor);
```

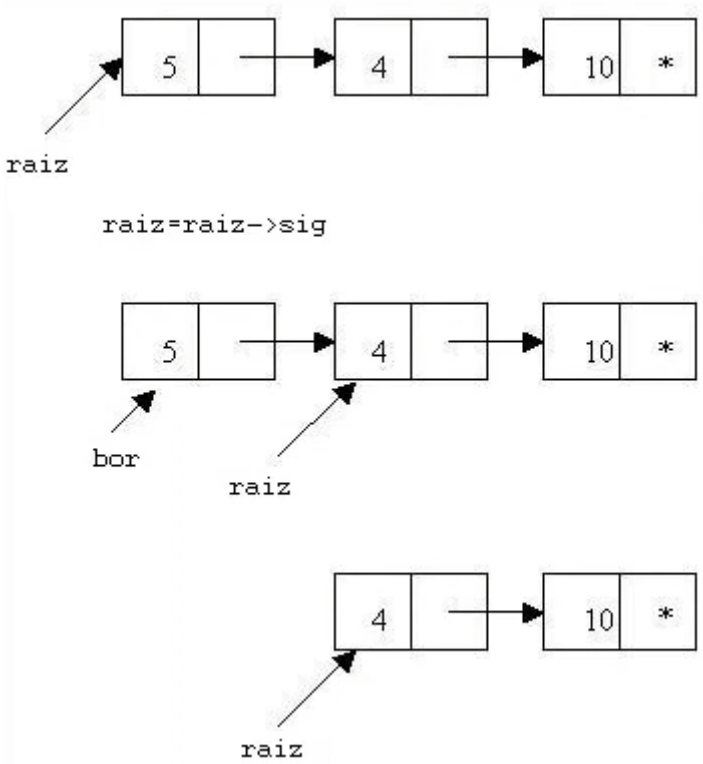
Retornamos la información:

```
return informacion;
```

En caso de estar vacía la pila retornamos el número -1 y lo tomamos como código de error (es decir nunca debemos guardar el entero -1 en la pila, podemos utilizar cualquier otro entero que indique que está vacía).

```
else
{
    return -1;
}
```

Es muy importante entender gráficamente el manejo de listas. La interpretación gráfica nos permitirá plantear inicialmente las soluciones para el manejo de listas.



Explicamos el método para recorrer una lista en forma completa e imprimir la información de cada nodo.

Explicamos el método para recorrer una lista completa e imprimir la información de cada nodo:

```c
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

struct nodo{
    int info;
    struct nodo *sig;
};

struct nodo *raiz=NULL;

void insertar(int x)
{
    struct nodo *nuevo;
    nuevo = malloc(sizeof(struct nodo));
    nuevo->info=x;
    if(raiz == NULL)
    {
        raiz = nuevo;
        nuevo->sig = NULL;
    }
    else
    {
        nuevo->sig = raiz;
        raiz = nuevo;
    }
}

int extraer()
{
    if (raiz != NULL)
    {
        int informacion = raiz->info;
        struct nodo *bor = raiz;
        raiz=raiz->sig;
        free(bor);
        return informacion;
    }
    else
    {
        return -1;
    }
}
```

```
45  void imprimir()
46  {
47      struct nodo *reco=raiz;
48      printf("Lista completa.\n");
49      while (reco!=NULL)
50      {
51          printf("%i ", reco->info);
52          reco=reco->sig;
53      }
54      printf("\n");
55  }
56
57  int main()
58  {
59      insertar(10);
60      insertar(4);
61      insertar(5);
62      printf("extraemos de la pila: %i\n", extraer());
63      imprimir();
64  }
```

Definimos un puntero auxiliar reco y hacemos que apunte al primer nodo de la lista:

```
struct nodo *reco=raiz;
```

Disponemos una estructura repetitiva que se repetirá reco sea distinto a NULL. Dentro de la estructura repetitiva hacemos que reco avance al siguiente nodo:

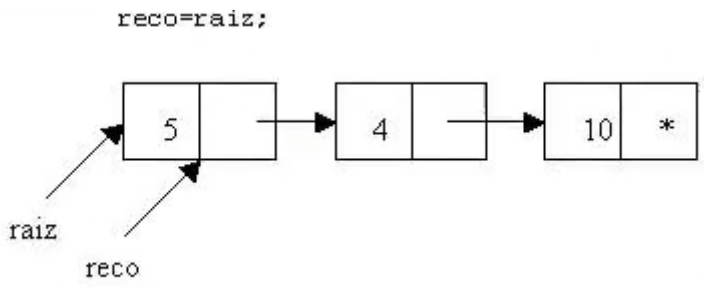
```
while (reco!=NULL)
{
    printf("%i ",reco->info);
    reco=reco->sig;
```

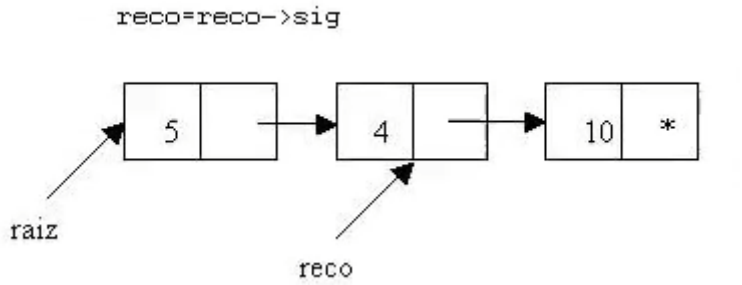
Es muy importante entender la línea:

```
reco=reco->sig;
```

Estamos diciendo que reco almacena la dirección que tiene el puntero sig del nodo apuntado actualmente por reco.

Gráficamente:





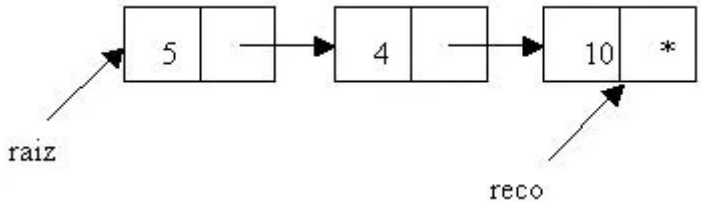
Al analizarse la condición:

```
while (reco!=NULL)
```

Es verdadero ya que reco apunta a un nodo y se vuelve a ejecutar la línea:

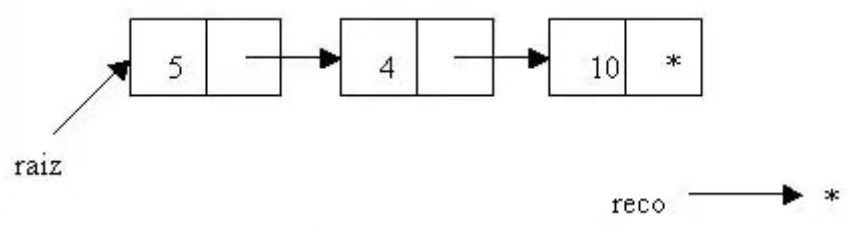
```
reco=reco->sig;
```

Ahora reco apunta al siguiente nodo:



La condición de while nuevamente se valúa en verdadera y avanza el puntero reco al siguiente nodo:

```
reco=reco->sig;
```



Ahora sí reco apunta a NULL (tiene almacenado un NULL) y ha llegado el final de la lista.

(Recordar que el último nodo de la lista tiene almacenado en el puntero sig el valor NULL, con el objetivo de saber que es el último nodo).

La última función analizar es liberar que tiene por objetivo liberar el espacio ocupado por los nodos de la lista, esta actividad es fundamental:

```c
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

struct nodo{
    int info;
    struct nodo *sig;
};

struct nodo *raiz=NULL;

void insertar(int x)
{
    struct nodo *nuevo;
    nuevo = malloc(sizeof(struct nodo));
    nuevo->info=x;
    if(raiz == NULL)
    {
        raiz = nuevo;
        nuevo->sig = NULL;
    }
    else
    {
        nuevo->sig = raiz;
        raiz = nuevo;
    }
}

int extraer()
{
    if (raiz != NULL)
    {
        int informacion = raiz->info;
        struct nodo *bor = raiz;
        raiz=raiz->sig;
        free(bor);
        return informacion;
    }
    else
    {
        return -1;
    }
}
```

```
44
45  void imprimir()
46  {
47      struct nodo *reco=raiz;
48      printf("Lista completa.\n");
49      while (reco!=NULL)
50      {
51          printf("%i ", reco->info);
52          reco=reco->sig;
53      }
54      printf("\n");
55  }
56
57  void liberar()
58  {
59      struct nodo *reco = raiz;
60      struct nodo *bor;
61      while (reco != NULL)
62      {
63          bor = reco;
64          reco = reco->sig;
65          free(bor);
66      }
67  }
68
69  int main()
70  {
71      insertar(10);
72      insertar(4);
73      insertar(5);
74      printf("extraemos de la pila: %i\n", extraer());
75      imprimir();
76      liberar();
77      getch();
78      return 0;
79  }
80
```

Definimos dos punteros auxiliares: reco y bor. A reco lo inicializamos con el primer nodo de la lista (en forma similar a como hicimos el imprimir donde recorremos toda la lista):

```
struct nodo *reco = raiz;
struct nodo *bor;
```

Mediante un while recorreremos toda la lista:

```
while (reco != NULL)
```

Dentro del while inicializamos el puntero bor con la dirección del nodo que apunta a reco:

```
bor = reco;
```

Inmediatamente avanzamos reco al siguiente nodo:

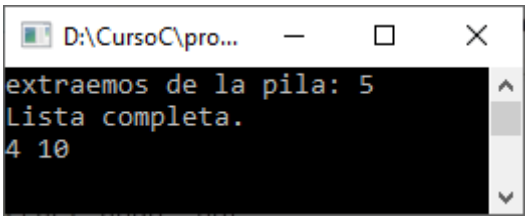
```
reco = reco->sig;
```

Y procedemos a borrar el nodo apuntado por bor:

```
free(bor);
```

Este while se repite mientras haya nodos en la lista.

Ejecutamos este será el resultado:



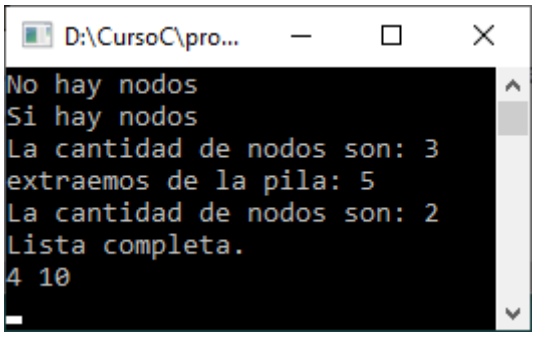
Capítulo 160.- Estructuras dinámicas en C: Listas tipo Pila – 2

Problema:

Agregar al problema anterior otras dos funciones que retornen la cantidad de nodos y otro que indique si está vacía (1=vacía y 0=no vacía).

```
69  int cantidadNodos()
70  {
71      struct nodo *reco=raiz;
72      int cant=0;
73      while (reco!=NULL)
74      {
75          cant++;
76          reco=reco->sig;
77      }
78      return cant;
79  }
80
81  int vacia()
82  {
83      if(raiz!=NULL)
84      {
85          return 0;
86      }
87      else
88      {
89          return 1;
90      }
91  }
92
93  int main()
94  {
95      if (vacia()==1)
96          printf("No hay nodos\n");
97      else
98          printf("Si hay nodos\n");
99      insertar(10);
100     if (vacia()==1)
101         printf("No hay nodos\n");
102     else
103         printf("Si hay nodos\n");
104     insertar(4);
105     insertar(5);
106     printf("La cantidad de nodos son: %i\n", cantidadNodos());
107     printf("extraemos de la pila: %i\n", extraer());
108     printf("La cantidad de nodos son: %i\n", cantidadNodos());
109     imprimir();
110     liberar();
111     getch();
112     return 0;
113 }
```

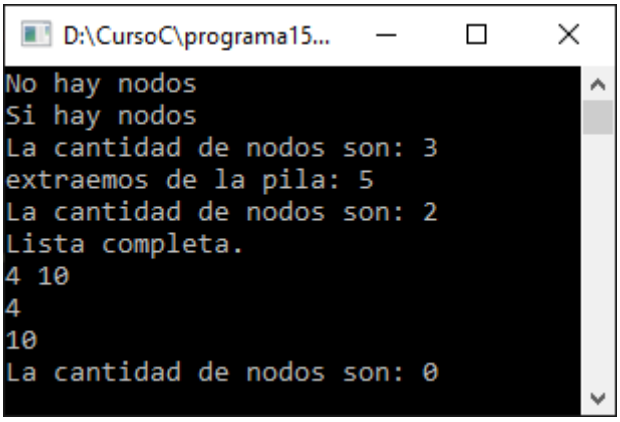
Si ejecutamos este será el resultado:



Modificamos el método main:

```
 93  int main()
 94  {
 95      if (vacia()==1)
 96          printf("No hay nodos\n");
 97      else
 98          printf("Si hay nodos\n");
 99      insertar(10);
100      if (vacia()==1)
101          printf("No hay nodos\n");
102      else
103          printf("Si hay nodos\n");
104      insertar(4);
105      insertar(5);
106      printf("La cantidad de nodos son: %i\n", cantidadNodos());
107      printf("extraemos de la pila: %i\n", extraer());
108      printf("La cantidad de nodos son: %i\n", cantidadNodos());
109      imprimir();
110      while (vacia()==0)
111      {
112          printf("%i \n", extraer());
113      }
114      printf("La cantidad de nodos son: %i\n", cantidadNodos());
115      liberar();
116      getch();
117      return 0;
118  }
```

Si ejecutamos este será el resultado:



Capítulo 161.- Estructuras dinámicas en C: Listas tipo Pila – 3

Problema propuesto

Agregar otra función al programa desarrollado para administrar una pila que retorne la información del primer nodo de la Pila sin borrarlo.

```
 93  int primerValorNodo()
 94  {
 95      if (raiz!=NULL)
 96      {
 97          return raiz->info;
 98      }
 99      else
100      {
101          return -1;
102      }
103  }
104
105  int main()
106  {
107      if (vacia()==1)
108          printf("No hay nodos\n");
109      else
110          printf("Si hay nodos\n");
111      insertar(10);
112      if (vacia()==1)
113          printf("No hay nodos\n");
114      else
115          printf("Si hay nodos\n");
116      insertar(4);
117      insertar(5);
118      printf("La cantidad de nodos son: %i\n", cantidadNodos());
119      printf("extraemos de la pila: %i\n", extraer());
120      printf("La cantidad de nodos son: %i\n", cantidadNodos());
121      if (primerValorNodo()==-1)
122      {
123          printf("La pila esta vacia");
124      }
125      else
126      {
127          printf("El primer nodo de la Pila tiene el valor %i\n",
128                  primerValorNodo());
129      }
130      imprimir();
131      while (vacia()==0)
132      {
133          printf("%i \n", extraer());
134      }
135      printf("La cantidad de nodos son: %i\n", cantidadNodos());
136      liberar();
137      getch();
138      return 0;
139  }
```

Capítulo 162.- Estructuras dinámicas en C: Listas tipo Pila – Problema de aplicación

Hasta ahora hemos visto como desarrollar los algoritmos para administrar una lista tipo Pila, hemos visto que hay bastante complejidad en el manejo de punteros pero todo este acarrea ventajas en la solución de problemas que requieren una estructura de tipo Pila.

Plateo del problema:

Este práctico tiene por objetivo mostrar la importancia de las pilas en las Ciencias de la computación y más precisamente en la programación de software de bajo nivel.

Todo compilador o intérprete de un lenguaje tiene un módulo dedicado a analizar si una expresión está correctamente codificada, es decir que los paréntesis estén abiertos y cerrados en un orden lógico y bien balanceados.

Se debe desarrollar un programa que tenga las siguientes responsabilidades:

- Ingresar una fórmula que contenga paréntesis, corchetes y llaves.
- Validar que los ( ) [ ] y { } estén correctamente balanceados.

Veamos como nos puede ayudar el empleo de una pila para solucionar este problema.

Primero cargaremos la fórmula por teclado.

Ejemplo de fórmula: (2+[3-12]*{/3})

El algoritmo de validación es el siguiente:

Analizamos carácter a carácter la presencia de los paréntesis, corchetes y llaves.

Si viene símbolo de apertura los almacenamos en la pila.

Si vienen símbolos de cerrado extraemos de la pila y verificamos si está el mismo símbolo pero de apertura: es caso negativo podemos inferir que la fórmula no está correctamente balanceada.

Si al finalizar el análisis del último carácter de la fórmula la pila está vacía podemos concluir que está correctamente balanceada.

Ejemplos de fórmulas no balanceadas:

```
}(2+[3-12]*{8/3})
Incorrecta: llega una } de cerrado y la pila está vacía.
{[2+4}]
Incorrecta: llega una llave } y en el tope de la pila hay un corchete [.
{[2+4]
Incorrecta: al finalizar el análisis del último caracter en la pila queda
pendiente una llave {.
```

Vamos a comentar todo el proyecto:

```
1  #include<stdio.h>
2  #include<conio.h>
3  #include<stdlib.h>
4  #include<string.h>
```

Cargamos los librerías.

```
6   struct nodo {
7       char simbolo;
8       struct nodo *sig;
9   };
```

Creamos un registro de tipo nodo, que almacena un carácter.

```
11  struct nodo *raiz=NULL;
```

Creamos un puntero de tipo nodo llamado raiz y le asignamos el valor NULL,

```
13  void insertar(char x)
14  {
15      struct nodo *nuevo;
16      nuevo=malloc(sizeof(struct nodo));
17      nuevo->simbolo = x;
18      if (raiz == NULL)
19      {
20          raiz = nuevo;
21          nuevo->sig = NULL;
22      }
23      else
24      {
25          nuevo->sig = raiz;
26          raiz = nuevo;
27      }
28  }
```

Creamos el método insertar para agregar nuevos nodos.

```
30  char extraer()
31  {
32      if (raiz != NULL)
33      {
34          char informacion = raiz->simbolo;
35          struct nodo *bor = raiz;
36          raiz=raiz->sig;
37          free(bor);
38          return informacion;
39      }
40      else
41      {
42         return -1;
43      }
44  }
```

Creamos el método extraer para eliminar el último nodo que hemos insertado.

```
46  void liberar()
47  {
48      struct nodo *reco = raiz;
49      struct nodo *bor;
50      while (reco != NULL)
51      {
52          bor=reco;
53          reco=reco->sig;
54          free(bor);
55      }
56  }
```

Creamos el método liberar para borrar todos los nodos de la memoria.

```
58  int vacia()
59  {
60      if(raiz == NULL)
61          return 1;
62      else
63          return 0;
64  }
```

Con este método verificamos si la Pila está vacía.

```
66  void cargar(char *formula)
67  {
68      printf("Ingrese la formula: ");
69      gets(formula);
70  }
```

Con este método podemos ingresar la formula que tiene que comprobar.

```
72  int verificarBalanceada(char *formula)
73  {
74      for (int f=0; f<strlen(formula); f++)
75      {
76          if(formula[f]=='(' || formula[f]=='[' || formula[f]=='{')
77              {
78                  insertar(formula[f]);
79              }
80              else
81              {
82                  if(formula[f]==')')
83                  {
84                      if (extraer()!='(')
85                      {
86                          return 0;
87                      }
88                  }
89                  else
90                  {
91                     if(formula[f]==']')
92                     {
93                         if (extraer()!='[')
94                      {
95                          return 0;
96                      }
97                  }
98                  else
99                  {
100                     if(formula[f]=='}')
101                     {
102                         if (extraer()!='{')
103                         {
104                             return 0;
```

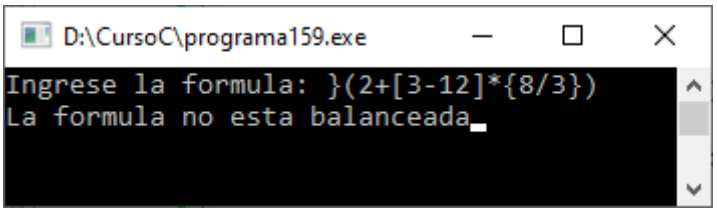
```
105                         }
106                     }
107                 }
108             }
109         }
110     }
111     if (vacia())
112     {
113         return 1;
114     }
115     else
116     {
117         return 0;
118     }
119 }
```

Con este método verificamos si la formula está correctamente balanceada.

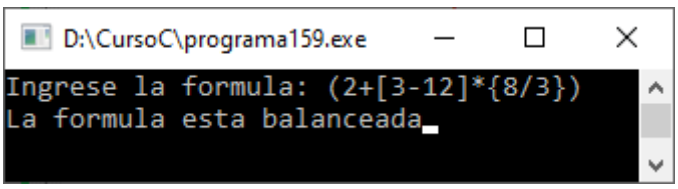
```
121 int main()
122 {
123     char formula[101];
124     cargar(formula);
125     if(verificarBalanceada(formula))
126     {
127         printf("La formula esta balanceada");
128     }
129     else
130     {
131         printf("La formula no esta balanceada");
132     }
133     liberar();
134     getch();
135     return 0;
136 }
137
```

El método principal (main).

Vamos a ingresar una formula que no esté balanceada:



Ahora vamos a ingresar un formula que esté balanceada:



Capítulo 163.- Estructura dinámica en C: Lista tipo Cola

Una lista se comporta como una cola si las inserciones las hacemos al final y las extracciones las hacemos por el frente de la lista. También se las llama listas FIFO (First In First Out – primero en entrar primero en salir).

Confeccionaremos un programa que permita administrar una lista tipo cola. Desarrollaremos las funciones de insertar, extraer, vaciar, imprimir y liberar.

```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<stdlib.h>
```

Agregamos las librerías.

```
5   struct nodo {
6       int info;
7       struct nodo *sig;
8   };
```

Creamos un registro tipo nodo.

```
10  struct nodo *raiz=NULL;
11  struct nodo *fondo=NULL;
```

Definimos dos punteros de tipo nodo *raiz igual a NULL y *fondo=NULL.

```
13  int vacia()
14  {
15      if(raiz==NULL)
16      {
17          return 1;
18      }
19      else
20      {
21          return 0;
22      }
23  }
```

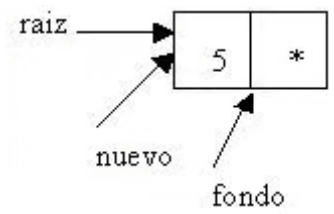
Con este método controlamos si la Fila está llena o vacía.

```
25  void insertar(int x)
26  {
27      struct nodo *nuevo;
28      nuevo=malloc(sizeof(struct nodo));
29      nuevo->info=x;
30      nuevo->sig=NULL;
31      if(vacia())
32      {
33          raiz=nuevo;
34          fondo=nuevo;
35      }
36      else
37      {
```

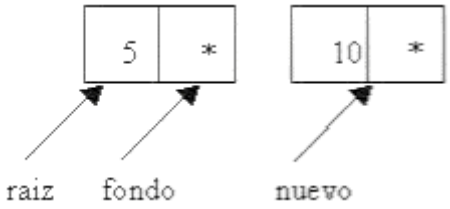
```
38            fondo->sig=nuevo;
39            fondo=nuevo;
40        }
41    }
```

recordemos que definimos un puntero llamado nuevo, luego creamos e nodo con la función malloc y cargamos los dos atributos, el de información con lo que llega en el parámetro y el puntero con NULL ya que se insertará al final de la lista, es decir no hay otro después de este.

Si la lista está vacía:

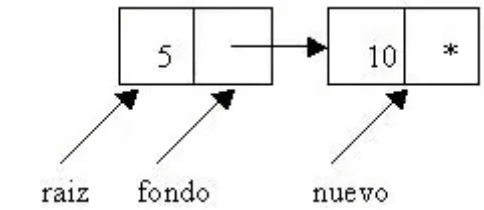


En caso de no estar vacía:



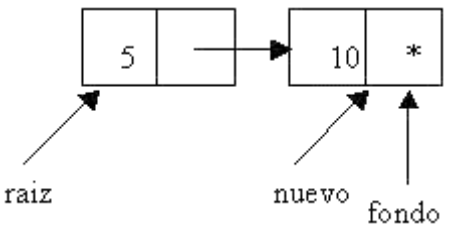
Debemos enlazar el puntero sig del último nodo con el nodo recién creado.

```
fondo->sig=nuevo;
```



Y por último el puntero externo fondo debe apuntar al nodo apuntado por nuevo:

```
fondo=nuevo;
```



Con esto ya tenemos correctamente enlazados los nodos en la lista tipo cola. Recordar que el puntero nuevo desaparece cuando se sale del método insertar, pero el nodo creado no se pierde porque queda enlazado en la lista.

```
43  int extraer()
44  {
45      if (!vacia())
46      {
47          int informacion=raiz->info;
48          struct nodo *bor=raiz;
49          if (raiz==fondo)
50          {
51              raiz=NULL;
52              fondo=NULL;
53          }
54          else
55          {
56              raiz=raiz->sig;
57          }
58          free(bor);
59          return informacion;
60      }
61      else
62      {
63          return -1;
64      }
65  }
```

El método extraer.

```
67  void imprimir()
68  {
69      struct nodo *reco=raiz;
70      printf("Listado completo de la lista tipo cola\n");
71      while (reco!=NULL)
72      {
73          printf(" %i ", reco->info);
74          reco=reco->sig;
75      }
76      printf("\n");
77  }
```

El método imprimir.

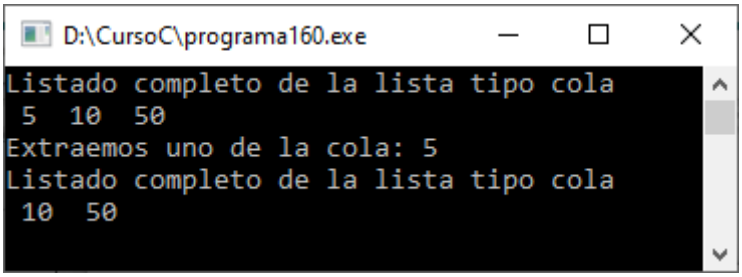
```
79  void liberar()
80  {
81      struct nodo *reco=raiz;
82      struct nodo *bor;
83      while (reco!=NULL)
84      {
85          bor=reco;
86          reco=reco->sig;
87          free(bor);
88      }
89  }
```

Método para borrar todos los nodos de la memoria.

```
91  int main()
92  {
93      insertar(5);
94      insertar(10);
95      insertar(50);
96      imprimir();
97      printf("Extraemos uno de la cola: %i\n", extraer());
98      imprimir();
99      liberar();
100     getch();
101     return 0;
102 }
```

El método principal.

Si ejecutamos este será el resultado:



Capítulo 164.- Estructura dinámica en C: Lista tipo Cola – Problema de aplicación

Este práctico tiene por objetivo mostrar la importancia de las colas en las Ciencias de la Computación y más precisamente en las simulaciones.

Las simulaciones permiten analizar situaciones de la realidad sin la necesidad de ejecutarlas realmente. Tiene el beneficio que su costo es muy inferior a hacer pruebas en la realidad.

Desarrollar un programa para la simulación de un cajero automático.

Se cuenta con la siguiente información:

- Llegan clientes a la puerta del cajero cada 2 ó 3 minutos.
- Cada cliente tarda entre 2 y 4 minutos para ser atendido.

Obtener la siguiente información:

1- Cantidad de clientes que se atienden en 10 horas.
2- Cantidad de clientes que hay en la cola después de 10 horas.
3- Hora de llegada del primer cliente que no es atendido luego de 10 horas (es decir la persona que está primera en la cola cuando se cumplen 10 horas).

```
1  #include<stdio.h>
2  #include<conio.h>
3  #include<stdlib.h>
4  #include<time.h>
```

Importamos las librerías.

```
6  struct nodo {
7      int info;
8      struct nodo *sig;
9  };
```

Creamos el registro nodo.

```
11 struct nodo *raiz=NULL;
12 struct nodo *fondo=NULL;
```

Creamos los punteros de memoria *raiz igual a NULL y *fondo igual a NULL.

```
14 int vacia()
15 {
16     if (raiz == NULL)
17         return 1;
18     else
19         return 0;
20 }
```

Este método comprueba si hay nodos o está vacía.

```
23  void insertar(int x)
24  {
25      struct nodo *nuevo;
26      nuevo=malloc(sizeof(struct nodo));
27      nuevo->info=x;
28      nuevo->sig=NULL;
29      if (vacia())
30      {
31          raiz = nuevo;
32          fondo = nuevo;
33      }
34      else
35      {
36          fondo->sig = nuevo;
37          fondo = nuevo;
38      }
39  }
```

Este método agrega nodos a una Cola.

```
41  int extraer()
42  {
43      if (!vacia())
44      {
45          int informacion = raiz->info;
46          struct nodo *bor = raiz;
47          if (raiz == fondo)
48          {
49              raiz = NULL;
50              fondo = NULL;
51          }
52          else
53          {
54              raiz=raiz->sig;
55          }
56          free(bor);
57          return informacion;
58      }
59      else
60          return -1;
61  }
```

Este método extrae nodos de una lista tipo Cola.

```
63  void liberar()
64  {
65      struct nodo *reco = raiz;
66      struct nodo *bor;
67      while (reco != NULL)
68      {
69          bor = reco;
70          reco = reco->sig;
71          free(bor);
72      }
73  }
```

Este método elimina de la memoria todos los nodos.

```
75  int cantidad()
76  {
77      struct nodo *reco = raiz;
78      int cant = 0;
79      while (reco != NULL)
80      {
81          cant++;
82          reco = reco->sig;
83      }
84      return cant;
85  }
```

Este método retorna la cantidad de nodos que hay en la Lista de tipo Cola.

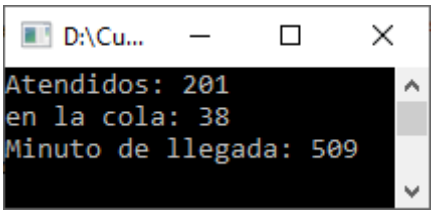
```
87  void simulacion()
88  {
89      srand(time(NULL));
90      //Estado del cajero 0=desocupado 1=ocupado
91      int estado=0;
92      int llegada=rand()%2 + 2;
93      int salida=-1;
94      int cantAtendidas=0;
95      for(int minuto=0; minuto<600; minuto++)
96      {
97          if(llegada==minuto)
98          {
99              if(estado==0)
100             {
101                 estado=1;
102                 salida = minuto + 2 + rand()%3;
103             }
104             else
105             {
106                 insertar(minuto);
107             }
108             llegada=minuto + 2 + rand()%2;
109         }
110         if (salida==minuto)
111         {
112             estado=0;
113             cantAtendidas++;
114             if(!vacia())
115             {
116                 extraer();
117                 estado=1;
118                 salida = minuto+2+rand()%3;
119             }
120         }
121     }
122     printf("Atendidos: %i\n", cantAtendidas);
123     printf("en la cola: %i\n", cantidad());
124     printf("Minuto de llegada: %i", extraer());
125 }
```

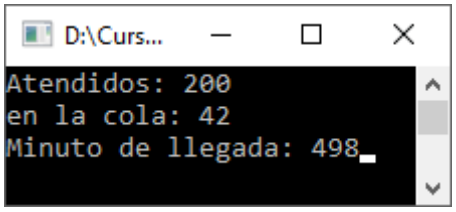
Este método realiza la simulación para obtener la cantidad de personas atendidas, el número de personas que aun están en la cola, y la siguiente persona que se encuentra al principio de la cola cuando han transcurrido las 10 horas.

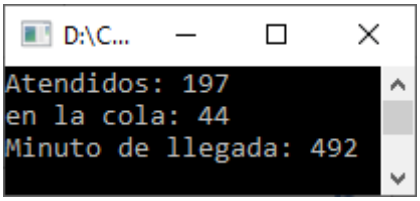
```
127  int main()
128  {
129      simulacion();
130      liberar;
131      getch();
132      return 0;
133  }
```

El método main.

Vamos a ejecutar 3 veces para realizar 3 simulaciones:







Capítulo 165.- Estructuras dinámicas en C: Listas genéricas – 1

Continuando con el tema de listas trabajaremos con las listas genéricas en C. Una lista se comporta como genérica cuando las inserciones y extracciones se realizan en cualquier parte de la lista.

Codificaremos una serie de métodos para administrar listas genéricas.

Métodos a desarrollar:

Insertar un nodo en la posición (pos) y con la información que hay en el parámetro x:

```
 void insertar(int pos, int x)
```

```
void insertar(int pos, int x)
{
    if (pos<=cantidad()+1)
    {
        struct nodo *nuevo;
        nuevo=malloc(sizeof(struct nodo));
        nuevo->info=x;
        if (pos==1)
        {
            nuevo->sig=raiz;
            raiz=nuevo;
        }
        else
        {
            if (pos==cantidad()+1)
            {
                struct nodo *reco=raiz;
                while (reco->sig!=NULL)
                {
                    reco=reco->sig;
                }
                reco->sig=nuevo;
                nuevo->sig=NULL;
            }
            else
            {
                struct nodo *reco=raiz;
                for(int f=1; f<=pos-2; f++)
                {
                    reco=reco->sig;
                }
                struct nodo *siguiente=reco->sig;
                reco->sig=nuevo;
                nuevo->sig=siguiente;
            }
        }
    }
}
```

Extraer la información del nodo en la posición indicada (pos). Se debe eliminar el nodo.

```
 int extraer(int pos)
```

```
int extraer(int pos)
{
    if(pos<=cantidad())
    {
        int informacion;
        struct nodo *bor;
        if(pos==1)
        {
            informacion=raiz->info;
            bor=raiz;
            raiz=raiz->sig;
        }
        else
        {
            struct nodo *reco=raiz;
            for(int f=1; f<=pos-2; f++)
            {
                reco=reco->sig;
            }
            struct nodo *prox=reco->sig;
            reco->sig=prox->sig;
            informacion=prox->info;
            bor=prox;
        }
        free(bor);
        return informacion;
    }
    else
        return -1;
}
```

Borrar el nodo de la posición (pos).

```
 void borrar(int pos)
```

```
void borrar(int pos)
{
    if(pos<=cantidad())
    {
        int informacion;
        struct nodo *bor;
        if(pos==1)
        {
            bor=raiz;
            raiz=raiz->sig;
        }
        else
        {
            struct nodo *reco=raiz;
            for(int f=1; f<=pos-2; f++)
```

```
            {
                reco=reco->sig;
            }
            struct nodo *prox=reco->sig;
            reco->sig=prox->sig;
            bor=prox;
        }
        free(bor);
    }
}
```

Intercambiar la información de los nodos en las posiciones pos1 y pos2.

```
 void intercambiar(int pos1,int pos2)
```

```
void intercambiar(int pos1, int pos2)
{
    if (pos1<=cantidad() && pos2<=cantidad())
    {

        struct nodo *reco1=raiz;
        for(int f=1;f<pos1; f++)
        {
            reco1=reco1->sig;
        }
        struct nodo *reco2=raiz;
        for(int f=1;f<pos2; f++)
        {
            reco2=reco2->sig;
        }
        int aux=reco1->info;
        reco1->info=reco2->info;
        reco2->info=aux;
    }
}
```

Retornar el valor del nodo con mayor información.

```
 int mayor()
```

```
int mayor()
{
    if(!vacia())
    {
        int may=raiz->info;
        struct nodo *reco=raiz->sig;
        while (reco!=NULL)
        {
            if (reco->info>may)
            {
                may=reco->info;
            }
            reco=reco->sig;
```

```
        }
        return may;
    }
    else
        return -1;
}
```

Retornar la posición del nodo con la mayor información.

```
 int posMayor()
```

```
int posMayor()
{
    if(!vacia())
    {
        int may=raiz->info;
        int x = 1;
        int pos = x;
        struct nodo *reco=raiz->sig;
        while (reco!=NULL)
        {
            if (reco->info>may)
            {
                may=reco->info;
                pos = x;
            }
            reco=reco->sig;
            x++;
        }
        return pos;
    }
    else
        return -1;
}
```

Retornar la cantidad de nodos de la lista.

```
 int cantidad()
```

```
int cantidad()
{
    int cant=0;
    struct nodo *reco=raiz;
    while(reco!=NULL)
    {
        cant++;
        reco=reco->sig;
    }
    return cant;
}
```

Debemos retornar 1 si la lista está ordenada de menor a mayor, 0 en caso contrario.

```
int ordenada()
```

```
int ordenada()
{
    if (cantidad()>1)
    {
        struct nodo *recol = raiz;
        struct nodo *reco2 =raiz->sig;
        while (reco2 != NULL)
        {
            if (reco2->info<recol->info)
            {
                return 0;
            }
            reco2 = reco2->sig;
            recol = recol->sig;
        }
    }
    return 1;
}
```

Debe retornar 1 si existe la información que llega en el parámetro, o en caso contrario.

```
int existe(int info)
```

```
int existe(int x)
{
    struct nodo *reco = raiz;
    while (reco != NULL)
    {
        if(reco->info == x)
            return 1;
        reco = reco->sig;
    }
    return 0;
}
```

El método vacía debe retornar 1 si está vacía y 0 se no lo está.

```
int vacia()
```

```
int vacia()
{
    if(raiz==NULL)
        return 1;
    else
        return 0;
}
```

Código completo:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

struct nodo {
   int info;
   struct nodo *sig;
};

struct nodo *raiz=NULL;
```

```
void liberar()
{
   struct nodo *reco=raiz;
   struct nodo *bor;
   while(reco!=NULL)
   {
      bor=reco;
      reco=reco->sig;
      free(bor);
   }
}
```

```
int vacia()
{
   if(raiz==NULL)
      return 1;
   else
      return 0;
}
```

```
int cantidad()
{
   int cant=0;
   struct nodo *reco=raiz;
   while(reco!=NULL)
   {
      cant++;
      reco=reco->sig;
   }
   return cant;
}
```

```
void insertar(int pos, int x)
{
   if (pos<=cantidad()+1)
   {
      struct nodo *nuevo;
      nuevo=malloc(sizeof(struct nodo));
      nuevo->info=x;
      if (pos==1)
      {
         nuevo->sig=raiz;
         raiz=nuevo;
      }
      else
      {
         if (pos==cantidad()+1)
         {
            struct nodo *reco=raiz;
            while (reco->sig!=NULL)
            {
               reco=reco->sig;
            }
            reco->sig=nuevo;
            nuevo->sig=NULL;
         }
         else
         {
            struct nodo *reco=raiz;
            for(int f=1; f<=pos-2; f++)
            {
               reco=reco->sig;
            }
            struct nodo *siguiente=reco->sig;
            reco->sig=nuevo;
            nuevo->sig=siguiente;
         }
      }
   }
}
```

```
int extraer(int pos)
{
   if(pos<=cantidad())
   {
      int informacion;
      struct nodo *bor;
      if(pos==1)
      {
         informacion=raiz->info;
         bor=raiz;
         raiz=raiz->sig;
      }
      else
      {
         struct nodo *reco=raiz;
         for(int f=1; f<=pos-2; f++)
         {
            reco=reco->sig;
         }
         struct nodo *prox=reco->sig;
         reco->sig=prox->sig;
         informacion=prox->info;
         bor=prox;
      }
      free(bor);
      return informacion;
   }
   else
      return -1;
}
```

```
void imprimir()
{
   struct nodo *reco=raiz;
   printf("Listado completo\n");
   while(reco!=NULL)
   {
     printf(" %i ", reco->info);
     reco=reco->sig;
   }
   printf("\n_____\n");
}
```

```
void borrar(int pos)
{
   if(pos<=cantidad())
   {
      int informacion;
      struct nodo *bor;
      if(pos==1)
      {
         bor=raiz;
         raiz=raiz->sig;
      }
      else
      {
         struct nodo *reco=raiz;
         for(int f=1; f<=pos-2; f++)
         {
            reco=reco->sig;
         }
         struct nodo *prox=reco->sig;
         reco->sig=prox->sig;
         bor=prox;
      }
      free(bor);
   }
}
```

```
void intercambiar(int pos1, int pos2)
{
   if (pos1<=cantidad() && pos2<=cantidad())
   {

      struct nodo *reco1=raiz;
      for(int f=1;f<pos1; f++)
      {
         reco1=reco1->sig;
      }
      struct nodo *reco2=raiz;
      for(int f=1;f<pos2; f++)
      {
         reco2=reco2->sig;
      }
      int aux=reco1->info;
      reco1->info=reco2->info;
      reco2->info=aux;
   }
}
```

```
int mayor()
{
   if(!vacia())
   {
      int may=raiz->info;
      struct nodo *reco=raiz->sig;
      while (reco!=NULL)
      {
         if (reco->info>may)
         {
            may=reco->info;
         }
         reco=reco->sig;
      }
      return may;
   }
   else
      return -1;
}
```

```
int posMayor()
{
   if(!vacia())
   {
      int may=raiz->info;
      int x = 1;
      int pos = x;
      struct nodo *reco=raiz->sig;
      while (reco!=NULL)
      {
         if (reco->info>may)
         {
            may=reco->info;
            pos = x;
         }
         reco=reco->sig;
         x++;
      }
      return pos;
   }
   else
      return -1;
}
```

```
int ordenada()
{
   if (cantidad()>1)
   {
      struct nodo *reco1 = raiz;
      struct nodo *reco2 =raiz->sig;
      while (reco2 != NULL)
      {
         if (reco2->info<reco1->info)
         {
            return 0;
         }
         reco2 = reco2->sig;
         reco1 = reco1->sig;
      }
   }
   return 1;
}
```

```
int existe(int x)
{
   struct nodo *reco = raiz;
   while (reco != NULL)
   {
      if(reco->info == x)
         return 1;
      reco = reco->sig;
   }
   return 0;
}
```

```
int main()
{
   insertar(1,10);
   insertar(2,20);
   insertar(3,30);
   imprimir();
   insertar(2,15);
   insertar(1,115);
   imprimir();
   printf("Extraemos el de la segunda posicion: %i", extraer(2));
   borrar(2);
   imprimir();
   intercambiar(1,2);
   imprimir();
   printf("Mayor de la lista: %i\n", mayor());
   printf("Posicion del mayor de la lista: %i\n", posMayor());
   if (ordenada())
```

```
        printf("La lista esta ordenada de menor a mayor\n");
    else
        printf("La lista no esta ordenada de menor a mayor\n");
    if (existe(115))
    {
        printf("El numero existe");
    }
    else
    {
        printf("El numero no existe");
    }
    liberar();
    getch();
    return 0;
}
```

Capítulo 166.- Estructuras dinámicas en C: Listas genéricas – 2

Problema propuesto

Plantear un programa para administrar una lista genérica implementando las siguientes funciones:

a) Insertar un nodo al principio de la lista.
b) Insertar un nodo al final de la lista.
c) Insertar un nodo en la segunda posición. Si la lista está vacía no se inserta el nodo.
d) Insertar un nodo en la ante última posición.
e) Borrar el primer nodo.
f) Borrar el segundo nodo.
g) Borrar el último nodo.
h) Borrar el nodo con la información mayor.

Para la realización de este ejercicio vamos a iniciar copiando parte del código del ejercicio realizado en el capítulo anterior:

```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<stdlib.h>
4   #include<string.h>
5
6   struct nodo {
7       int info;
8       struct nodo *sig;
9   };
10
11  struct nodo *raiz=NULL;
12
13  void liberar()
14  {
15      struct nodo *reco=raiz;
16      struct nodo *bor;
17      while(reco!=NULL)
18      {
19          bor=reco;
20          reco=reco->sig;
21          free(bor);
22      }
23  }
24
25  int vacia()
26  {
27      if(raiz==NULL)
28          return 1;
29      else
30          return 0;
31  }
32
```

```
33
34  int cantidad()
35  {
36      int cant=0;
37      struct nodo *reco=raiz;
38      while(reco!=NULL)
39      {
40          cant++;
41          reco=reco->sig;
42      }
43      return cant;
44  }
45
46  void imprimir()
47  {
48      struct nodo *reco=raiz;
49      printf("Listado completo\n");
50      while(reco!=NULL)
51      {
52         printf(" %i ", reco->info);
53         reco=reco->sig;
54      }
55      printf("\n");
56  }
```

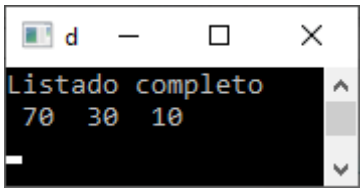
```
68  int main()
69  {
70      getch();
71      return 0;
72  }
73
```

Solución:

a) Insertar un nodo al principio de la lista.

```
58  void insertarPrimero(int x)
59  {
60      struct nodo *nuevo;
61      nuevo=malloc(sizeof(struct nodo));
62      nuevo->info=x;
63      nuevo->sig = raiz;
64      raiz = nuevo;
65  }
66
67  int main()
68  {
69      insertarPrimero(10);
70      insertarPrimero(30);
71      insertarPrimero(70);
72      imprimir();
73      getch();
74      return 0;
75  }
```

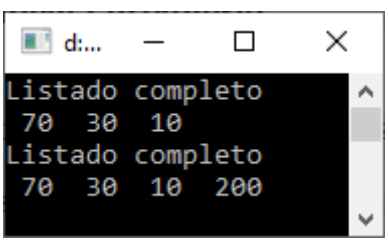
Si ejecutamos este será el resultado:



b) Insertar un nodo al final de la lista.

```
void insertarFinal(int x)
{
    struct nodo *nuevo;
    nuevo=malloc(sizeof(struct nodo));
    nuevo->info=x;
    nuevo->sig=NULL;
    if(raiz==NULL)
    {
        raiz=nuevo;
    }
    else
    {
        struct nodo *reco=raiz;
        while (reco->sig!=NULL)
        {
            reco=reco->sig;
        }
        reco->sig=nuevo;
    }
}

int main()
{
    insertarPrimero(10);
    insertarPrimero(30);
    insertarPrimero(70);
    imprimir();
    insertarFinal(200);
    imprimir();
    liberar();
    getch();
    return 0;
}
```

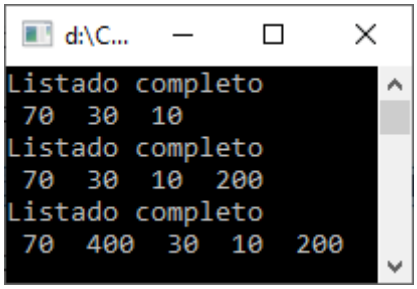
Si ejecutamos este será el resultado:



c) Insertar un nodo en la segunda posición. Si la lista está vacía no se inserta el nodo.

```
void insertarSegundo(int x)
{
    if (raiz!=NULL)
    {
        struct nodo *nuevo;
        nuevo=malloc(sizeof(struct nodo));
        nuevo->info=x;
        if (raiz->sig==NULL)
        {
            raiz->sig=nuevo;
            nuevo->sig=NULL;
        }
        else
        {
            nuevo->sig=raiz->sig;
            raiz->sig=nuevo;
        }
    }
}

int main()
{
    insertarPrimero(10);
    insertarPrimero(30);
    insertarPrimero(70);
    imprimir();
    insertarFinal(200);
    imprimir();
    insertarSegundo(400);
    imprimir();
    liberar();
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:

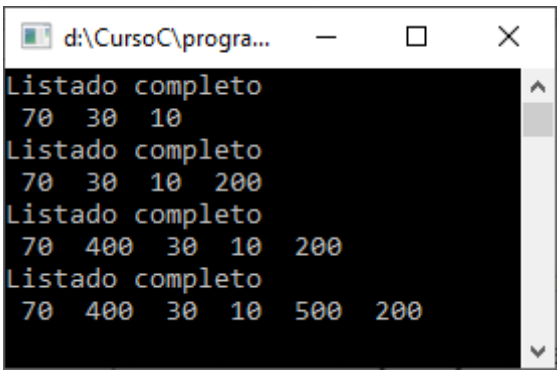


d) Insertar un nodo en la ante última posición.

```
void insertarAnteUltimo(int x)
{
    if(raiz!=NULL)
    {
        struct nodo *nuevo;
```

```
        nuevo=malloc(sizeof(struct nodo));
        nuevo->info=x;
        if (raiz->sig==NULL)
        {
            nuevo->sig=raiz;
            raiz=nuevo;
        }
        else
        {
            struct nodo *atras=raiz;
            struct nodo *reco=raiz->sig;
            while(reco->sig!=NULL)
            {
                atras=reco;
                reco=reco->sig;
            }
            nuevo->sig=atras->sig;
            atras->sig=nuevo;
        }
    }
}

int main()
{
    insertarPrimero(10);
    insertarPrimero(30);
    insertarPrimero(70);
    imprimir();
    insertarFinal(200);
    imprimir();
    insertarSegundo(400);
    imprimir();
    insertarAnteUltimo(500);
    imprimir();
    liberar();
    getch();
    return 0;
}
```

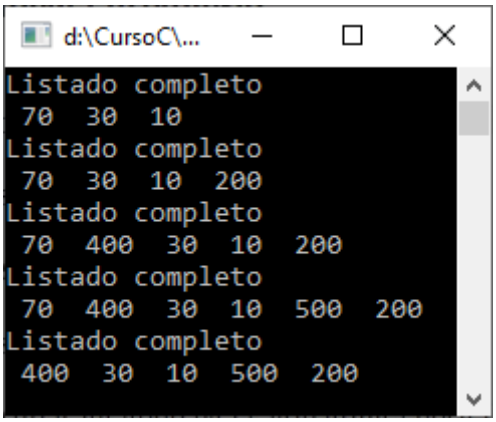
Si ejecutamos este será el resultado:



e) Borrar el primer nodo.

```
void borrarPrimero()
{
     if(raiz!=NULL)
     {
          struct nodo *bor=raiz;
          raiz=raiz->sig;
          free(bor);
     }
}

int main()
{
     insertarPrimero(10);
     insertarPrimero(30);
     insertarPrimero(70);
     imprimir();
     insertarFinal(200);
     imprimir();
     insertarSegundo(400);
     imprimir();
     insertarAnteUltimo(500);
     imprimir();
     borrarPrimero();
     imprimir();
     liberar();
     getch();
     return 0;
}
```

Si ejecutamos este será el resultado:

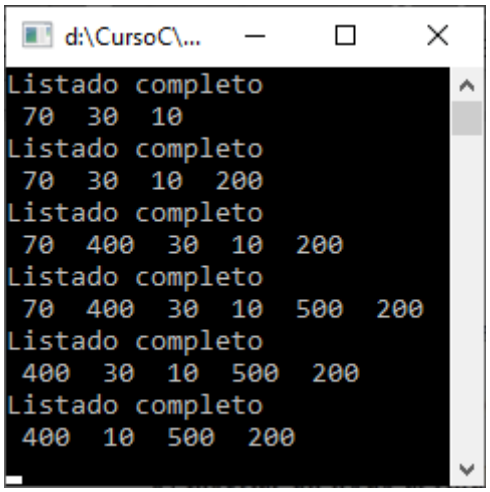


f) Borrar el segundo nodo.

```
void borrarSegundo()
{
     if(raiz!=NULL)
     {
          if(raiz->sig!=NULL)
          {
               struct nodo *bor=raiz->sig;
               struct nodo *tercero=raiz->sig;
```

```
            tercero=tercero->sig;
            raiz->sig=tercero;
            free(bor);
        }

    }
}

int main()
{
    insertarPrimero(10);
    insertarPrimero(30);
    insertarPrimero(70);
    imprimir();
    insertarFinal(200);
    imprimir();
    insertarSegundo(400);
    imprimir();
    insertarAnteUltimo(500);
    imprimir();
    borrarPrimero();
    imprimir();
    borrarSegundo();
    imprimir();
    liberar();
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:

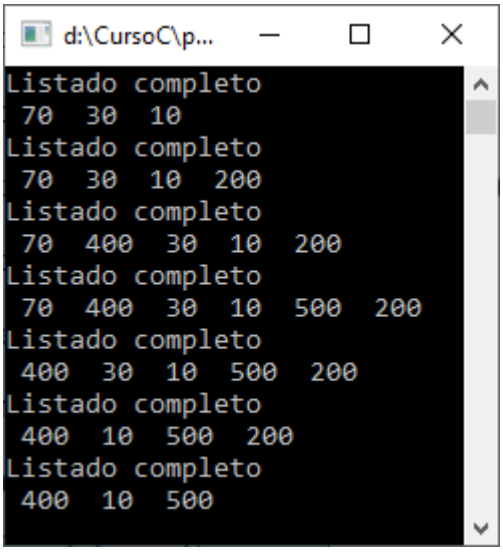


g) Borrar el último nodo.

```
void borrarUltimo()
{
    if(raiz!=NULL)
    {
        if(raiz->sig==NULL)
        {
            free(raiz);
```

```
            raiz=NULL;
        }
        else
        {
            struct nodo *reco=raiz->sig;
            struct nodo *atras=raiz;
            while (reco->sig!=NULL)
            {
                atras=reco;
                reco=reco->sig;
            }
            atras->sig=NULL;
            free(reco);
        }
    }
}
int main()
{
    insertarPrimero(10);
    insertarPrimero(30);
    insertarPrimero(70);
    imprimir();
    insertarFinal(200);
    imprimir();
    insertarSegundo(400);
    imprimir();
    insertarAnteUltimo(500);
    imprimir();
    borrarPrimero();
    imprimir();
    borrarSegundo();
    imprimir();
    borrarUltimo();
    imprimir();
    liberar();
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:

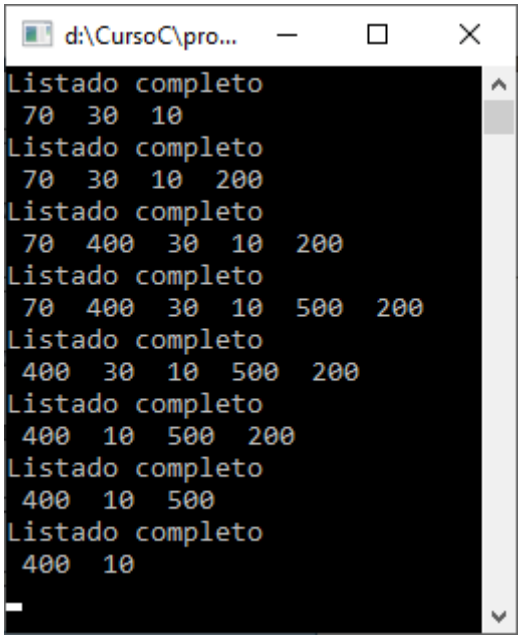


h) Borrar el nodo con la información mayor.

```
void borrarMayor()
{
    if (raiz!=NULL)
    {
        struct nodo *reco=raiz;
        int mayor=raiz->info;
        while(reco!=NULL)
        {
            if(reco->info>mayor)
            {
                mayor=reco->info;
            }
            reco=reco->sig;
        }
        reco=raiz;
        struct nodo *atras=raiz;
        struct nodo *bor;
        while(reco!=NULL)
        {
            if(reco->info==mayor)
            {
                if(reco==raiz)
                {
                    bor=raiz;
                    raiz=raiz->sig;
                    atras=raiz;
                    reco=raiz;
                    free(bor);
                }
                else
                {
                    atras->sig=reco->sig;
                    bor=reco;
                    reco=reco->sig;
                    free(bor);
                }
            }
            else
            {
                atras=reco;
                reco=reco->sig;
            }
        }
    }
}
```

```c
int main()
{
    insertarPrimero(10);
    insertarPrimero(30);
    insertarPrimero(70);
    imprimir();
    insertarFinal(200);
    imprimir();
    insertarSegundo(400);
    imprimir();
    insertarAnteUltimo(500);
    imprimir();
    borrarPrimero();
    imprimir();
    borrarSegundo();
    imprimir();
    borrarUltimo();
    imprimir();
    borrarMayor();
    imprimir();
    liberar();
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:

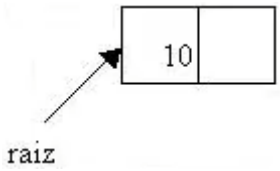


Capítulo 167.- Estructuras dinámicas en C: Listas genéricas ordenadas – 1

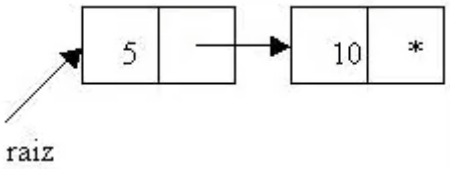
Una lista genérica es ordenada si cuando insertamos información en la lista queda ordenada respecto al campo info (sea de menor a mayor o a la inversa).

Ejemplo:

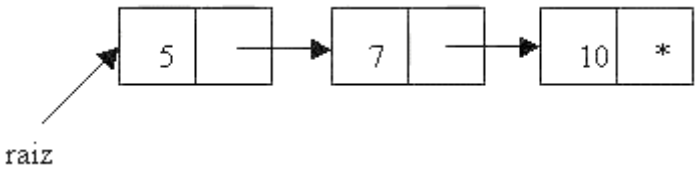
```
insertar(10);
```



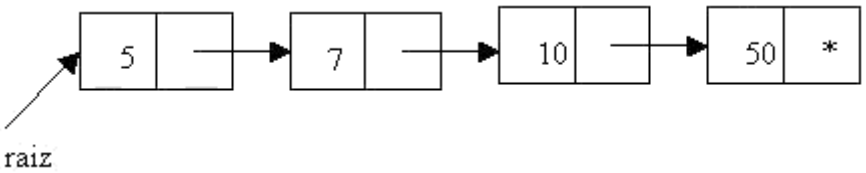

```
insertar(5)
```




```
insertar(7)
```




```
insertar(50)
```



Podemos observar que si recorremos la lista podemos acceder a la información de menor a mayor.

No requiere una función para ordenar la lista, sino que siempre permanece ordenada, ya que se inserta ordenada.

A continuación se adjunta parte del codigo que ya comentamos en los capítulos anteriores:

```c
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

truct nodo {
    int info;
    struct nodo *sig;
};

struct nodo *raiz=NULL;

void liberar()
{
    struct nodo *reco=raiz;
    struct nodo *bor;
    while(reco!=NULL)
    {
        bor=reco;
        reco=reco->sig;
        free(bor);
    }
}

int vacia()
{
    if(raiz==NULL)
        return 1;
    else
        return 0;
}

void imprimir()
{
    struct nodo *reco=raiz;
    printf("Listado completo\n");
    while(reco!=NULL)
    {
       printf(" %i ", reco->info);
       reco=reco->sig;
    }
    printf("\n");
}
```

Vamos a crear el método insertar:

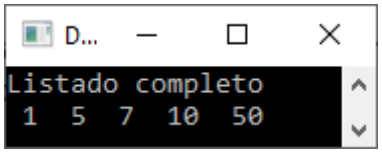
```c
void insertar(int x)
{
    struct nodo *nuevo;
    nuevo=malloc(sizeof(struct nodo));
    nuevo->info=x;
    nuevo->sig=NULL;
    if(raiz==NULL)
        raiz=nuevo;
    else
    {
```

```
        if(x<raiz->info)
        {
            nuevo->sig=raiz;
            raiz=nuevo;
        }
        else
        {
            struct nodo *reco=raiz;
            struct nodo *atras=raiz;
            while(x>=reco->info && reco->sig!=NULL)
            {
                atras=reco;
                reco=reco->sig;
            }
            if(x>=reco->info)
            {
                reco->sig=nuevo;
            }
            else
            {
                nuevo->sig=reco;
                atras->sig=nuevo;
            }
        }

    }
}
```

Ahora vamos a crear el método main:

```
int main()
{
    insertar(10);
    insertar(5);
    insertar(7);
    insertar(50);
    insertar(1);
    imprimir();
    liberar();
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 168.- Estructuras dinámicas en C: Listas genéricas ordenadas – 2

Problema propuesto

Se tiene la siguiente declaración de nodo:

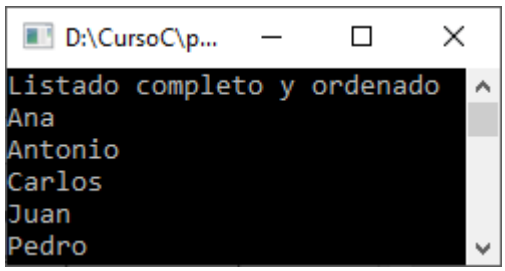
```
struct nodo {
    char usuario[51];
    struct nodo *sig;
};
```

Implementar una lista ordenada con respecto al campo usuario.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>

struct nodo {
    char usuario[51];
    struct nodo *sig;
};

struct nodo *raiz=NULL;

void liberar()
{
    struct nodo *reco=raiz;
    struct nodo *bor;
    while(reco!=NULL)
    {
        bor=reco;
        reco=reco->sig;
        free(bor);
    }
}

void imprimir()
{
    struct nodo *reco=raiz;
    printf("Listado completo y ordenado\n");
    while(reco!=NULL)
    {
        printf("%s \n", reco->usuario);
        reco=reco->sig;
    }
    printf("\n");
}
```

```
36
37  void insertar(char nombre[51])
38  {
39      struct nodo *nuevo;
40      nuevo=malloc(sizeof(struct nodo));
41      strcpy(nuevo->usuario,nombre);
42      nuevo->sig=NULL;
43      if(raiz==NULL)
44      {
45          raiz=nuevo;
46      }
47      else
48      {
49          if(strcmp(nombre, raiz->usuario)<0)
50          {
51              nuevo->sig=raiz;
52              raiz=nuevo;
53          }
54          else
55          {
56              struct nodo *reco=raiz;
57              struct nodo *atras=raiz;
58              while (strcmp(nombre, reco->usuario)>0 && reco->sig!=NULL)
59              {
60                  atras=reco;
61                  reco=reco->sig;
62              }
63              if(strcmp(nombre, reco->usuario)>0)
64              {
65                  reco->sig=nuevo;
66              }
67              else
68              {
69                  nuevo->sig=reco;
70                  atras->sig=nuevo;
71              }
72          }
73      }
74  }
75
76  int main()
77  {
78      insertar("Juan");
79      insertar("Carlos");
80      insertar("Antonio");
81      insertar("Pedro");
82      insertar("Ana");
83      imprimir();
84      liberar();
85      getch();
86      return 0;
87  }
```

Si ejecutamos este será el resultado:



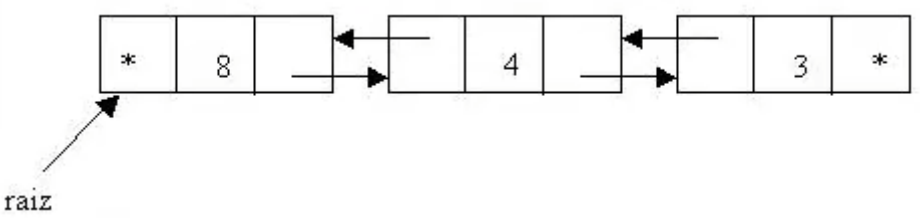
Capítulo 169.- Estructuras dinámicas en C: Listas genéricas doblemente encadenadas – 1

A las listas vistas hasta ahora el momento podemos correrlas solamente en una dirección (Listas simplemente encadenadas).

Hay problemas donde se requiere la lista en ambas direcciones, en estos casos el empleo de listas doblemente encadenadas es recomendable.

Como ejemplo pensamos que debemos almacenar un menú de opciones en una lista, la opción a seleccionar puede ser la siguiente o la anterior, podemos desplazarnos en ambas direcciones.

Representación gráfica de una lista doblemente encadenada:



Observemos que una lista doblemente encadenada tiene dos punteros por cada nodo, uno apunta al nodo siguiente y otro al nodo anterior.

Seguimos teniendo un puntero (raiz) que tiene la dirección del primer nodo de la lista.

El puntero sig del último nodo igual que las listas simplemente encadenadas apuntan a NULL, y el puntero anterior del primer nodo apunta a NULL.

Se puede plantear Listas de tipo pila, cola y genéricas con enlace doble.

Hay que tener en cuenta que el requerimiento de memoria es mayor en las listas doblemente encadenadas ya que tenemos dos punteros por nodo.

La estructura de nodo es:

```
struct nodo {
    int info;
    struct nodo *sig;
    struct nodo *ant;
};
```

Resolveremos algunas funciones para administrar listas genéricas empleando listas doblemente encadenadas para analizar la mecánica de enlace de nodos.

Muchas de las funciones, para listas simple y doblemente encadenadas no varía, como por ejemplo: el vacía, cantidad, liberar, etc.

```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<stdlib.h>
4
```

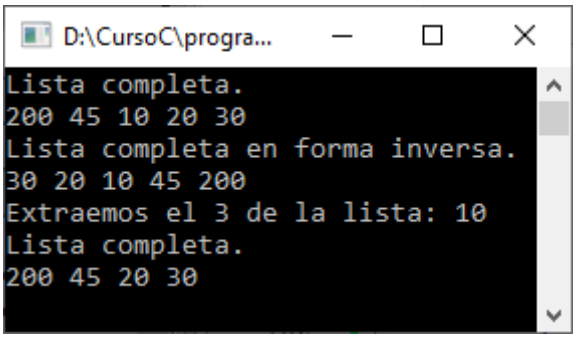
```c
struct nodo{
    int info;
    struct nodo *sig;
    struct nodo *ant;
};

struct nodo *raiz=NULL;

void liberar()
{
    struct nodo *reco = raiz;
    struct nodo *bor;
    while (reco!=NULL)
    {
        bor=reco;
        reco=reco->sig;
        free(bor);
    }
}

int vacia()
{
    if (raiz==NULL)
        return 1;
    else
        return 0;
}

int cantidad()
{
    struct nodo *reco=raiz;
    int cant=0;
    while (reco != NULL)
    {
        cant++;
        reco  = reco->sig;
    }
    return cant;
}

void imprimir()
{
    struct nodo *reco=raiz;
    printf("Lista completa.\n");
    while(reco!=NULL)
    {
        printf("%i ", reco->info);
        reco=reco->sig;
    }
    printf("\n");
}
```

```
57  void insertar(int pos, int x)
58  {
59      if(pos<=cantidad()+1)
60      {
61          struct nodo *nuevo;
62          nuevo=malloc(sizeof(struct nodo));
63          nuevo->info=x;
64          nuevo->ant=NULL;
65          nuevo->sig=NULL;
66          if (pos==1)
67          {
68              nuevo->sig=raiz;
69              if (raiz!=NULL)
70              {
71                  raiz->ant=nuevo;
72              }
73              raiz=nuevo;
74          }
75          else
76          {
77              if(pos==cantidad()+1)
78              {
79                  struct nodo *reco=raiz;
80                  while(reco->sig!=NULL)
81                  {
82                      reco=reco->sig;
83                  }
84                  reco->sig=nuevo;
85                  nuevo->ant=reco;
86              }
87              else
88              {
89                  struct nodo *reco=raiz;
90                  for(int f=1; f<=pos-2; f++)
91                  {
92                      reco=reco->sig;
93                  }
94                  struct nodo *siguiente=reco->sig;
95                  reco->sig=nuevo;
96                  nuevo->ant=reco;
97                  nuevo->sig=siguiente;
98                  siguiente->ant=nuevo;
99              }
100         }
101     }
102 }
```

```
103
104  void imprimirInverso()
105  {
106      if (raiz!=NULL)
107      {
108          struct nodo *reco=raiz;
109          while(reco->sig!=NULL)
110          {
111              reco=reco->sig;
112          }
113          printf("Lista completa en forma inversa.\n");
114          while (reco!=NULL)
115          {
116              printf("%i ", reco->info);
117              reco=reco->ant;
118          }
119          printf("\n");
120      }
121  }
122
123  int extraer(int pos)
124  {
125      if (pos<=cantidad())
126      {
127          int informacion;
128          struct nodo *bor;
129          if (pos==1)
130          {
131              informacion=raiz->info;
132              bor=raiz;
133              raiz=raiz->sig;
134              if(raiz!=NULL)
135              {
136                  raiz->ant=NULL;
137              }
138          }
139          else
140          {
141              struct nodo *reco=raiz;
142              for(int f=1; f<=pos-2; f++)
143              {
144                  reco=reco->sig;
145              }
146              struct nodo *prox=reco->sig;
147              bor=prox;
148              reco->sig=prox->sig;
149              struct nodo *siguiente=prox->sig;
150              if(siguiente!=NULL)
151              {
152                  siguiente->ant=reco;
153              }
154              informacion=prox->info;
```

```
155             }
156             free(bor);
157             return informacion;
158         }
159         else
160             return -1;
161     }
162
163     int main()
164     {
165         insertar(1,10);
166         insertar(2,20);
167         insertar(3,30);
168         insertar(1,200);
169         insertar(2, 45);
170         imprimir();
171         imprimirInverso();
172         printf("Extraemos el 3 de la lista: %i\n", extraer(3));
173         imprimir();
174         liberar();
175         getch();
176         return;
177     }
178
```

Si ejecutamos este será el resultado:



Capítulo 170.- Estructuras dinámicas en C: Listas genéricas doblemente encadenadas – 2

Problema propuesto

Plantear un programa para administrar una lista genérica doblemente encadenada implementando las siguientes funciones:

a) Insertar un nodo al principio de la lista.
b) Insertar un nodo al final de la lista.
c) Insertar un nodo en la segunda posición.
d) Insertar un nodo en la anteúltima posición.
e) Borrar el primer nodo.
f) Borrar el segundo nodo.
g) Borrar el último nodo.
h) Borrar el nodo con información mayor.

Los método básicos que hicimos en el capítulo anterior los vamos a reutilizar:

```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<stdlib.h>
4
5   struct nodo {
6       int info;
7       struct nodo *sig;
8       struct nodo *ant;
9   };
10
11  struct nodo *raiz=NULL;
12
13  void liberar()
14  {
15      struct nodo *reco = raiz;
16      struct nodo *bor;
17      while (reco!=NULL)
18      {
19          bor = reco;
20          reco = reco->sig;
21          free(bor);
22      }
23  }
24
25  int vacia()
26  {
27      if(raiz==NULL)
28          return 1;
29      else
30          return 0;
31  }
32
```

```
33  int cantidad()
34  {
35      struct nodo *reco = raiz;
36      int cant=0;
37      while (reco!=NULL)
38      {
39          cant++;
40          reco=reco->sig;
41      }
42  }
43
44  void imprimir()
45  {
46      struct nodo *reco=raiz;
47      printf("!Lista completa.\n");
48      while (reco!=NULL)
49      {
50          printf("%i ", reco->info);
51          reco=reco->sig;
52      }
53      printf("\n");
54  }
55
56  int main()
57  {
58
59      liberar();
60      getch();
61      return 0;
62  }
```

a) Insertar un nodo al principio de la lista.

```
56  void insertarPrimero(int x)
57  {
58      struct nodo *nuevo;
59      nuevo=malloc(sizeof(struct nodo));
60      nuevo->info=x;
61      nuevo->ant=NULL;
62      nuevo->sig=raiz;
63      if(raiz!=NULL)
64      {
65          raiz->ant=nuevo;
66      }
67      raiz=nuevo;
68  }
69
70  int main()
71  {
72      insertarPrimero(20);
73      insertarPrimero(10);
74      insertarPrimero(5);
75      insertarPrimero(70);
```

```
76      insertarPrimero(1);
77      imprimir();
78      liberar();
79      getch();
80      return 0;
81  }
```

b) Insertar un nodo al final de la lista.

```
70  void insertarUltimo(int x)
71  {
72      struct nodo *nuevo;
73      nuevo=malloc(sizeof(struct nodo));
74      nuevo->info=x;
75      nuevo->ant=NULL;
76      nuevo->sig=NULL;
77      if (vacia())
78      {
79          raiz=nuevo;
80      }
81      else
82      {
83          struct nodo *reco=raiz;
84          while(reco->sig!=NULL)
85          {
86              reco=reco->sig;
87          }
88          reco->sig=nuevo;
89          nuevo->ant=reco;
90      }
91  }
92
93  int main()
94  {
95      insertarPrimero(20);
96      insertarPrimero(10);
97      insertarPrimero(5);
98      insertarPrimero(70);
99      insertarPrimero(1);
100     imprimir();
101     insertarUltimo(400);
102     imprimir();
103     liberar();
104     getch();
105     return 0;
106 }
```

c) Insertar un nodo en la segunda posición. Si la lista está vacía no se inserta el nodo.

```
93  void insertarSegundo(int x)
94  {
95      if(raiz!=NULL)
96      {
97          struct nodo *nuevo;
98          nuevo=malloc(sizeof(struct nodo));
```

```
 99            nuevo->info=x;
100            nuevo->ant=NULL;
101            nuevo->sig=NULL;
102            if (raiz->sig==NULL)
103            {
104                raiz->sig=nuevo;
105                nuevo->ant=raiz;
106            }
107            else
108            {
109                struct nodo *segundo=raiz->sig;
110                nuevo->sig=segundo;
111                nuevo->ant=raiz;
112                raiz->sig=nuevo;
113                segundo->ant=nuevo;
114            }
115        }
116    }
117
118    int main()
119    {
120        insertarPrimero(20);
121        insertarPrimero(10);
122        insertarPrimero(5);
123        insertarPrimero(70);
124        insertarPrimero(1);
125        imprimir();
126        insertarUltimo(400);
127        imprimir();
128        insertarSegundo(9);
129        imprimir();
130        liberar();
131        getch();
132        return 0;
133    }
```

d) Insertar un nodo en la anteúltima posición.

```
118    void insertarAnteUltimo(int x)
119    {
120        if (raiz!=NULL)
121        {
122            struct nodo *nuevo;
123            nuevo=malloc(sizeof(struct nodo));
124            nuevo->info= x;
125            nuevo->info=x;
126            nuevo->ant=NULL;
127            nuevo->sig=NULL;
128            if (raiz->sig==NULL)
129            {
130                nuevo->sig=raiz;
131                raiz->ant=nuevo;
132                raiz=nuevo;
133            }
```

```
134             else
135             {
136                 struct nodo *reco=raiz;
137                 while(reco->sig!=NULL)
138                 {
139                     reco=reco->sig;
140                 }
141                 struct nodo *anteultimo=reco->ant;
142                 anteultimo->sig=nuevo;
143                 nuevo->ant=anteultimo;
144                 nuevo->sig=reco;
145                 reco->ant=nuevo;
146             }
147         }
148     }
149
150     int main()
151     {
152         insertarPrimero(20);
153         insertarPrimero(10);
154         insertarPrimero(5);
155         insertarPrimero(70);
156         insertarPrimero(1);
157         imprimir();
158         insertarUltimo(400);
159         imprimir();
160         insertarSegundo(9);
161         imprimir();
162         insertarAnteUltimo(1000);
163         imprimir();
164         liberar();
165         getch();
166         return 0;
167     }
```

e) Borrar el primer nodo

```
150     void borrarPrimero()
151     {
152         if(raiz!=NULL)
153         {
154             struct nodo *bor=raiz;
155             raiz=raiz->sig;
156             if (raiz!=NULL)
157             {
158                 raiz->ant=NULL;
159             }
160             free(bor);
161         }
162     }
163
```

```
164  int main()
165  {
166      insertarPrimero(20);
167      insertarPrimero(10);
168      insertarPrimero(5);
169      insertarPrimero(70);
170      insertarPrimero(1);
171      imprimir();
172      insertarUltimo(400);
173      imprimir();
174      insertarSegundo(9);
175      imprimir();
176      insertarAnteUltimo(1000);
177      imprimir();
178      borrarPrimero();
179      imprimir();
180      liberar();
181      getch();
182      return 0;
183  }
```

f) Borrar el segundo nodo.

```
164  void borrarSegundo()
165  {
166      if(raiz!=NULL)
167      {
168          if(raiz->sig!=NULL)
169          {
170              struct nodo *bor=raiz->sig;
171              struct nodo *tercero=raiz->sig;
172              tercero=tercero->sig;
173              raiz->sig=tercero;
174              if(tercero!=NULL)
175              {
176                  tercero->ant=raiz;
177              }
178              free(bor);
179          }
180      }
181  }
```

```
182
183  int main()
184  {
185      insertarPrimero(20);
186      insertarPrimero(10);
187      insertarPrimero(5);
188      insertarPrimero(70);
189      insertarPrimero(1);
190      imprimir();
191      insertarUltimo(400);
192      imprimir();
193      insertarSegundo(9);
194      imprimir();
```

```
195         insertarAnteUltimo(1000);
196         imprimir();
197         borrarPrimero();
198         imprimir();
199         borrarSegundo();
200         imprimir();
201         liberar();
202         getch();
203         return 0;
204     }
```

g) Borrar el último nodo.

```
183 void borrarUltimo()
184 {
185     if(raiz!=NULL)
186     {
187         if(raiz->sig==NULL)
188         {
189             free(raiz);
190             raiz=NULL;
191         }
192         else
193         {
194             struct nodo *reco=raiz;
195             while (reco->sig!=NULL)
196             {
197                 reco=reco->sig;
198             }
199             struct nodo *ante=reco->ant;
200             ante->sig=NULL;
201             free(reco);
202         }
203     }
204 }
205
206 int main()
207 {
208     insertarPrimero(20);
209     insertarPrimero(10);
210     insertarPrimero(5);
211     insertarPrimero(70);
212     insertarPrimero(1);
213     imprimir();
214     insertarUltimo(400);
215     imprimir();
216     insertarSegundo(9);
217     imprimir();
218     insertarAnteUltimo(1000);
219     imprimir();
220     borrarPrimero();
221     imprimir();
222     borrarSegundo();
223     imprimir();
```

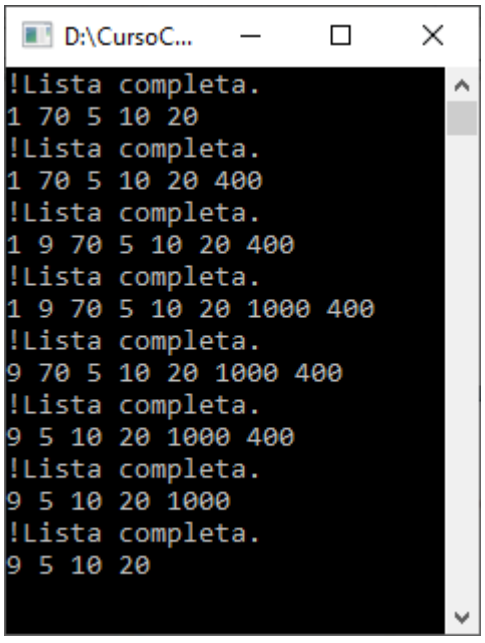
```
224         borrarUltimo();
225         imprimir();
226         liberar();
227         getch();
228         return 0;
229     }
```

h) Borrar el nodo con información mayor.

```
206 void borrarMayor()
207 {
208     if(raiz!=NULL)
209     {
210         struct nodo *reco=raiz;
211         int may=raiz->info;
212         while(reco!=NULL)
213         {
214             if(reco->info>may)
215             {
216                 may=reco->info;
217             }
218             reco=reco->sig;
219         }
220         reco=raiz;
221         struct nodo *bor;
222         while (reco!=NULL)
223         {
224             if(reco->info==may)
225             {
226                 if(reco==raiz)
227                 {
228                     bor=raiz;
229                     raiz=raiz->sig;
230                     if(raiz!=NULL)
231                     {
232                         raiz->ant=NULL;
233                     }
234                     free(bor);
235                     return;
236                 }
237                 else
238                 {
239                     if(reco->sig==NULL)
240                     {
241                         bor=reco;
242                         reco=reco->ant;
243                         reco->sig=NULL;
244                         free(bor);
245                         return;
```

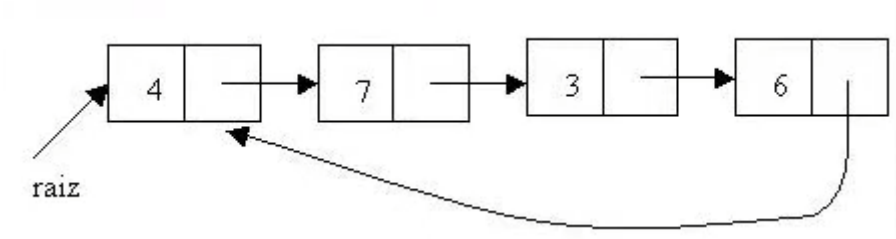
```
246                         }
247                         else
248                         {
249                             struct nodo *ante=reco->ant;
250                             bor=reco;
251                             reco=reco->sig;
252                             ante->sig=reco;
253                             reco->ant=ante;
254                             free(bor);
255                             return;
256                         }
257                     }
258                 }
259                 else
260                 {
261                     reco=reco->sig;
262                 }
263             }
264         }
265 }
266
267 int main()
268 {
269     insertarPrimero(20);
270     insertarPrimero(10);
271     insertarPrimero(5);
272     insertarPrimero(70);
273     insertarPrimero(1);
274     imprimir();
275     insertarUltimo(400);
276     imprimir();
277     insertarSegundo(9);
278     imprimir();
279     insertarAnteUltimo(1000);
280     imprimir();
281     borrarPrimero();
282     imprimir();
283     borrarSegundo();
284     imprimir();
285     borrarUltimo();
286     imprimir();
287     borrarMayor();
288     imprimir();
289     liberar();
290     getch();
291     return 0;
292 }
```

Cuando ejecutemos este será el resultado:



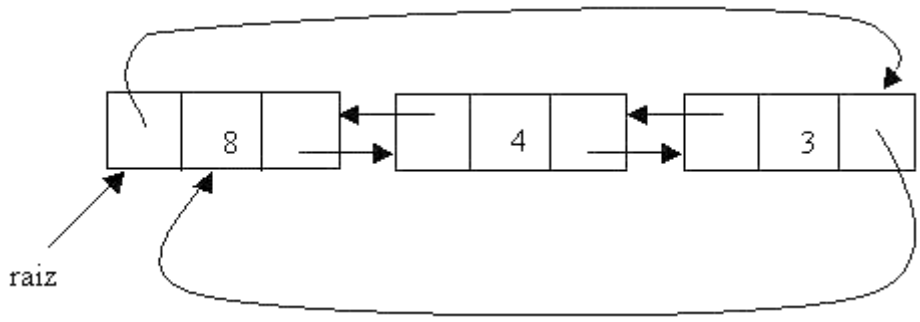
Capítulo 171.- Estructuras dinámicas en C: Listas genéricas circulares

Una lista circular simplemente encadenada la podemos representar gráficamente:



Observemos que el puntero sig del último nodo apunta al primer nodo. En este tipo de listas si avanzamos raiz no perdemos la referencia al nodo anterior ya que es un círculo.

Una lista circula puede también ser doblemente encadenada.



El puntero ant del primer nodo apunta al último nodo de la lista y el puntero sig del último nodo de la lista apunta al primero.

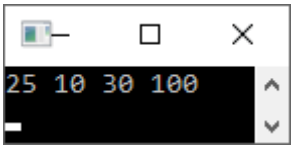
Resolveremos algunos métodos para administrar listas genéricas circulares encadenadas para analizar la mecánica de enlace de nodos.

```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<stdlib.h>
4
5   struct nodo {
6       int info;
7       struct nodo *sig;
8       struct nodo *ant;
9   };
10
11  struct nodo *raiz=NULL;
12
13  void liberar()
14  {
15      if(raiz!=NULL)
16      {
17          struct nodo *reco=raiz->sig;
18          struct nodo *bor;
19          while (reco!=raiz)
20          {
```

```
21              bor=reco;
22              reco=reco->sig;
23              free(bor);
24          }
25          free(raiz);
26      }
27  }
28
29  int vacia()
30  {
31      if(raiz==NULL)
32          return 1;
33      else
34          return 0;
35  }
36
37  void insertarPrimero(int x)
38  {
39      struct nodo *nuevo;
40      nuevo=malloc(sizeof(struct nodo));
41      nuevo->info=x;
42      if(vacia())
43      {
44          nuevo->sig=nuevo;
45          nuevo->ant=nuevo;
46          raiz=nuevo;
47      }
48      else
49      {
50          struct nodo *ultimo=raiz->ant;
51          nuevo->sig=raiz;
52          nuevo->ant=ultimo;
53          ultimo->sig=nuevo;
54          raiz->ant=nuevo;
55          raiz=nuevo;
56      }
57  }
58
59  void imprimir()
60  {
61      if(!vacia())
62      {
63          struct nodo *reco=raiz;
64          do{
65              printf("%i ", reco->info);
66              reco=reco->sig;
67          }while (reco!=raiz);
68          printf("\n");
69      }
70  }
```

```
71
72  int main()
73  {
74      insertarPrimero(100);
75      insertarPrimero(30);
76      insertarPrimero(10);
77      insertarPrimero(25);
78      imprimir();
79      liberar();
80      getch();
81      return 0;
82  }
```

Si ejecutamos este será el resultado:

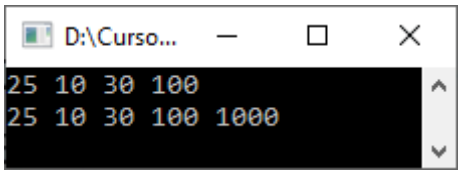


Insertar por la última posición:

```
72  void insertarUlitmo(int x)
73  {
74      struct nodo *nuevo;
75      nuevo=malloc(sizeof(struct nodo));
76      nuevo->info=x;
77      if(vacia())
78      {
79          nuevo->sig=nuevo;
80          nuevo->ant=nuevo;
81          raiz=nuevo;
82      }
83      else
84      {
85          struct nodo *ultimo=raiz->ant;
86          nuevo->sig=raiz;
87          nuevo->ant=ultimo;
88          ultimo->sig=nuevo;
89          raiz->ant=nuevo;
90      }
91  }
92
93
94  int main()
95  {
96      insertarPrimero(100);
97      insertarPrimero(30);
98      insertarPrimero(10);
99      insertarPrimero(25);
100     imprimir();
101     insertarUlitmo(1000);
```

```
102         imprimir();
103         liberar();
104         getch();
105         return 0;
106     }
```

Si ejecutamos este será el resultado:



Queremos retornar la cantidad de nodos:

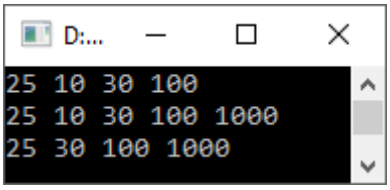
```
72  int cantidad()
73  {
74      int cant=0;
75      if (!vacia())
76      {
77          struct nodo *reco=raiz;
78          do{
79              cant++;
80              reco=reco->sig;
81          }while (reco!=raiz);
82      }
83      return cant;
84  }
```

Ahora queremos borrar un nodo en una posición determinada:

```
107 void borrar(int pos)
108 {
109     if (pos<=cantidad())
110     {
111         if(pos==1)
112         {
113             if(cantidad()==1)
114             {
115                 free(raiz);
116                 raiz=NULL;
117             }
118             else
119             {
120                 struct nodo *bor=raiz;
121                 struct nodo *ultimo=raiz->ant;
122                 raiz=raiz->sig;
123                 ultimo->sig=raiz;
124                 raiz->ant=ultimo;
125                 free(bor);
126             }
127         }
128         else
129         {
130             struct nodo *reco=raiz;
```

```
131                 for(int f=1; f<=pos-1; f++)
132                 {
133                     reco=reco->sig;
134                 }
135                 struct nodo *bor=reco;
136                 struct nodo *anterior=reco->ant;
137                 reco=reco->sig;
138                 anterior->sig=reco;
139                 reco->ant=anterior;
140                 free(bor);
141             }
142         }
143     }
144
145
146 int main()
147 {
148     insertarPrimero(100);
149     insertarPrimero(30);
150     insertarPrimero(10);
151     insertarPrimero(25);
152     imprimir();
153     insertarUlitmo(1000);
154     imprimir();
155     borrar(2);
156     imprimir();
157     liberar();
158     getch();
159     return 0;
160 }
```

Si ejecutamos este será el resultado:



Capítulo 172.- Recursividad Conceptos básicos – 1

Primero debemos decir que la recursividad no es una estructura de datos, sino que es una técnica de programación que nos permite que un bloque de instrucciones se ejecute n veces. Remplaza en ocasiones a estructuras repetitivas.

Este concepto será de gran utilidad para los conceptos de la estructura de datos tipo árbol.

La recursividad es un concepto difícil de entender en principio, pero luego de analizar diferentes problemas los conceptos se aclaran.

En C las funciones pueden llamarse a sí mismos. Si dentro de una función existe la llamada a sí mismo decimos que la función es recursiva.

Cuando una función se llama a sí misma, se asigna espacio en la pila para las nuevas variables locales y parámetros.

El volver de una llamada recursiva, se recupera de la pila las variables locales y parámetros antiguos y la ejecución se reanuda en el punto de la llamada a la función.

Problema 1

Implementación de una función recursiva.

```
#include <stdio.h>
#include<conio.h>

void repetir()
{
    repetir();
}

int main()
{
    repetir();
    getch();
    return 0;
}
```

La función repetir se recursiva porque dentro de la función se llama a sí misma.

Cuando ejecuta este programa se bloqueará y generará error.

Analicemos como funciona:

Primero se ejecuta la función main, luego se llama la función repetir.

Hay que tener en cuenta que cada vez que se llama a la función se reserva 4 bytes por lo general de memoria que se liberarán cuando finalice su ejecución.

La primera línea de la función llama a la función repetir, es decir que se reservan 4 bytes nuevamente. Se ejecuta nuevamente una instancia del la función repetir y así sucesivamente hasta que la pila estática se colme y se cuelgue el programa.

Problema 2

Implementar una función que reciba un parámetro de tipo entero y luego llame en forma recursiva con el valor del parámetro menos 1.

```
#include <stdio.h>
#include<conio.h>

void imprimir(int x)
{
    printf("%i ",x);
    imprimir(x - 1);
}
int main()
{
    imprimir(5);
    getch();
    return 0;
}
```

Desdela main se llama a la función imprimir y se le envía el valor 5. El parámetro x recibe el valor 5. Se ejecuta el algoritmo de la función, imprime el contenido del parámetro (5) y seguidamente se llama a una función, en este caso a sí misma (por eso decimos que es una función recursiva), enviándole el valor 4.

El parámetro x recibe el valor 4 y se imprime en pantalla el cuatro, llamando nuevamente a la función imprimir enviándole el valor 3.

Si continuamos este algoritmo podremos observar que en pantalla se imprime:

```
5 4 3 2 1 0 -1 -2 -3   . . . . . .
```

hasta que se bloquee el programa.

Tener en cuenta que cada llamada a una función consume 4 bytes por la llamada y en este caso 4 bytes por el parámetro x. Como nunca finaliza la ejecución completa de las funciones se desborda la pila estática por las sucesivas llamadas.

Problema 3:

Implementar una función recursiva que imprima en forma descendente de 5 a 1 de uno en uno.

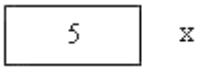
```
#include <stdio.h>
#include<conio.h>

void imprimir(int x)
{
    if (x > 0)
    {
        printf("%i ", x);
        imprimir(x - 1);
    }
}

int main()
{
    imprimir(5);
    getch();
    return 0;
}
```

Con este ejemplo se presenta una situación donde debe analizarse línea a línea la ejecución del programa y el porqué de estos resultados.

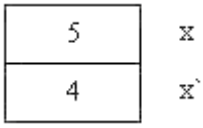
¿Por qué se imprime en pantalla 1 2 3 4 5 ?

Veamos como se aplican las llamadas recursivas:

En la primera llamada desde la función main el parámetro x recibe el valor 5.

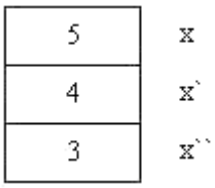


Cuando llamamos desde al misma función le enviamos el valor de x menos 1 y la memoria ueda de la siguiente forma:

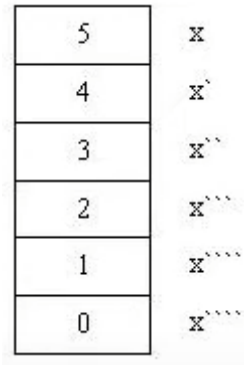


Debemos entender que el parámetro x en la nueva llamada está en otra parte de la memoria y que almacena un 4, nosotros le llamaremos x prima.

Comienza a ejecutarse la función, la condición del if se valúa como verdadero por lo que entra al bloque y llama recursivamente a la función imprimir pasándole el valor 3 al parámetro.



Nuevamente la condición se valúa como verdadero y llama a la función enviándole un 2, lo mismo ocurre cuando le envía un 1 y un 0.




```
void imprimir(int x)
{
    if (x > 0)
    {
        imprimir(x - 1);
        printf("%i ", x);
    }
}
```

Cuando x vale 0 la condición if se valúa como falsa y sale de la función imprimir.

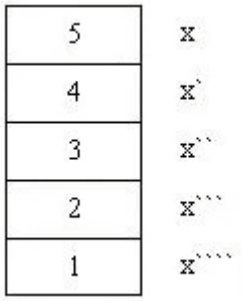
¿Qué línea ahora se ejecuta?

¿Vuelve a la función main ? NO.

Recordemos que la última llamada de la función imprimir se había hecho desde la misma función imprimir por lo que vuelva a la línea:

```
printf("%i ", x);
```

Ahora si analizamos que valor tiene el parámetro x. Observamos la pila de llamadas del gráfico:



x cuarta tiene el valor 1. Por lo que se imprime dicho valor en pantalla.

Luego de imprimir el 1 finaliza la ejecución de la función, se libera espacio ocupado por el parámetro x y pasa a ejecutarse la siguiente línea donde se había llamado a la función:

```
printf("%i ", x);
```

Ahora x en esta instancia de la función tiene el valor 2.

Así sucesivamente hasta liberar todas las llamadas recursivas.

Es importante tener en cuenta que siempre en una fusión recursiva debe haber un if para finalizar la recursividad (en caso contrario la función recursiva será infinita y provocará que el programa se bloquee).

Capítulo 173.- Recursividad Conceptos básicos – 2

Problema

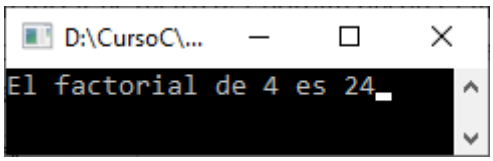
Otro problema típico que se presenta para analizar la recursividad es el obtener el factorial de un número.

Recordar que el factorial de un número es el resultado que se obtiene de multiplicar dicho número por el anterior y así sucesivamente hasta llegar a uno.

Ej. El factorial de 4 es 4 * 3 * 2* 1 es decir 24.

```
1  #include<stdio.h>
2  #include<conio.h>
3
4  int factorial(int fact)
5  {
6      if (fact>1)
7      {
8          int valor=fact*factorial(fact-1);
9          return valor;
10     }
11     else
12         return 1;
13
14 }
15
16 int main()
17 {
18     printf("El factorial de 4 es %i", factorial(4));
19     getch();
20     return 0;
21 }
22
```

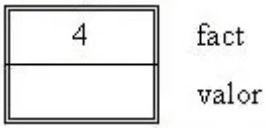
Si ejecutamos este será el resultado:



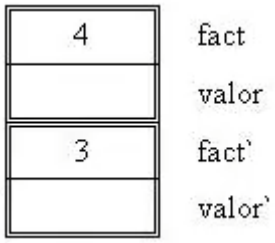
La función factorial es recursiva porque desde la misma función llamamos a la función factorial.

Debemos hacer el seguimiento del problema para analizar cómo se calcula.

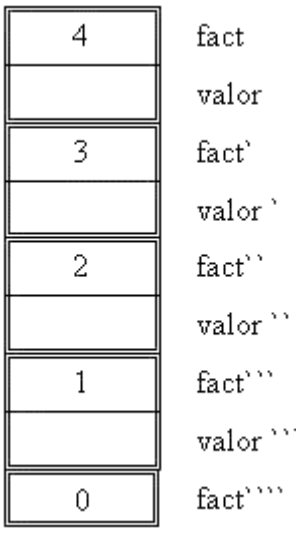
La memoria en la primera llamada:



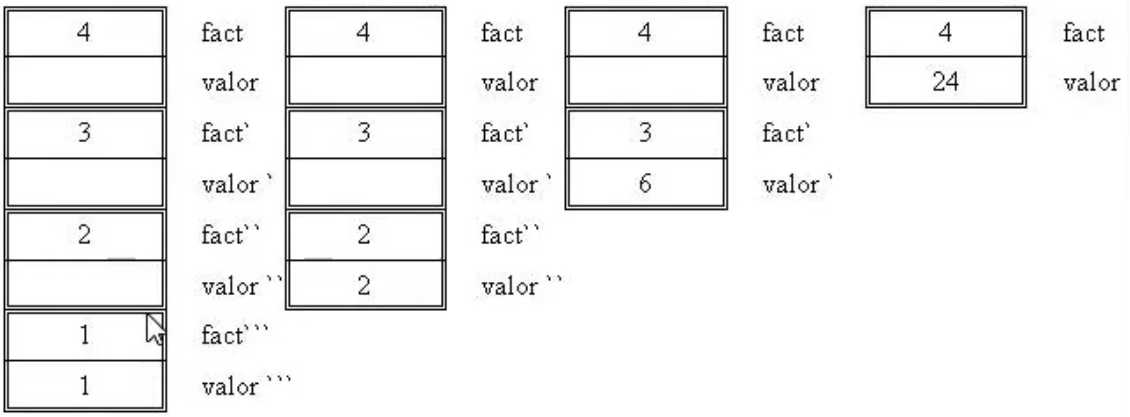
fact recibe el valor de 4 y valor se cargará con el valor que obtenga con el producto de fac por vector devuelto por la función factorial (llamada recursiva).



Nuevamente se llama recursivamente hasta que el parámetro fact reciba el valor 0.



Cuando fact recibe un cero la condición del if se valúa como falsa y ejecuta el else retornando un 1, la variable local de la llamada anterior a la función queda de la siguiente manera:



Es importantísimo entender la liberación del espacio de las variables locales y los parámetros en las sucesivas llamadas recursivas.

Por último la función main recibe "valor", en este caso el valor 24.

Capítulo 174.- Recursividad Conceptos básicos – 3

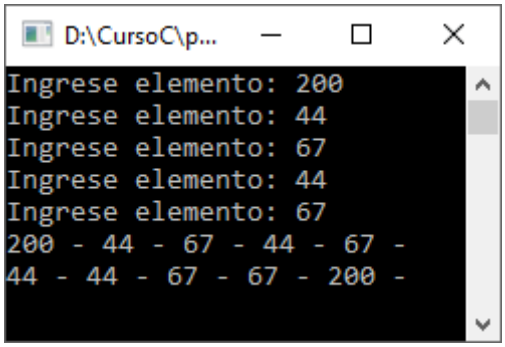
Problema

Implementar un algoritmo recursivo para ordenar los elementos de un vector.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   #define CANTIDAD 5
5
6   void cargar(int vec[CANTIDAD])
7   {
8       for(int f=0; f<CANTIDAD; f++)
9       {
10          printf("Ingrese elemento: ");
11          scanf("%i", &vec[f]);
12      }
13  }
14
15  void ordenar(int vec[5], int n)
16  {
17      if (n>1)
18      {
19          for(int y=0; y<n-1; y++)
20          {
21              if(vec[y]>vec[y+1])
22              {
23                  int aux=vec[y];
24                  vec[y]=vec[y+1];
25                  vec[y+1]=aux;
26              }
27          }
28          ordenar(vec, n-1);
29      }
30  }
31
32
33  void imprimir(int vec[CANTIDAD])
34  {
35      for (int x=0; x<5;  x++)
36      {
37          printf("%i - ", vec[x]);
38      }
39      printf("\n");
40  }
41
42
43  int main()
44  {
45      int vec[CANTIDAD];
46      cargar(vec);
47      imprimir(vec);
48      ordenar(vec, CANTIDAD);
```

```
49      imprimir(vec);
50      getch();
51      return 0;
52  }
```

Si ejecutamos este será el resultado:



Capítulo 175.- Recursividad: problema donde conviene aplicar recursividad

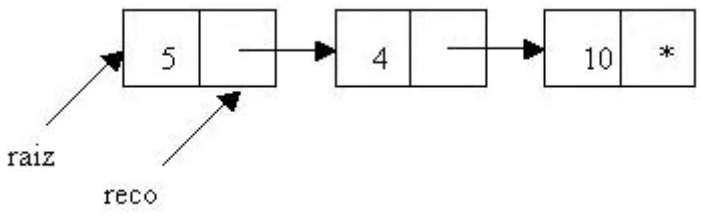
En el capítulo anterior se vieron pequeños problemas para entender como funciona la recursividad, pero no se desarrollaron problemas donde conviene utilizar la recursividad.

Problema

Imprimir la información de una lista simple encadenada de atrás para adelante.

El empleo de estructuras repetitivas para resolver este problema es bastante engorroso y lento (debemos avanzar hasta el último nodo e imprimir, luego avanzar desde el principio hasta el anteúltimo nodo y así sucesivamente)

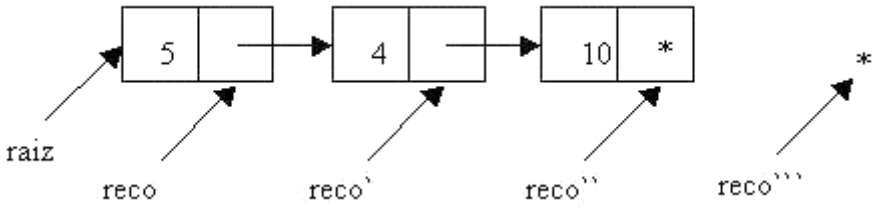
El ejemplo de la recursividad para este problema hace más sencillo su solución.



Desde el main llamamos a la función imprimir y le pasamos la dirección del primer nodo de la lista.

El parámetro reco recibe en la primera llamada la dirección raiz. Si reco es distinto a NULL llamamos recursivamente a la función enviándole la dirección del puntero sig del nodo.

Por lo que el parámetro reco recibe la dirección del segundo nodo.



Podemos observar como en las distintas llamadas recursivas el parámetro reco apunta a un nodo.

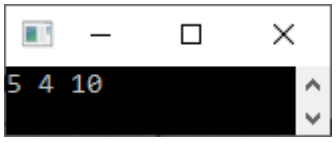
Cuando se van desapilando las llamadas recursivas se imprime primeramente el 10 luego el 4 y por último el 5.

```
1  #include<stdio.h>
2  #include<conio.h>
3  #include<stdlib.h>
4
5  struct nodo {
6      int info;
7      struct nodo *sig;
8  };
```

```
 9
10  struct nodo *raiz=NULL;
11
12  void insertarPrimero(int x)
13  {
14      struct nodo *nuevo;
15      nuevo=malloc(sizeof(struct nodo));
16      nuevo->info=x;
17      nuevo->sig=raiz;
18      raiz=nuevo;
19  }
20
21  void liberar()
22  {
23      struct nodo *reco=raiz;
24      struct nodo *bor;
25      while (reco!=NULL)
26      {
27          bor=reco;
28          reco=reco->sig;
29          free(bor);
30      }
31  }
32
33  void imprimirizquierdaderecha()
34  {
35      struct nodo *reco=raiz;
36      while(reco!=NULL)
37      {
38          printf("%i ", reco->info);
39          reco=reco->sig;
40      }
41      printf("\n");
42  }
43
44
45  int main()
46  {
47      insertarPrimero(10);
48      insertarPrimero(4);
49      insertarPrimero(5);
50      imprimirizquierdaderecha();
51      liberar();
52      getch();
53      return 0;
54  }
```

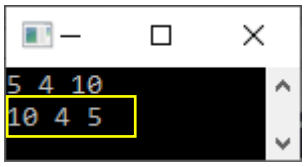
Hasta aquí este condigo ya lo hemos utilizado en capítulos anteriores.

Este será el resultado:



```
44  void imprimir(struct nodo *reco)
45  {
46      if(reco!=NULL)
47      {
48          imprimir(reco->sig);
49          printf("%i ", reco->info);
50      }
51  }
52
53
54  int main()
55  {
56      insertarPrimero(10);
57      insertarPrimero(4);
58      insertarPrimero(5);
59      imprimirizquierdaderecha();
60      imprimir(raiz);
61      liberar();
62      getch();
63      return 0;
64  }
```

Vamos a ejecutar de nuevo:

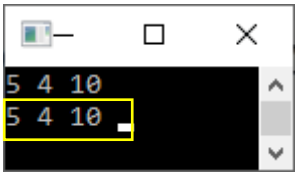


Ahora se muestras los nodo desde el final hacia el principio.

Para entender mejor la recursividad te propongo que realices el siguiente cambio.

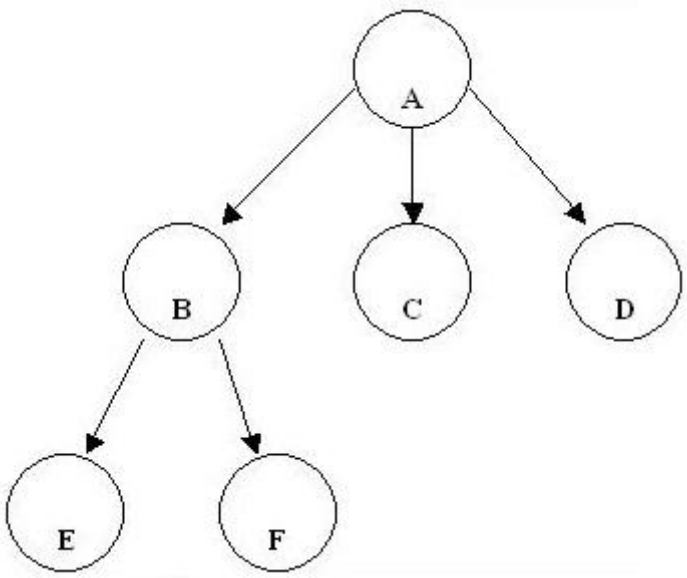
```
44  void imprimir(struct nodo *reco)
45  {
46      if(reco!=NULL)
47      {
48          printf("%i ", reco->info);
49          imprimir(reco->sig);
50      }
51  }
```

Antes de ejecutar piensa cual puede ser el resultado:



Capítulo 176.- Estructura dinámica en C: Conceptos de árboles

Igual que la lista, el árbol es una estructura de datos. Son muy eficientes para la búsqueda de información. Los árboles soportan estructuras no lineales.



Algunos conceptos de la estructura de datos tipo árbol:

Nodo hoja: Es un nodo sin descendientes (Nodo terminal).
Ej. Nodos E, F, C y D.

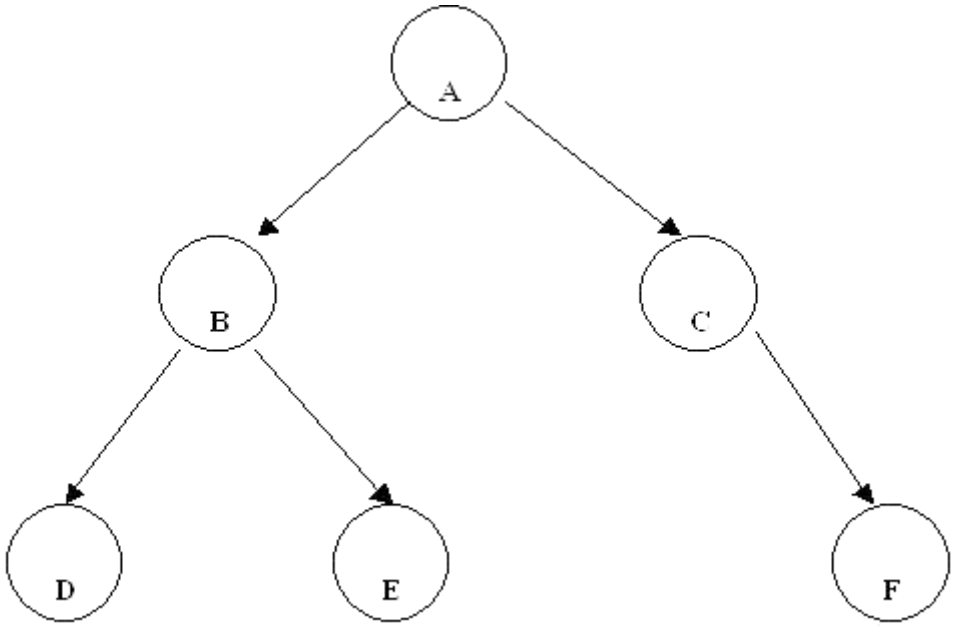
Nodo interior: Es un nodo que no es hoja.
Ej. Nodos A y B.

Nivel de un árbol: El nodo A está en el nivel 1 sus descendientes directos están en el nivel 2 y así sucesivamente.
El nivel del árbol está dado por el nodo de máximo nivel.
Ej. Este árbol es de nivel 3.

Grado de un nodo: Es el número de nodos hijos que tiene dicho nodo (solo se tiene en cuenta los nodos interiores).
Ej. El nodo A tiene el grado 3.
El nodo B tiene el grado 2.
Los otros nodos no tienen grado porque no tienen descendientes.

Grado de un árbol: Es el máximo de los grados que deben ser recorridos para llegar a un nodo x, partiendo de la raíz.
La raíz tiene longitud de camino 1, sus descendientes directos tienen longitud de camino 2, etc.
En forma general un nodo en el nivel 1 tiene longitud de camino i.

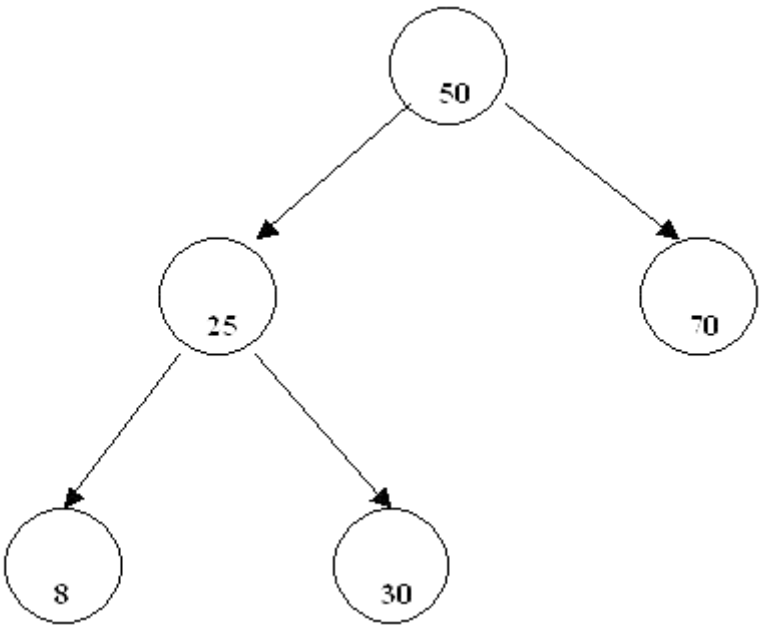
Árbol binario: Un árbol es binario si cada nodo tiene como máximo 2 descendientes.



Para cada nodo está definido es subárbol izquierdo y el derecho.
Para el nodo A el subárbol izquierdo está constituido por los nodos B, D y E. Y el subárbol derecho está formado por los nodos C y F.
Lo mismo para el nodo B tiene el subárbol izquierdo con el nodo (D) y un nodo en el subárbol derecho (E).
El nodo D tiene ambos subárboles vacíos.
El nodo C tiene el subárbol izquierdo vacío y el subárbol derecho con un nodo (F).

Hay nodos hoja en los niveles 4, 3, y 2. No debería haber nodos hojas en el nivel 2.

Árbol binario ordenado: Si para cada nodo árbol, los nodos ubicados a la izquierda son inferiores al que consideramos raíz para ese momento y los nodos ubicados a la derecha son mayores a la raíz.



Ej. Analicemos si se trata de un árbol binario ordenado:

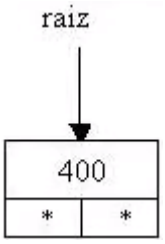
Para el nodo tiene el 50:
¿Los nodos del subárbol izquierdo son todos menores a 50? 8, 25, 30 Si.
¿Los nodos del subárbol derecho son todos mayores a 50? 70 Si.
Para el nodo que tiene el 25:
¿Los nodos del subárbol izquierdo son todos menores a 25? 8 Si.
¿Los nodos del subárbol derecho son todos mayores a 25? 30 Si.

No hace falta analizar los nodos hoja, Si todas las respuestas son afirmativas podemos luego decir que se trata de un árbol binario ordenado.

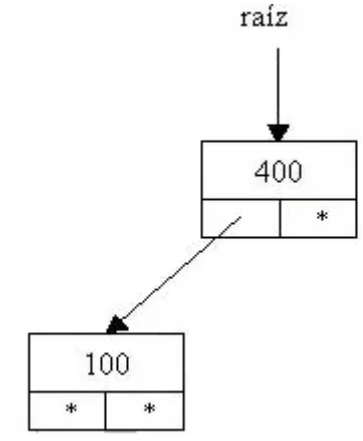
Para administrar un árbol binario ordenado debemos tener especial cuidado en la inserción.

Inicialmente el árbol está vacío, es decir raíz punta a NULL.

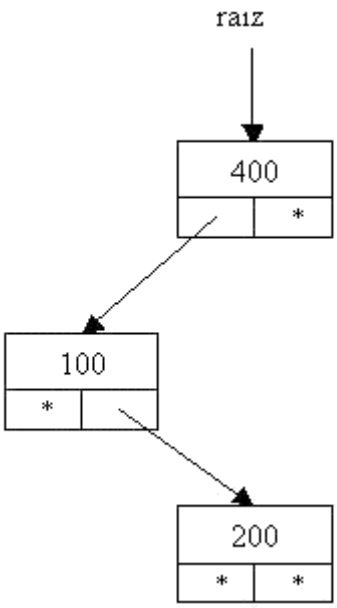
Insertamos el 400.



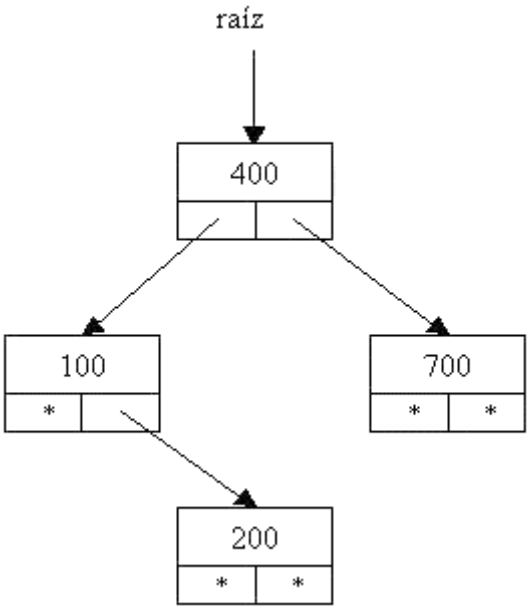
Insertamos el valor 100. Debemos analizar si raíz es distinto a NULL verificamos si 100 es mayor o menor a la información del nodo apuntado por raíz, en este caso es menor y como el subárbol izquierdo es NULL debemos insertarlo allí.



Insertamos el 200. Hay que tener en cuenta que siempre comenzamos las comparaciones a partir de la raíz. El 200 es menor que 400, descendemos por el subárbol izquierdo. Luego analizamos y vemos que el 200 es mayor a 100, debemos avanzar por la derecha. Como el subárbol derecho es NULL lo insertamos en dicha posición.



Insertamos el 700 y el árbol será:



Como podemos observar si cada vez que insertamos un nodo respetamos esta algoritmo siempre estaremos en presencia de un árbol binario ordenado. Posteriormente veremos el algoritmo en C para la inserción de información en el árbol.

Búsqueda de información en un árbol binario ordenado

Este es una de los principales usos de los árboles binarios.

Para realizar la búsqueda debemos ir comprobando la información a buscar y descender por el subárbol izquierdo o derecho según corresponda.

Ej. Si el árbol anterior necesitamos verificar si está almacenado el 700, primero verificamos si la información del nodo apuntado por raíz es 700, en caso negativo si la información a buscar (700) es mayor a la información de dicho nodo (400) es caso afirmativo descendemos por el subárbol derecho en caso contrario descendemos por el subárbol izquierdo.

Este proceso lo repetiremos hasta encontrar la información buscada o encontrar un subárbol vacío.

Recorridos de árboles binarios

Recorrer: Pasar a través del árbol enumerando cada uno de sus nodos una vez.

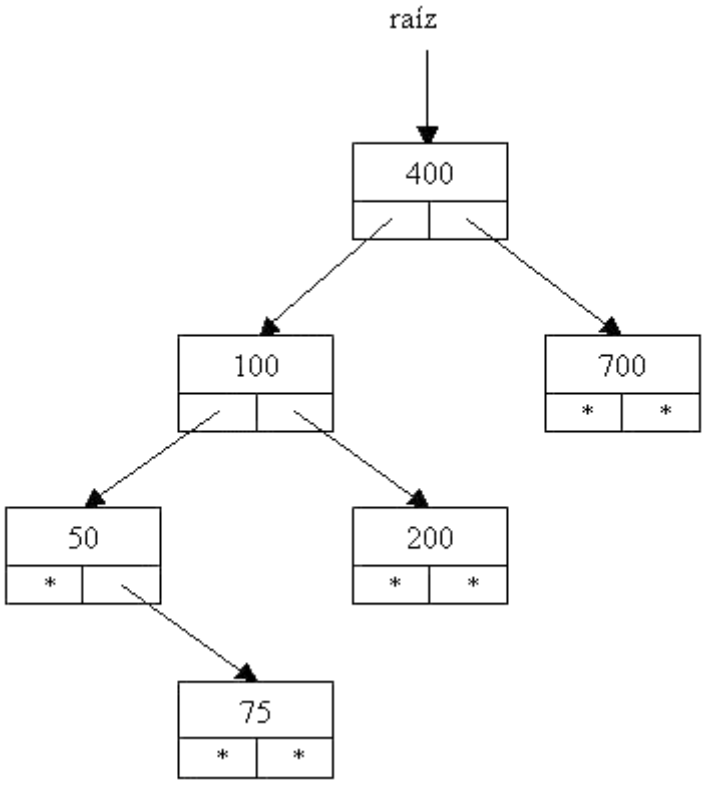
Visitar: Realizar algún procesamiento del nodo (borrar, modificar, etc.)

Los árboles pueden ser recorridos en varios órdenes:

Pre-orden:
- Visitar la raíz.
- recorrer el subárbol izquierdo en pre-orden.
- recorrer el subárbol derecho en pre-orden.

Entre-orden:
- Recorrer el subárbol izquierdo en entre-orden.
- Visitar la raíz.
- Recorrer el subárbol derecho en entre-orden.

Post-orden:
- Recorrer el subárbol izquierdo en post-orden.
- Recorrer el subárbol derecho en post-orden.
- Visitar la ráiz.



Veamos cómo se imprimen las informaciones de los nodos según su recorrido:

Recorrido pre-orden:

Visitar la raiz: 400
Recorrer el subárbol izquierdo en preorden.
Visitar la raiz: 100
Recorrer el subárbol izquierdo en preorden.
Visitar la raiz: 50
Recorrer el subárbol izquierdo en preorden.
Vacío
Recorrer el subárbol derecho en preorden.
Visitar la raiz: 75
Recorrer el subárbol izquierdo en preorden.
Vacío
Recorrer el subárbol derecho en preorden.
Vacío
Recorrer el subárbol derecho en preorden.
Visitar la raiz: 200
Recorrer el subárbol izquierdo en preorden.
Vacío
Recorrer el subárbol derecho en preorden.
Vacío
Recorrer el subárbol derecho en preorden.
Visitar la raiz: 700
Recorrer el subárbol izquierdo en preorden.
Vacío
Recorrer el subárbol derecho en preorden.
Vacío

Es decir que el orden de impresión de la información es:

```
50 - 75 - 100 - 200 - 400 - 700
```

Si observamos podemos ver que la información aparece ordenada.

Este tipo de recorrido es muy útil cuando queremos procesar la información del árbol en orden.

Recorrido post-orden:

Recorrer el subárbol izquierdo en postorden.
Recorrer el subárbol izquierdo en postorden.
Recorrer el subárbol izquierdo en postorden.
Vacío
Recorrer el subárbol derecho en postorden.
Recorrer el subárbol izquierdo en postorden.
Vacío
Recorrer el subárbol derecho en postorden.
Vacío
Visitar la raiz: 75
Visitar la raiz: 50
Recorrer el subárbol derecho en postorden.
Recorrer el subárbol izquierdo en postorden.
Vacío
Recorrer el subárbol derecho en postorden.
Vacío
Visitar la raiz: 200
Visitar la raiz: 100

Recorrer el subárbol derecho en postorden.
Recorrer el subárbol izquierdo en postorden.
Vacío
Recorrer el subárbol derecho en postorden.

Visitar la raiz: 700

Visitar la raiz: 400

Es decir que el orden de impresión de la información es:

```
75 - 50 - 200 - 100 - 700 - 400
```

Es importante analizar que el recorrido del árbol es recursivo. Recorrer un subárbol es semejante a recorrer un árbol.

Es buena práctica dibujar el árbol en un papel y hacer el seguimiento del recorrido y las visitas a cada nodo.

Recorrido entre-orden:

Recorrer el subárbol izquierdo en entreorden.
Recorrer el subárbol izquierdo en entreorden.
Recorrer el subárbol izquierdo en entreorden.
Vacío
Visitar la raiz: 50
Recorrer el subárbol derecho en entreorden.
Recorrer el subárbol izquierdo en entreorden.
Vacío
Visitar la raiz : 75
Recorrer el subárbol derecho en entreorden.
Vacío
Visitar la raiz: 100
Recorrer el subárbol derecho en entreorden.
Recorrer el subárbol izquierdo en entreorden.
Vacío
Visitar la raiz: 200
Recorrer el subárbol derecho en entreorden.
Vacío

Visitar la raiz: 400
Recorrer el subárbol derecho en entreorden.
Recorrer el subárbol izquierdo en entreorden.
Vacío
Visitar la raiz: 700
Recorrer el subárbol derecho en entreorden.
Vacío

Es decir el orden de impresión de la información es:

```
50 - 75 - 100 - 200 - 400 - 700
```

Si observamos podemos ver que la información aparece ordenada.

Este tipo de recorrido es muy útil cuando queremos procesar la información del árbol en orden.

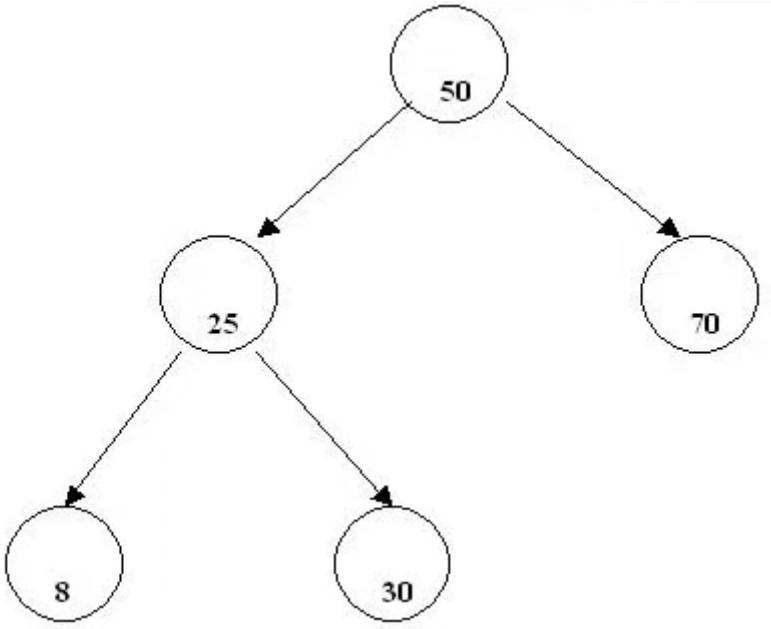
Capítulo 177.- Estructuras dinámicas en C: Implementación de un árbol binario ordenado – 1

Problema

Desarrollar un programa para la administración de un árbol binario ordenado con información de tipo int.

Implementar las funciones para recorrer el árbol en pre, entre y post orden.

Insertar estos valores:




```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<stdlib.h>
4
5   struct nodo {
6       int info;
7       struct nodo *izq, *der;
8   };
9
10  struct nodo *raiz= NULL;
11
12  void insertar(int x)
13  {
14      struct nodo *nuevo;
15      nuevo=malloc(sizeof(struct nodo));
16      nuevo->info=x;
17      nuevo->izq=NULL;
18      nuevo->der=NULL;
19      if (raiz==NULL)
20          raiz=nuevo;
21      else
22      {
23          struct nodo *anterior, *reco;
24          reco=raiz;
```

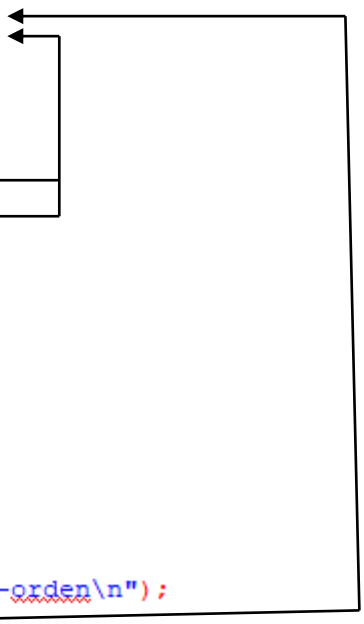
```
25          while (reco!=NULL)
26          {
27              anterior=reco;
28              if(x<reco->info)
29                  reco=reco->izq;
30              else
31                  reco=reco->der;
32
33          }
34          if(x<anterior->info)
35              anterior->izq=nuevo;
36          else
37              anterior->der=nuevo;
38      }
39  }
40
41  int main()
42  {
43      insertar(50);
44      insertar(25);
45      insertar(70);
46      insertar(8);
47      insertar(30);
48      getch();
49      return 0;
50  }
51
```

Con este codigo ya hemos introducido valores en una estructura de árbol.

Vamos a ver las tres formas de consultar los nodos:

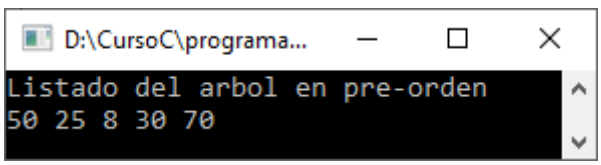
Recorrido pre-orden:

```
41  void imprimirPre(struct nodo *reco)
42  {
43      if (reco!=NULL)
44      {
45          printf("%i ", reco->info);
46          imprimirPre(reco->izq);
47          imprimirPre(reco->der);
48      }
49  }
50
51  int main()
52  {
53      insertar(50);
54      insertar(25);
55      insertar(70);
56      insertar(8);
57      insertar(30);
58      printf("Listado del arbol en pre-orden\n");
59      imprimirPre(raiz);
```



```
41  void imprimirPre(struct nodo *reco)
42  {
43      if (reco!=NULL)
44      {
45          printf("%i ", reco->info);
46          imprimirPre(reco->izq);
47          imprimirPre(reco->der);
48      }
49  }
50
51  int main()
52  {
53      insertar(50);
54      insertar(25);
55      insertar(70);
56      insertar(8);
57      insertar(30);
58      printf("Listado del arbol en pre-orden\n");
59      imprimirPre(raiz);
60      getch();
61      return 0;
62  }
```

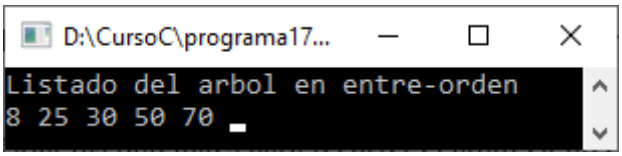
Si ejecutamos este será el resultado:



Recorrido entre-orden:

```
41  void imprimirEntre(struct nodo *reco)
42  {
43      if (reco!=NULL)
44      {
45          imprimirEntre(reco->izq);
46          printf("%i ", reco->info);
47          imprimirEntre(reco->der);
48      }
49  }
50
51  int main()
52  {
53      insertar(50);
54      insertar(25);
55      insertar(70);
56      insertar(8);
57      insertar(30);
58      printf("Listado del arbol en entre-orden\n");
59      imprimirEntre(raiz);
60      getch();
61      return 0;
62  }
```

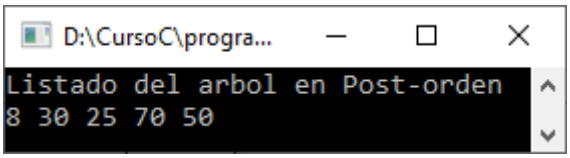
Si ejecutamos este será el resultado:



Recorrido post-orden:

```
41  void imprimirPost(struct nodo *reco)
42  {
43      if (reco!=NULL)
44      {
45          imprimirPost(reco->izq);
46          imprimirPost(reco->der);
47          printf("%i ", reco->info);
48      }
49  }
50
51  int main()
52  {
53      insertar(50);
54      insertar(25);
55      insertar(70);
56      insertar(8);
57      insertar(30);
58      printf("Listado del arbol en Post-orden\n");
59      imprimirPost(raiz);
60      getch();
61      return 0;
62  }
```

Si ejecutamos este será el resultado:



Para borrar todos los nodos:

```
51  void borrar(struct nodo *reco)
52  {
53      if (reco!=NULL)
54      {
55          borrar(reco->izq);
56          borrar(reco->der);
57      }
58  }
59
60  int main()
61  {
62      insertar(50);
63      insertar(25);
64      insertar(70);
65      insertar(8);
66      insertar(30);
67      printf("Listado del arbol en Post-orden\n");
68      imprimirPost(raiz);
69      borrar(raiz);
70      getch();
71      return 0;
72  }
```

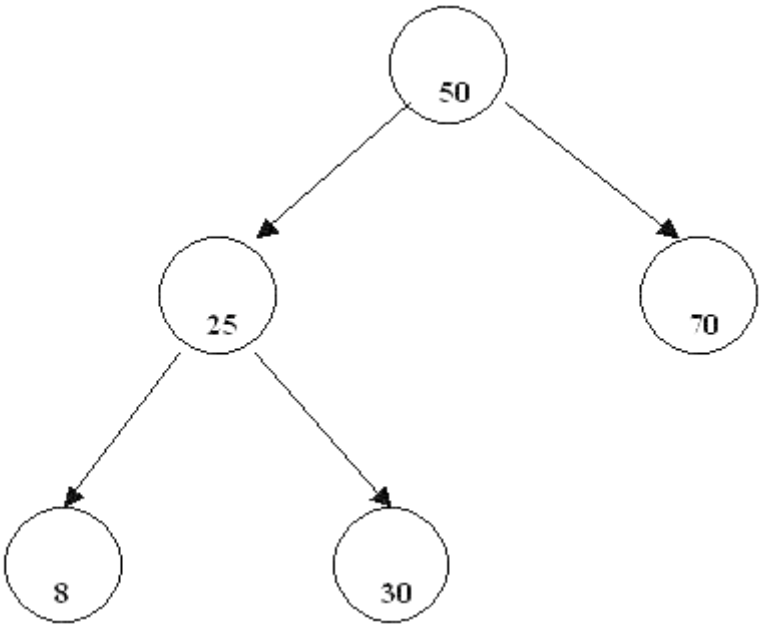
Capítulo 178.- Estructuras dinámicas en C: Implementación de un árbol binario ordenado – 2

Problema

Confeccionar un programa que permita insertar un entero en un árbol binario ordenado verificando que no se encuentra previamente dicho número.

Desarrollas las siguientes funcionalidades:

1- Retornar la cantidad de nodos del árbol.
2- Retornar la cantidad de nodos hoja del árbol.
3- Imprimir en entre orden.
4- Imprimir en entre orden junto al nivel donde se encuentra dicho nodo.
5- Retornar la altura del árbol.
6- Imprimir el mayor valor del árbol.
7- Borrar el nodo menor del árbol.



Código que reutilizamos del capítulo anterior:

```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<stdlib.h>
4
5   struct nodo {
6       int info;
7       struct nodo *izq, *der;
8   };
9
10  struct nodo *raiz= NULL;
11
```

```
12  void insertar(int x)
13  {
14      struct nodo *nuevo;
15      nuevo=malloc(sizeof(struct nodo));
16      nuevo->info=x;
17      nuevo->izq=NULL;
18      nuevo->der=NULL;
19      if (raiz==NULL)
20          raiz=nuevo;
21      else
22      {
23          struct nodo *anterior, *reco;
24          reco=raiz;
25          while (reco!=NULL)
26          {
27              anterior=reco;
28              if(x<reco->info)
29                  reco=reco->izq;
30              else
31                  reco=reco->der;
32
33          }
34          if(x<anterior->info)
35              anterior->izq=nuevo;
36          else
37              anterior->der=nuevo;
38      }
39  }
```

Vamos a crear un método para verificar que no se encuentre previamente dicho número:

```
44  int existe(int x)
45  {
46      struct nodo *reco=raiz;
47      while (reco!=NULL)
48      {
49          if(x==reco->info)
50              return 1;
51          else
52              if(x>reco->info)
53                  reco=reco->der;
54              else
55                  reco=reco->izq;
56      }
57      return 0;
58  }
```

Ahora vamos a modificar el método insertar:

```
12  void insertar(int x)
13  {
14      if(!existe(x))
15      {
16          struct nodo *nuevo;
17          nuevo=malloc(sizeof(struct nodo));
```

```
18          nuevo->info=x;
19          nuevo->izq=NULL;
20          nuevo->der=NULL;
21          if (raiz==NULL)
22              raiz=nuevo;
23          else
24          {
25              struct nodo *anterior, *reco;
26              reco=raiz;
27              while (reco!=NULL)
28              {
29                  anterior=reco;
30                  if(x<reco->info)
31                      reco=reco->izq;
32                  else
33                      reco=reco->der;
34
35              }
36              if(x<anterior->info)
37                  anterior->izq=nuevo;
38              else
39                  anterior->der=nuevo;
40          }
41      }
42  }
```

Vamos a crear el método principal (main):

```
60  int main()
61  {
62      insertar(50);
63      insertar(25);
64      insertar(70);
65      insertar(8);
66      insertar(30);
67      getch();
68      return 0;
69  }
```

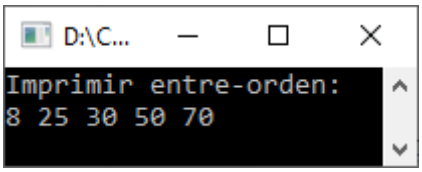
A continuación vamos a realizar el método para imprimir en entre orden los nodos.

```
60  void imprimirEntre(struct nodo *reco)
61  {
62      if (reco!=NULL)
63      {
64          imprimirEntre(reco->izq);
65          printf("%i ", reco->info);
66          imprimirEntre(reco->der);
67      }
68  }
```

Modificamos el método main:

```
70  int main()
71  {
72      insertar(50);
73      insertar(25);
74      insertar(70);
75      insertar(8);
76      insertar(30);
77      printf("Imprimir entre-orden: \n");
78      imprimirEntre(raiz);
79      getch();
80      return 0;
81  }
```

Si ejecutamos este será el resultado:



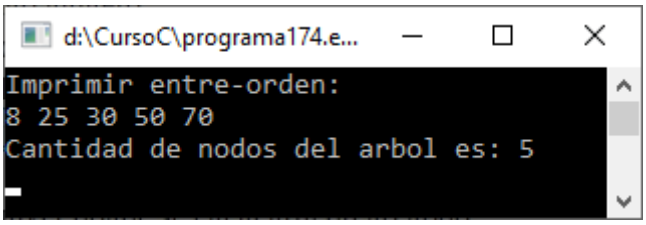
Borrar todos los nodos:

```
70  void borrar(struct nodo *reco)
71  {
72      if(reco!=NULL)
73      {
74          borrar(reco->izq);
75          borrar(reco->der);
76          free(reco);
77      }
78  }
79
80  int main()
81  {
82      insertar(50);
83      insertar(25);
84      insertar(70);
85      insertar(8);
86      insertar(30);
87      printf("Imprimir entre-orden: \n");
88      imprimirEntre(raiz);
89      printf("\n");
90      borrar(raiz);
91      getch();
92      return 0;
93  }
```

Ahora vamos a realizar un método para retornar la cantidad de nodos:

```
80  void cantidad(struct nodo *reco, int *cant)
81  {
82      if(reco!=NULL)
83      {
84          (*cant)++;
85          cantidad(reco->izq,cant);
86          cantidad(reco->der,cant);
87      }
88  }
89
90  int main()
91  {
92      int canti=0;
93      insertar(50);
94      insertar(25);
95      insertar(70);
96      insertar(8);
97      insertar(30);
98      printf("Imprimir entre-orden: \n");
99      imprimirEntre(raiz);
100     printf("\n");
101     cantidad(raiz,&canti);
102     printf("Cantidad de nodos del arbol es: %i\n", canti);
103     borrar(raiz);
104     getch();
105     return 0;
106 }
```

Cuando ejecutemos este será el resultado:



Ahora vamos a crear un método que nos retorne el número de nodos hoja del árbol.

```
90  void cantidadHojas(struct nodo *reco, int *cant)
91  {
92      if (reco!=NULL)
93      {
94          if (reco->der==NULL && reco->izq==NULL)
95              (*cant)++;
96          cantidadHojas(reco->izq,cant);
97          cantidadHojas(reco->der,cant);
98      }
99  }
```

```
101  int main()
102  {
103      int canti=0;
104      int cantHojas=0;
105      insertar(50);
106      insertar(25);
107      insertar(70);
108      insertar(8);
109      insertar(30);
110      printf("Imprimir entre-orden: \n");
111      imprimirEntre(raiz);
112      printf("\n");
113      cantidad(raiz,&canti);
114      printf("Cantidad de nodos del arbol es: %i\n", canti);
115      cantidadHojas(raiz,&cantHojas);
116      printf("Cantidad de nodos hoja del arbol es: %i\n", cantHojas);
117      borrar(raiz);
118      getch();
119      return 0;
120  }
```

Si ejecutamos este será el resultado:

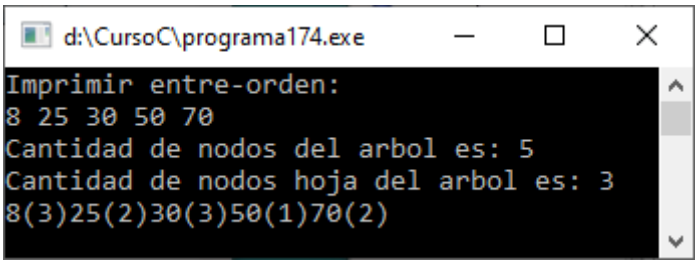
```
d:\CursoC\programa174.exe
Imprimir entre-orden:
8 25 30 50 70
Cantidad de nodos del arbol es: 5
Cantidad de nodos hoja del arbol es: 3
```

Ahora vamos a imprimir entre-orden junto al nivel donde se encuentra dicho nodo.

```
101  void imprimirEntreConNivel(struct nodo *reco, int nivel)
102  {
103      if(reco!=NULL)
104      {
105          imprimirEntreConNivel(reco->izq, nivel+1);
106          printf("%i(%i)", reco->info, nivel);
107          imprimirEntreConNivel(reco->der, nivel+1);
108      }
109  }
110
111  int main()
112  {
113      int canti=0;
114      int cantHojas=0;
115      insertar(50);
116      insertar(25);
117      insertar(70);
118      insertar(8);
119      insertar(30);
120      printf("Imprimir entre-orden: \n");
121      imprimirEntre(raiz);
122      printf("\n");
123      cantidad(raiz,&canti);
124      printf("Cantidad de nodos del arbol es: %i\n", canti);
125      cantidadHojas(raiz,&cantHojas);
126      printf("Cantidad de nodos hoja del arbol es: %i\n", cantHojas);
127      imprimirEntreConNivel(raiz,1);
```

```
128         printf("\n");
129         borrar(raiz);
130         getch();
131         return 0;
132     }
```

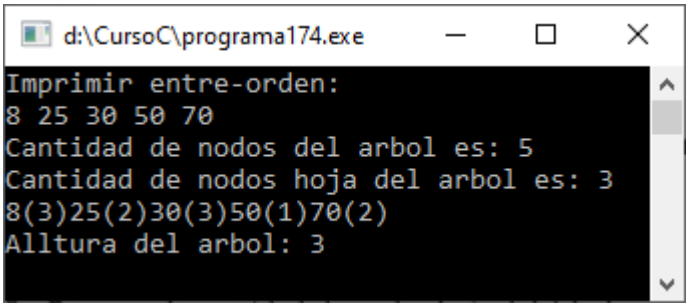
Si ejecutamos este será el resultado:



Ahora vamos a crear otra función para que retorne la altura del árbol:

```
111 void retornarAltura(struct nodo *reco, int nivel, int *altura)
112 {
113     if (reco!=NULL)
114     {
115         retornarAltura(reco->izq, nivel+1, altura);
116         if(nivel>*altura)
117             *altura=nivel;
118         retornarAltura(reco->der, nivel+1, altura);
119     }
120 }
121
122 int main()
123 {
124     int canti=0;
125     int cantHojas=0;
126     int altura=0;
127     insertar(50);
128     insertar(25);
129     insertar(70);
130     insertar(8);
131     insertar(30);
132     printf("Imprimir entre-orden: \n");
133     imprimirEntre(raiz);
134     printf("\n");
135     cantidad(raiz,&canti);
136     printf("Cantidad de nodos del arbol es: %i\n", canti);
137     cantidadHojas(raiz,&cantHojas);
138     printf("Cantidad de nodos hoja del arbol es: %i\n", cantHojas);
139     imprimirEntreConNivel(raiz,1);
140     printf("\n");
141     retornarAltura(raiz,1,&altura);
142     printf("Alltura del arbol: %i\n", altura);
143     borrar(raiz);
144     getch();
145     return 0;
146 }
```

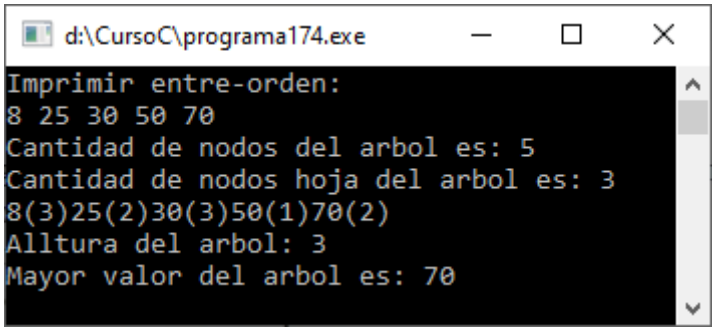
Si ejecutamos este será el resultado:



Ahora vamos a crear una función para imprimir el mayor valor del árbol:

```
122  void mayorValor()
123  {
124      if(raiz!=NULL)
125      {
126          struct nodo *reco=raiz;
127          while (reco->der!=NULL)
128              reco=reco->der;
129          printf("Mayor valor del arbol es: %i\n", reco->info);
130      }
131  }
132
133  int main()
134  {
135      int canti=0;
136      int cantHojas=0;
137      int altura=0;
138      insertar(50);
139      insertar(25);
140      insertar(70);
141      insertar(8);
142      insertar(30);
143      printf("Imprimir entre-orden: \n");
144      imprimirEntre(raiz);
145      printf("\n");
146      cantidad(raiz,&canti);
147      printf("Cantidad de nodos del arbol es: %i\n", canti);
148      cantidadHojas(raiz,&cantHojas);
149      printf("Cantidad de nodos hoja del arbol es: %i\n", cantHojas);
150      imprimirEntreConNivel(raiz,1);
151      printf("\n");
152      retornarAltura(raiz,1,&altura);
153      printf("Alltura del arbol: %i\n", altura);
154      mayorValor();
155      borrar(raiz);
156      getch();
157      return 0;
158  }
```

Si ejecutamos este será el resultado:



Ahora vamos a realizar la función para borrar el menor nodo del árbol.

```
133  void borrarMenor()
134  {
135      if(raiz!=NULL)
136      {
137          struct nodo *bor;
138          if(raiz->izq==NULL)
139          {
```

```
140                 bor=raiz;
141                 raiz=raiz->der;
142                 free(bor);
143             }
144             else
145             {
146                 struct nodo *atras=raiz;
147                 struct nodo *reco=raiz->izq;
148                 while(reco->izq!=NULL)
149                 {
150                     atras=reco;
151                     reco=reco->izq;
152                 }
153
154                 atras->izq=reco->der;
155                 free(reco);
156             }
157         }
158 }
159
160 int main()
161 {
162     int canti=0;
163     int cantHojas=0;
164     int altura=0;
165     insertar(50);
166     insertar(25);
167     insertar(70);
168     insertar(8);
169     insertar(30);
170     printf("Imprimir entre-orden: \n");
171     imprimirEntre(raiz);
172     printf("\n");
173     cantidad(raiz,&canti);
174     printf("Cantidad de nodos del arbol es: %i\n", canti);
175     cantidadHojas(raiz,&cantHojas);
176     printf("Cantidad de nodos hoja del arbol es: %i\n", cantHojas);
177     imprimirEntreConNivel(raiz,1);
178     printf("\n");
179     retornarAltura(raiz,1,&altura);
180     printf("Alltura del arbol: %i\n", altura);
181     mayorValor();
182     borrarMenor();
183     printf("Imprimir entre-orden: \n");
184     imprimirEntre(raiz);
185     borrar(raiz);
186     getch();
187     return 0;
188 }
```

Si ejecutamos este será el resultado:

```
d:\CursoC\programa174.exe
Imprimir entre-orden:
8 25 30 50 70
Cantidad de nodos del arbol es: 5
Cantidad de nodos hoja del arbol es: 3
8(3)25(2)30(3)50(1)70(2)
Alltura del arbol: 3
Mayor valor del arbol es: 70
Imprimir entre-orden:
25 30 50 70
```

Capítulo 179.- Todos los tipos de datos primitivos en el lenguaje C

Como hemos estado aprendiendo a programar utilizando el lenguaje C no nos hemos quedado a hila fino en los tipos de datos primitivos.

Cada vez que necesitamos almacenar un entero utilizamos el tipo int, también dijimos que el tipo char permite almacenar un entero pequeño, pero el lenguaje C permite definir distintos tipos de enteros según la capacidad máxima a almacenar y si necesitamos guardar el signo.

Hay cuatro modificadores de tipo para las variables enteras: signed, unsigned, short y long que nos permiten ampliar los tipos enteros en el lenguaje C.

Variables de tipo entera

- char (permite almacenar un valor entero entre -128 y 127 o entre 0 y 255 dependiendo del compilador utilizado).
- signed char (permite almacenar un valor entero entre [-128, 127]).
- unsignde char (permite almacenar un valor entero entre [0, 255]).
- signed short int (permite almacenar un valor como mínimo entre [-32767, 32767]). Cuando definimos una variable de este tipo podemos resumirla poniendo solo short o short int o signed short.
- unsigned short int (permite almacenar un valor como mínimo entre [0, 65535]). Cuando definimos una variable de este tipo podemos resumirla poniendo solo unsigned short.
- signed int (Permite almacenar un valor como mínimo entre [2147483648 , 2147833647]. Cuando definimos una variable de este tipo podemos resumirla poniendo solo int, recordemos que hasta ahora siempre utilizamos el tipo de dato para almacenar enteros, pero ahora en más podemos utilizar la más eficiente según la necesidad de almacenamiento.
- usigned int (permite almacenar un valor como mínimo entre [0, 4294967295]. Cuando definimos una variable de este tipo podemos resumirla poniendo solo usingned.
- signed long int (permite almacenar un valor como mínimo entre [-2147483648, 2147483647]. Cuando definimos una variable de este tipo podemos resumirla poniendo solo long o lont int o signed long.
- unsigned long int (permite almacenar un valor como mínimo [0, 4294967295] Cuando definimos una variable de este tipo podeos resumirla poniendo solo unsigned log.
- signed long long int (permite almacenar una valor como mínimo entre [-9223372036854775807, 9223372036854775807]. Cuando definimos una variable de este tipo podemos resumirla poniendo solo long long o long long int o signed long long.
- usigned long long int (permite almacenar un valor como mínimo entre [0, 18446744073709551615]. Cuando definimos una variable de este tipo podemos resumirla poniendo solo unsigned long long.

Problema

Definir variables de tipo entera, almacenar valores por asignación y mostrar su contenido.

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    signed char c1=65;
    printf("c1 es de tipo signed char: %c\n", c1);
    unsigned char c2=160;
    printf("c2 es de tipo unsigned char: %c\n", c2);
    signed short int e1=32000;
    printf("e1 es de tipo short int: %i\n", e1);
    unsigned short int e2=64000;
    printf("e2 es de tipo unsigned short int: %i\n", e2);
    signed int e3=-2147483648;
    printf("e3 es de tipo signed int: %i\n",e3);
    unsigned int e4=4294967295;
    printf("e4 es de tipo unsigned int: %u\n",e4);
    signed long int e5=-2147483647;
    printf("e5 es de tipo singend long int: %li\n",e5);
    unsigned long int e6=2147483647;
    printf("e6 es de tipo unsigned long int: %lu\n",e6);
    signed long long int e7=-9223372036854775807;
    printf("e7 es de tipo singend long long int: %lli\n",e7);
    unsigned long long int e8=18446744073709551615;
    printf("e8 es de tipo unsigned long long int: %llu\n",e8);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:

```
D:\CursoC\programa175.exe
c1 es de tipo signed char: A
c2 es de tipo unsigned char: á
e1 es de tipo short int: 32000
e2 es de tipo unsigned short int: 64000
e3 es de tipo signed int: -2147483648
e4 es de tipo unsigned int: 4294967295
e5 es de tipo singend long int: -2147483647
e6 es de tipo unsigned long int: 2147483647
e7 es de tipo singend long long int: -9223372036854775807
e8 es de tipo unsigned long long int: 18446744073709551615
```

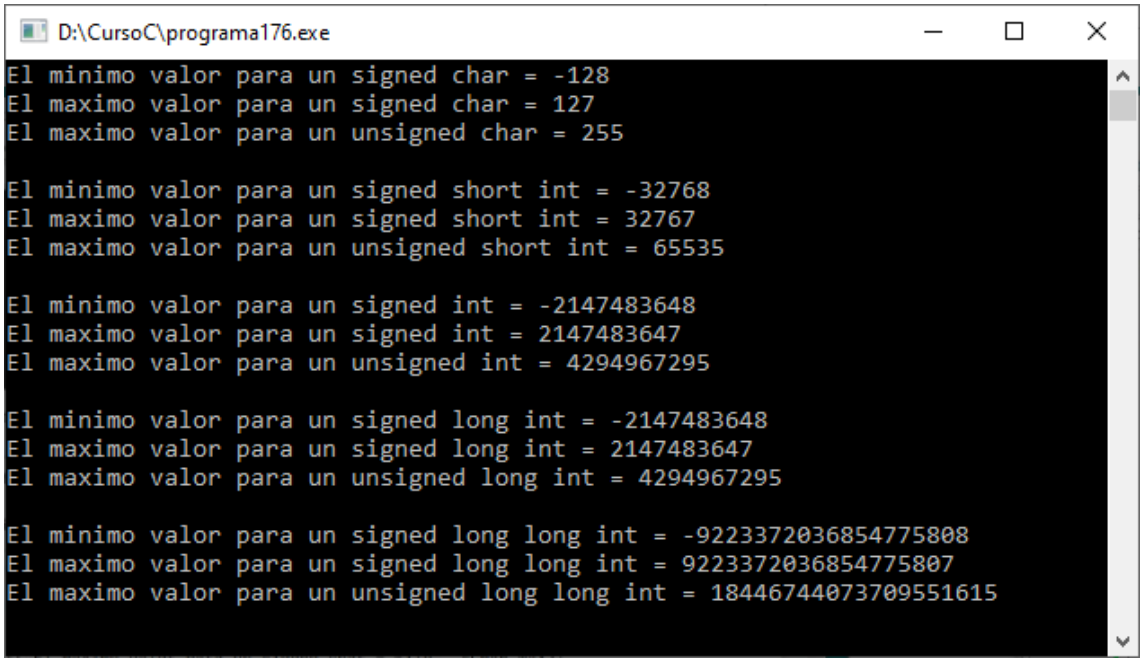
Como vemos tenemos que pasar distintos valores luego del carácter % en la función printf.

Si queremos conocer cual es el valor máximo y mínimo para cada uno de los tipos de datos enteros del lenguaje C podemos ejecutar el siguiente programa:

```c
#include <stdio.h>
#include<conio.h>
#include <limits.h>

int main() {

   printf("El minimo valor para un signed char = %i\n", SCHAR_MIN);
   printf("El maximo valor para un signed char = %i\n", SCHAR_MAX);
   printf("El maximo valor para un unsigned char = %i\n\n", UCHAR_MAX);
```

```c

    printf("El minimo valor para un signed short int = %i\n", SHRT_MIN);
    printf("El maximo valor para un signed short int = %i\n", SHRT_MAX);
    printf("El maximo valor para un unsigned short int = %i\n\n", USHRT_MAX);

    printf("El minimo valor para un signed int = %i\n", INT_MIN);
    printf("El maximo valor para un signed int = %i\n", INT_MAX);
    printf("El maximo valor para un unsigned int = %u\n\n", UINT_MAX);

    printf("El minimo valor para un signed long int = %li\n", LONG_MIN);
    printf("El maximo valor para un signed long int = %li\n", LONG_MAX);
    printf("El maximo valor para un unsigned long int = %lu\n\n", ULONG_MAX);

    printf("El minimo valor para un signed long long int = %lli\n", LONG_LONG_MIN);
    printf("El maximo valor para un signed long long int = %lli\n", LONG_LONG_MAX);
    printf("El maximo valor para un unsigned long long int = %llu\n\n", ULONG_LONG_MAX);

    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:

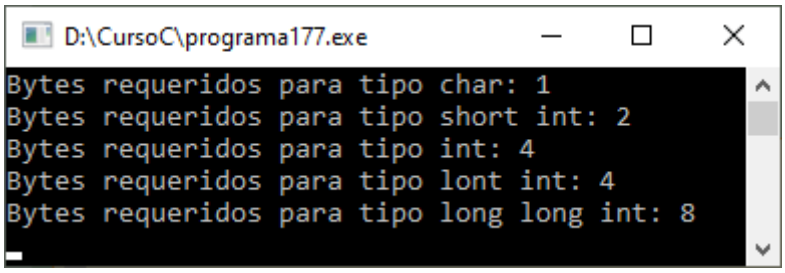


Debemos elegir la variable más adecuada según el valor máximo y mínimo a guardar para que nuestro programa ser más eficiente en cuanto a espacio para reservar como el tiempo de procesamiento de dichas variables.

Si queremos saber cuantos bytes se necesitan para reservar para cada uno de los tipos de variables podemos ejecutar el programa:

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    printf("Bytes requeridos para tipo char: %i\n",sizeof(char));
    printf("Bytes requeridos para tipo short int: %i\n",sizeof(short int));
    printf("Bytes requeridos para tipo int: %i\n",sizeof(int));
    printf("Bytes requeridos para tipo lont int: %i\n",sizeof(long int));
    printf("Bytes requeridos para tipo long long int: %i\n",sizeof(long long int));
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Como podemos ver definir una variable de tipo long long int requiere reservar 8 bytes para el almacenamiento del entero.

No dispusimos con el modificador unsigned ya que el tamaño de la variable sigue siendo igual.

Variables de tipo decimal

Cuando queremos almacenar un valor con parte decimal hemos estado definiendo variables de tipo float.

En el lenguaje C podemos definir variables decimales de tipo:

- float
- double
- long double

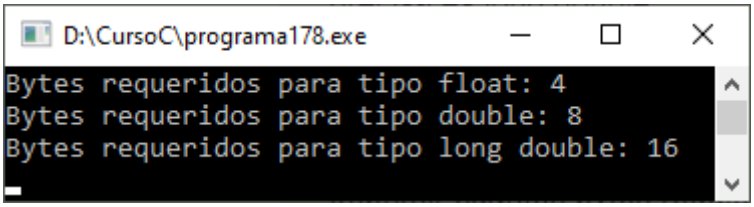
Se diferencian en la precisión como almacenan el valor real. La menos precica es el tipo float, le sigue double y finalmente la más precisa es long double.

problema:

Imprimir cuantos bytes requieren cada uno de los tipos de datos decimales que tiene el lenguaje C.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   int main()
5   {
6       printf("Bytes requeridos para tipo float: %i\n",sizeof(float));
7       printf("Bytes requeridos para tipo double: %i\n",sizeof(double));
8       printf("Bytes requeridos para tipo long double: %i\n",sizeof(long double));
9       getch();
10      return 0;
11  }
```

Si ejecutamos este será el resultado:



Capítulo 180.- Operador de molde con tipos enteros y reales (cast)

Hemos visto que cuando realizamos una operación con un conjunto de datos enteros el resultado es otro entero.

En algunas situaciones necesitamos que el resultado sea de otro tipo en estos casos utilizmos el concepto de cast (moldear) Ej:

```
float f=7/2;
printf("%f",f); //3
```

Luego podemos ver si imprimimos la variable f su contenido es 3.
Esto ocurre porque la división de un int con respecto a otro valor int da como resultado un entero.
Si queremos que el resultado sea exacto podemos anteceder entre paréntesis el tipo de datos que queremos que se moldee:

```
float f=(float)7/2;
printf("%f",f); //3.5
```

```
float f=5.3;
int x=f;
printf("%i",x); //5
```

No es necesario indicar (más allá que no genera error):

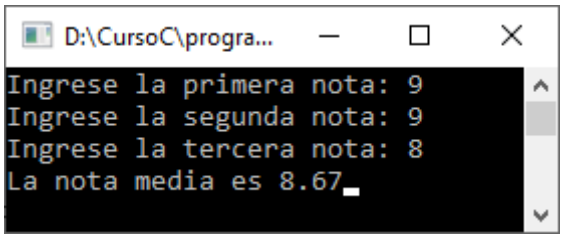
```
int x=(int)f;
```

Problema

Cargar las 3 notas de un alumno como valores enteros. Luego mostrar el promedio teniendo en cuenta la parte decimal.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   int main()
5   {
6       int nota1, nota2, nota3;
7       float promedio;
8       printf("Ingrese la primera nota: ");
9       scanf("%i", &nota1);
10      printf("Ingrese la segunda nota: ");
11      scanf("%i", &nota2);
12      printf("Ingrese la tercera nota: ");
13      scanf("%i", &nota3);
14      promedio=(float)(nota1+nota2+nota3)/3;
15      printf("La nota media es %0.02f",promedio);
16      getch();
17      return 0;
18  }
```

Si ejecutamos este será el resultado:



Capítulo 181.- Operadores de asignación

Hemos visto hasta ahora que en el lenguaje C el operador de asignación =.
En C existen un grupo de operadores de asignación que pueden reemplazar a expresiones que hemos utilizado hasta ahora, por ejemplo si tenemos la siguiente expresión de acumulación:

```
suma=suma+valor;
```

Esto mismo es común en el lenguaje C escribiendo con la sintaxis:

```
suma+=valor;
```

El operador de asignación += suma a la variable indicada a la izquierda la expresión indicada a la derecha del operador +=.
Es obligatorio que los dos caracteres += estén sin espacios en blanco entre ellos.
Los operadores de asignación más comunes son:

```
+=
-=
*=
/=
%=
```

Para ver que se utiliza mucho esta sintaxis de asignación mostramos una de las tantas funciones del código fuente del sistema operativo Linux:

```
static size_t print_prefix(const struct printk_log *msg, bool syslog, char *buf)
{
        size_t len = 0;
        unsigned int prefix = (msg->facility << 3) | msg->level;

        if (syslog) {
                if (buf) {
                        len += sprintf(buf, "<%u>", prefix);
                } else {
                        len += 3;
                        if (prefix > 999)
                                len += 3;
                        else if (prefix > 99)
                                len += 2;
                        else if (prefix > 9)
                                len++;
```

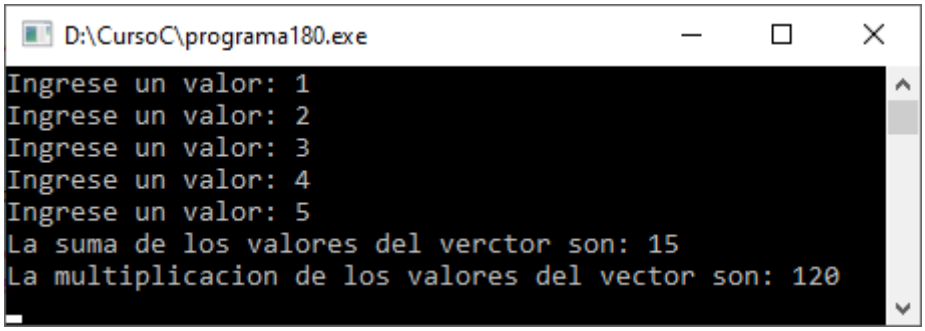
Problema

Confeccionar un programa que permita cargar un vector de 5 elementos enteros. Calcular la suma y el producto de todos sus elementos.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   void cargar(int v[5])
5   {
6       for (int x=0; x<5; x++)
7       {
8           printf("Ingrese un valor: ");
9           scanf("%i", &v[x]);
10      }
11  }
```

```c
#include<stdio.h>
#include<conio.h>

void cargar(int v[5])
{
    for (int x=0; x<5; x++)
    {
        printf("Ingrese un valor: ");
        scanf("%i", &v[x]);
    }
}

int sumarValores(int v[5])
{
    int sumar=0;
    for(int x=0; x<5; x++)
    {
        sumar+=v[x];
    }
    return sumar;
}

int multiplicarValores(int v[5])
{
    int multiplicar=1;
    for(int x=0; x<5; x++)
    {
        multiplicar*=v[x];
    }
    return multiplicar;
}

int main()
{
    int vector[5];
    cargar(vector);
    printf("La suma de los valores del verctor son: %i\n",
           sumarValores(vector));
    printf("La multiplicacion de los valores del vector son: %i\n",
           multiplicarValores(vector));
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 182.- Operador condicional ?: - 1

Este operador hace más compacta ciertas algoritmos en las que intervienen una estructura condicional if.

El sintaxis del operador condicional entonces es:

```
condición ? valor si cierto : valor si falso
```

Para ver que se utiliza esta sintaxis del operador condicional ?: mostramos una de las tantas funciones del código fuente del sistema operativo Linux:

```
static size_t print_prefix(const struct printk_log *msg, bool syslog, char *buf)
{
        size_t len = 0;
        unsigned int prefix = (msg->facility << 3) | msg->level;

        if (syslog) {
                if (buf) {
                        len += sprintf(buf, "<%u>", prefix);
                } else {
                        len += 3;
                        if (prefix > 999)
                                len += 3;
                        else if (prefix > 99)
                                len += 2;
                        else if (prefix > 9)
                                len++;
                }
        }

        len += print_time(msg->ts_nsec, buf ? buf + len : NULL);
        return len;
}
```

Problema

Cargar dos valores por teclado. Guardar en otra variable el mayor valor ingresado y mostrarlo.

```
1  #include<stdio.h>
2  #include<conio.h>
3
4  int main()
5  {
6      int num1, num2, mayor;
7      printf("Ingrese un valor: ");
8      scanf("%i", &num1);
9      printf("Ingrese otro valor: ");
10     scanf("%i", &num2);
11     mayor = (num1>=num2) ? num1 : num2;
12     printf("El valor mayor de los dos numeros es %i", mayor);
13     getch();
14     return 0;
15 }
16
```

Si ejecutamos este será el resultado:

```
D:\CursoC\programa181.exe
Ingrese un valor: 10
Ingrese otro valor: 2
El valor mayor de los dos numeros es 10
```

Tenemos la expresión:

```
mayor = (num1>=num2) ? num1 : num2;
```

El operador condición empieza después del operador =

```
(num1>=num2) ? num1 : num2;
```

La condición que se encuentra entre paréntesis:

```
(num1>=num2)
```

Puede generar un verdadero o un falso, en caso de ser verdadero lo indicado después del carácter ? se guarda en mayor. Si la condición genera un falso el contenido del num2 se guarda en mayor.

Esto mismo utilizando un if quedaría de la siguiente forma:

```
if (num1>num2)
{
    mayor=num1;
}
else
{
    mayor=num2;
}
```

Hay que tener en cuenta que el operador condicional ?: puede remplazar el uso de if en algunas situaciones y no en todas.

La sintaxis del operador condicional entonces es:

```
condición ? valor si cierto : valor si falso
```

Luego para usar el resultado que genera el operador condicional disponemos una sintaxis similar a esta:

```
variable = condición ? valor si cierto : valor si falso
```

Capítulo 183.- Operador condicional ?: - 2

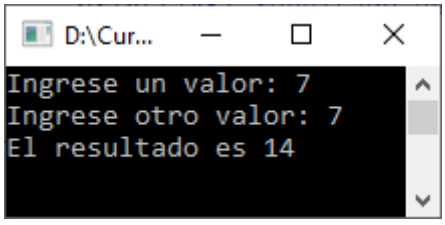
Problema propuesto

Ingresar por teclado dos números enteros. Si los valores son iguales sumarlos, sino multiplicarlos.

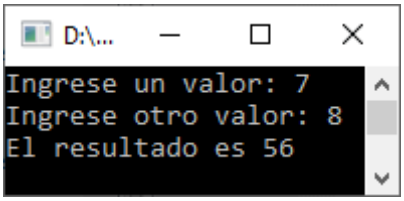
Emplear el operador condicional ?: para guardar en una variable el resultado.

```
#include<stdio.h>
#include<conio.h>

int main()
{
    int num1, num2, resultado;
    printf("Ingrese un valor: ");
    scanf("%i", &num1);
    printf("Ingrese otro valor: ");
    scanf("%i", &num2);
    resultado = (num1==num2) ? num1+num2 : num1*num2;
    printf("El resultado es %i\n", resultado);
    getch();
    return 0;
}
```

Vamos a ejecutar introduciendo dos números iguales:



Ejecutamos de nuevo introduciendo dos números distintos:



Capítulo 184.- Estructura condicional switch – 1

Hasta ahora siempre que se presentan bifurcaciones en un programa lo resolvemos con una estructura condicional if, también vimos en un concepto anterior el operador condicional ?:

Ahora veremos que en el lenguaje C existe una estructura condicional que nos permite seleccionar un camino de un conjunto de caminos dependiendo de una condición.

La estructura switch remplaza en muchas situaciones, si bien no en todas, a un conjunto de if.

Veremos varios ejemplos y la sintaxis propuestas del switch.

Para ver que se utiliza esta estructura condicional podemos mostrar una de las tantas funciones del código fuente del sistema operativo Linux donde la implementa:

```
static int find_poly_roots(struct bch_control *bch, unsigned int k,
                           struct gf_poly *poly, unsigned int *roots)
{
        int cnt;
        struct gf_poly *f1, *f2;

        switch (poly->deg) {
                /* handle low degree polynomials with ad hoc techniques */
        case 1:
                cnt = find_poly_deg1_roots(bch, poly, roots);
                break;
        case 2:
                cnt = find_poly_deg2_roots(bch, poly, roots);
                break;
        case 3:
                cnt = find_poly_deg3_roots(bch, poly, roots);
                break;
        case 4:
                cnt = find_poly_deg4_roots(bch, poly, roots);
                break;
        default:
                /* factor polynomial using Berlekamp Trace Algorithm (BTA) */
                cnt = 0;
                if (poly->deg && (k <= GF_M(bch))) {
                        factor_polynomial(bch, k, poly, &f1, &f2);
                        if (f1)
                                cnt += find_poly_roots(bch, k+1, f1, roots);
                        if (f2)
                                cnt += find_poly_roots(bch, k+1, f2, roots+cnt);
                }
                break;
        }
        return cnt;
}
```

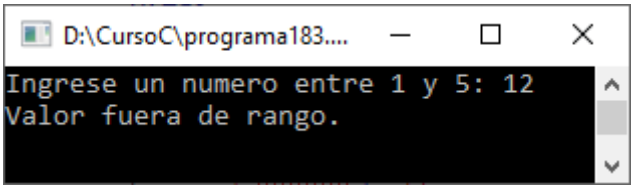
Problema

Ingresar por teclado un valor entero comprendido entre 1 y 5. Mostrar en castellano el valor ingresado o un mensaje indicando que se cargó un valor fuera de rango.

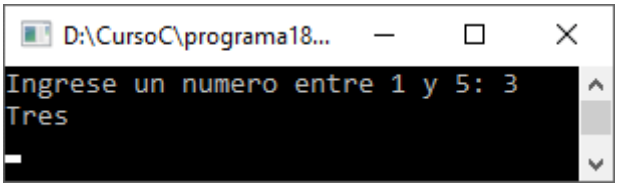
```
1   #include<stdio.h>
2   #include<conio.h>
3
4   int main()
5   {
6       int num;
7       printf("Ingrese un numero entre 1 y 5: ");
```

```
8       scanf("%i", &num);
9       switch(num){
10      case 1:
11          printf("Uno\n");
12          break;
13      case 2:
14          printf("Dos\n");
15          break;
16      case 3:
17          printf("Tres\n");
18          break;
19      case 4:
20          printf("Cuatro\n");
21              break;
22      case 5:
23          printf("Cinco\n");
24          break;
25      default:
26          printf("Valor fuera de rango.\n");
27          break;
28      }
29      getch();
30      return 0;
31  }
```

Vamos a ejecutar introduciendo el valor 12:



Ejecutamos de nuevo introduciendo un valor entre 1 y 5.



Para realizar el mismo resultado utilizando el if:

```
#include<stdio.h>
#include<conio.h>

int main()
{
    int valor;
    printf("Ingrese un valor entre 1 y 5:");
    scanf("%i",&valor);
    if (valor==1)
    {
        printf("Uno");
    }
    else
```

```
    {
        if (valor==2)
        {
            printf("Dos");
        }
        else
        {
            if (valor==3)
            {
                printf("Tres");
            }
            else
            {
                if (valor==4)
                {
                    printf("Cuatro");
                }
                else
                {
                    if (valor==5)
                    {
                        printf("Cinco");
                    }
                    else
                    {
                        printf("El valor esta fuera de rango");
                    }
                }
            }
        }
    }
    getch();
    return 0;
}
```

Capítulo 185.- Estructura condicional switch – 2

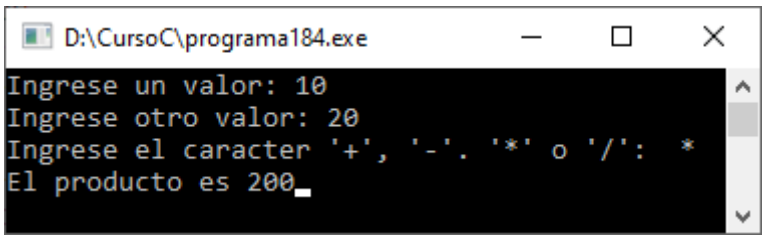
Problema

Ingresar por teclado dos valores enteros. Seguidamente solicitar el ingreso de un carácter '+', '-', '*' o '/'. Dependiendo del operador ingresado sumar, restar, multiplicar o dividir los valores ingresados.

Si no se ingresa un operador válido no hace nada.

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int num1, num2;
    char caracter;
    printf("Ingrese un valor: ");
    scanf("%i", &num1);
    printf("Ingrese otro valor: ");
    scanf("%i", &num2);
    printf("Ingrese el caracter '+', '-'. '*' o '/':  ");
    scanf(" %c", &caracter);
    switch(caracter){
        case '+':
                printf("La suma es %i", num1+num2);
                break;
        case '-':
                printf("La resta es %i", num1-num2);
                break;
        case '*':
                printf("El producto es %i", num1*num2);
                break;
        case '/':
                printf("La division es %i", num1/num2);
                break;
        default:
                printf("el operador introducido es erroneo.");
                break;
    }
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



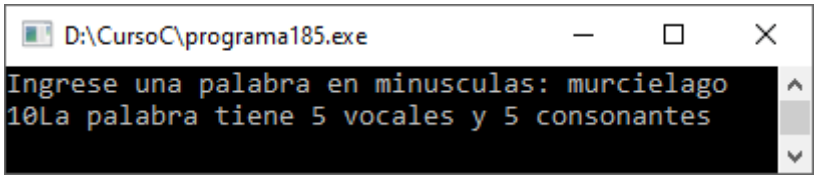
Capítulo 186.- Estructura condicional switch – 3

Problema

Ingresar por teclado una palabra en minúsculas y luego contar la cantidad de vocales y consonantes que tiene:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

int main()
{
    char palabra[41];
    int longitud;
    int vocales=0;
    int consonantes=0;
    printf("Ingrese una palabra en minusculas: ");
    gets(palabra);
    longitud=strlen(palabra);
    printf("%i",longitud);
    for (int x=0; x<longitud; x++)
    {
        switch(palabra[x]){
            case 'a':
                vocales++;
                break;
            case 'e':
                vocales++;
                break;
            case 'i':
                vocales++;
                break;
            case 'o':
                vocales++;
                break;
            case 'u':
                vocales++;
                break;
            default:
                consonantes++;
                break;
        }
    }
    printf("La palabra tiene %i vocales y %i consonantes",
           vocales, consonantes);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Otra forma correcta de realizarlo es:

```c
#include<stdio.h>
#include<conio.h>
#include<string.h>

int main()
{
    char palabra[41];
    int longitud;
    int vocales=0;
    int consonantes=0;
    printf("Ingrese una palabra en minusculas: ");
    gets(palabra);
    longitud=strlen(palabra);
    printf("%i",longitud);
    for (int x=0; x<longitud; x++)
    {
        switch(palabra[x]){
            case 'a':
            case 'e':
            case 'i':
            case 'o':
            case 'u':
                vocales++;
                break;
            default:
                consonantes++;
                break;
        }
    }
    printf("La palabra tiene %i vocales y %i consonantes",
           vocales, consonantes);
    getch();
    return 0;
}
```

Si ejecutamos el resultado será el mismo.

Capítulo 187.- Estructura condicional switch – 4

Problema propuesto

Confeccionar un programa que permita administrar un vector de 5 enteros.

Mostrar un menú de opciones:

1- Cargar el vector.
2- Mostrarlo.
3- Imprimir mayor.
4- Imprimir menor.
5- Finalizar el programa.

Utiliza un switch para la selección de la opción y la llamada a la función repetitiva.

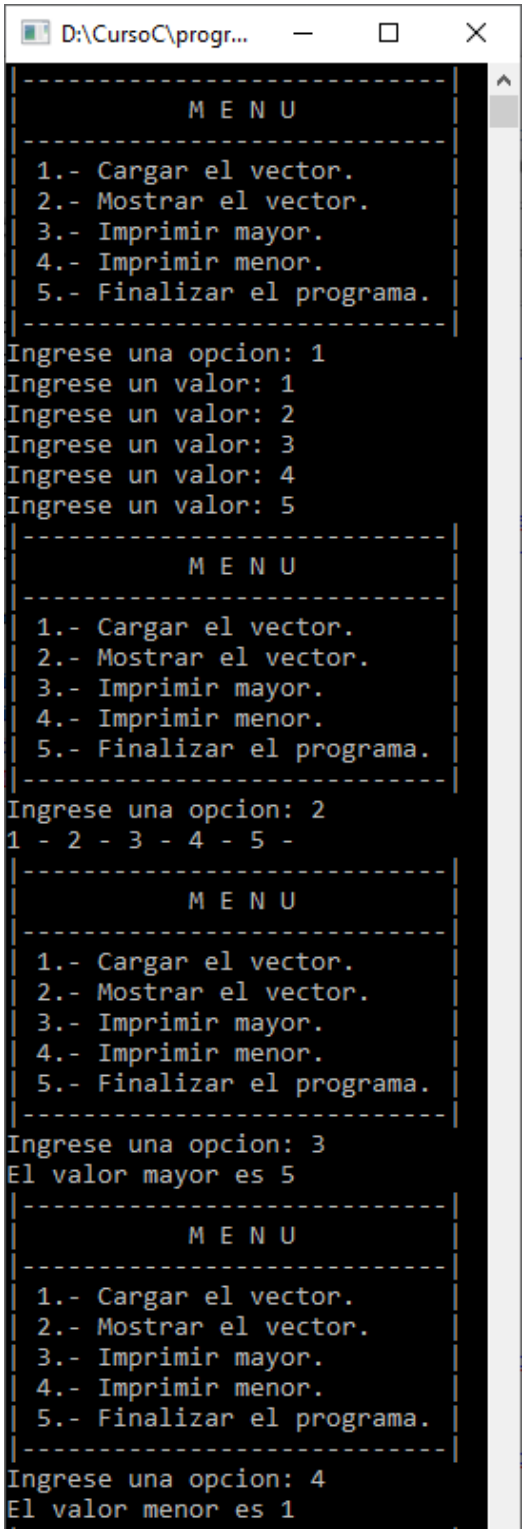
```
#include<stdio.h>
#include<conio.h>

void cargar(int vec[5])
{
    for (int x=0; x<5; x++)
    {
        printf("Ingrese un valor: ");
        scanf("%i", &vec[x]);
    }
}

void imprimir(int vec[5])
{
    for (int x=0; x<5; x++)
    {
        printf("%i - ", vec[x]);
    }
    printf("\n");
}

void mayor(int vec[5])
{
    int mayor=vec[0];
    for (int x=1; x<5; x++)
    {
        if(vec[x]>mayor)
        {
            mayor=vec[x];
        }
    }
    printf("El valor mayor es %i\n", mayor);
}
```

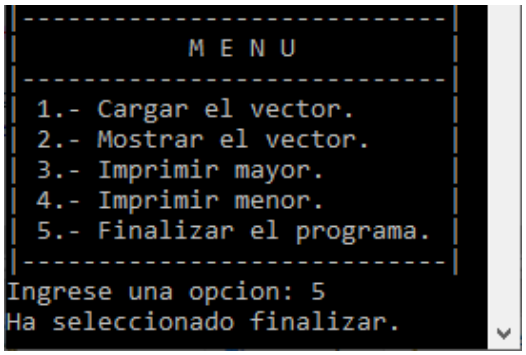
```
34
35  void menor(int vec[5])
36  {
37      int menor=vec[0];
38      for (int x=1; x<5; x++)
39      {
40          if(vec[x]<menor)
41          {
42              menor=vec[x];
43          }
44      }
45      printf("El valor menor es %i\n", menor);
46  }
47
48  void menu()
49  {
50      printf("|----------------------------|\n");
51      printf("|           M E N U          |\n");
52      printf("|----------------------------|\n");
53      printf("| 1.- Cargar el vector.      |\n");
54      printf("| 2.- Mostrar el vector.     |\n");
55      printf("| 3.- Imprimir mayor.        |\n");
56      printf("| 4.- Imprimir menor.        |\n");
57      printf("| 5.- Finalizar el programa. |\n");
58      printf("|----------------------------|\n");
59  }
```

```
60  int main()
61  {
62      int vector[5];
63      int opcion;
64      do
65      {
66         menu();
67         printf("Ingrese una opcion: ");
68         scanf("%i", &opcion);
69         switch(opcion){
70              case 1:
71                  cargar(vector);
72                  break;
73              case 2:
74                  imprimir(vector);
75                  break;
76              case 3:
77                  mayor(vector);
78                  break;
79              case 4:
80                  menor(vector);
81                  break;
82              case 5:
83                  printf("Ha seleccionado finalizar.\n");
84                  break;
85              default:
86                  printf("Ha seleccionado una opcion incorrecta.\n");
87                  break;
88         }
89      }while (opcion!=5);
```

```
90
91          getch();
92          return 0;
93      }
```

Si ejecutamos este será el resultado:




|----------------------------|
M E N U
1.- Cargar el vector.
2.- Mostrar el vector.
3.- Imprimir mayor.
4.- Imprimir menor.
5.- Finalizar el programa.
Ingrese una opcion: 5
Ha seleccionado finalizar.

Capítulo 188.- Comandos break y continue dentro de estructuras repetitivas – 1

Hay dos comandos en el lenguaje C que modifican un ciclo repetitivo for, while o do while.

break

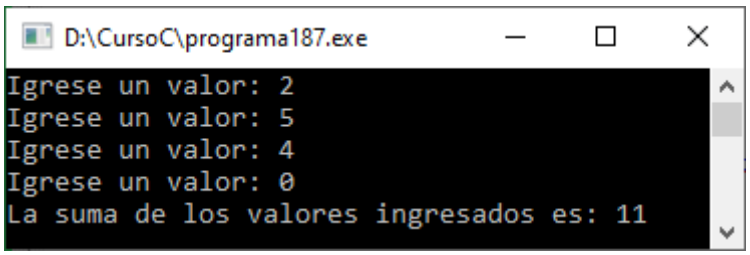
El primer comando se llama break y lo que hace es finalizar el ciclo repetitivo inmediatamente sin tener que finalizar la condición del ciclo.

Problema

Realizar la carga de enteros por teclado y sumarlos, finalizar cuando se ingrese el valor cero o se hayan cargado 10 valores.

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int valor;
    int suma=0;
    for (int x=0; x<10; x++)
    {
        printf("Igrese un valor: ");
        scanf("%i", &valor);
        if (valor==0)
            break;
        suma+=valor;
    }
    printf("La suma de los valores ingresados es: %i", suma);

    getch();
    return 0;
}
```

Vamos a ejecutar ingresando un 0 en el valor 4:



Si no introducimos el valor 0 nos preguntará por 10 valores.

Capítulo 189.- Comandos break y continue dentro de estructuras repetitivas – 2

Hay dos comandos en el lenguaje C que modifican en ciclo for, while o do while.

continue

El comando continue cuando se ejecuta vuelve al principio del ciclo repetitivo.

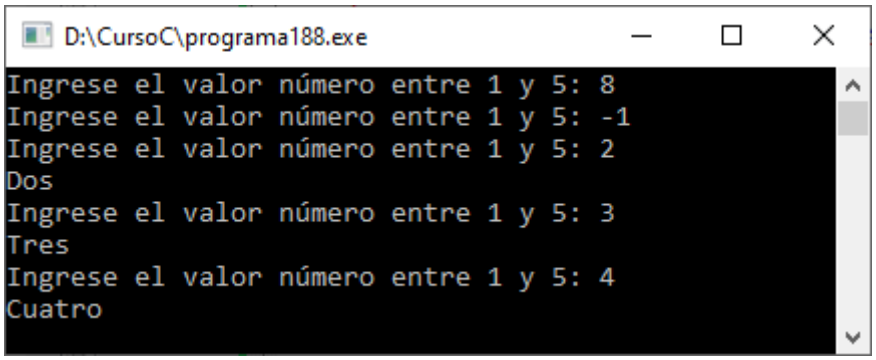
Problema

Realizar la carga de 5 enteros por teclado. Mostrar por pantalla en castellano dicho número si está comprendido entre 1 y 5.

Utilizar el comando continue en caso que el valor ingresado no se encuentre entre 1 y 5.

```c
#include<stdio.h>
#include<conio.h>
#include<locale.h>

int main()
{
    setlocale(LC_ALL, "");
    int numero;
    for (int x=0; x<5; x++)
    {
        printf("Ingrese el valor número entre 1 y 5: ");
        scanf("%i", &numero);
        if(numero>5 || numero<1)
            continue;
        switch(numero){
            case 1:
                printf("Uno");
                break;
            case 2:
                printf("Dos");
                break;
            case 3:
                printf("Tres");
                break;
            case 4:
                printf("Cuatro");
                break;
            case 5:
                printf("Cinco");
                break;
        }
        printf("\n");
    }
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Los valores que están fuera del rango 1-5 no muestran el número en letras.

Capítulo 190.- Comandos break y continue dentro de estructuras repetitivas – 3

Ciclos con condiciones siempre verdaderas

Podemos disponer un ciclo repetitivo while, foor o do/while con una condición que siempre será verdadera. Luego en estos casos es obligatorio para que el ciclo finalice que haya un comando break en su interior.

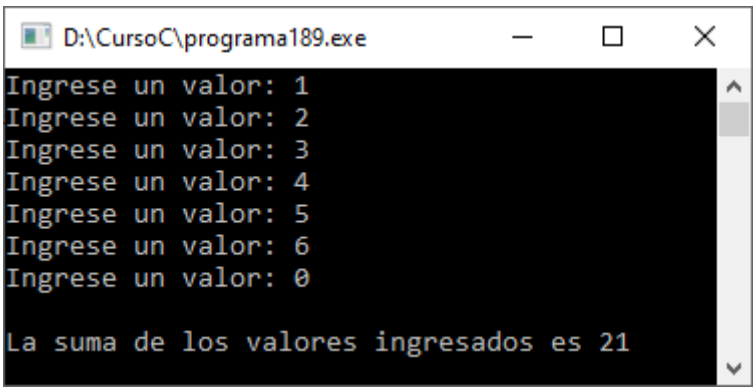
Problema

Ingresar valores por teclado y sumarlos. Finalizar al ingresar el cero.

```
1  #include<stdio.h>
2  #include<conio.h>
3
4  int main()
5  {
6      int valor, suma=0;
7      while(1)
8      {
9          printf("Ingrese un valor: ");
10         scanf("%i", &valor);
11         if(valor==0)
12             break;
13         suma+=valor;
14     }
15     printf("\n");
16     printf("La suma de los valores ingresados es %i", suma);
17     getch();
18     return 0;
19 }
```

1 es igual a verdadero y 0 es igual a falso.

Si ejecutamos este será el resultado, finalizará cuando ingresemos el valor 0.



Capítulo 191.- Comando goto – 1

Otro comando que tiene el lenguaje C es el goto. Este comando nos permite saltar en forma incondicional al cualquier otra parte de la misma función.

La sentencia goto no es aconsejada en la metodología de la programación estructurada pero su uso en casos especiales nos permite crear código más conciso.

Un caso común donde se la utiliza es cuando queremos salir de una serie de for anidados a un bloque de la misma función pero fuera de dicho for (tener en cuenta que el comando break solo sale del for que la contiene).

Si consultamos el código fuente del sistema operativo Linux veremos que utiliza el comando goto en numerosas funciones:

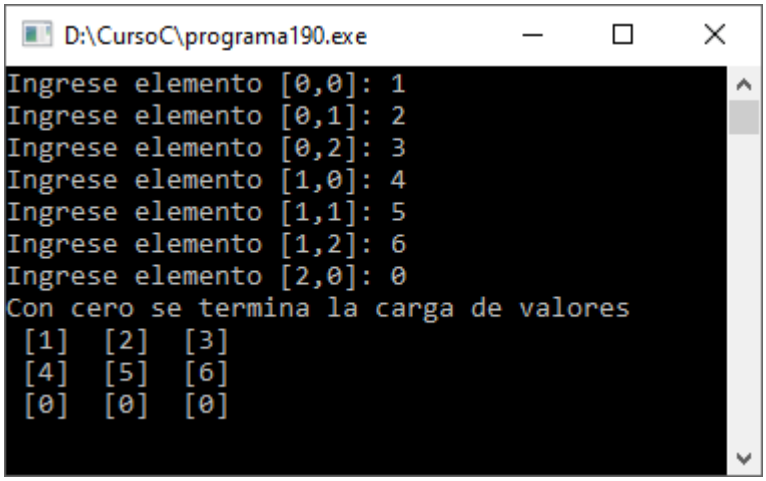
```
static struct audit_entry *audit_find_rule(struct audit_entry *entry,
                                           struct list_head **p)
{
        struct audit_entry *e, *found = NULL;
        struct list_head *list;
        int h;

        if (entry->rule.inode_f) {
                h = audit_hash_ino(entry->rule.inode_f->val);
                *p = list = &audit_inode_hash[h];
        } else if (entry->rule.watch) {
                /* we can't know the inode number, so must walk entire
                for (h = 0; h < AUDIT_INODE_BUCKETS; h++) {
                        list = &audit_inode_hash[h];
                        list_for_each_entry(e, list, list)
                                if (!audit_compare_rule(&entry->rule,
                                        found = e;
                                        goto out;
                                }
                }
                goto out;
        } else {
                *p = list = &audit_filter_list[entry->rule.listnr];
        }

        list_for_each_entry(e, list, list)
                if (!audit_compare_rule(&entry->rule, &e->rule)) {
                        found = e;
                        goto out;
                }

out:
        return found;
}
```

Problema

Definir e inicializar una matriz 3x3 con valores 0. Implementar la carga por teclado de valores enteros, si se carga algún momento un cero no permitir ingresar más valores.

```c
#include<stdio.h>
#include<conio.h>

#define FILAS 3
#define COLUMNAS 3


void cargar(int mat[FILAS][COLUMNAS])
{
    for (int f=0; f<3; f++)
    {
        for (int c=0; c<3; c++)
        {
            printf("Ingrese elemento [%i,%i]: ",f,c);
            scanf("%i", &mat[f][c]);
            if (mat[f][c]==0)
            {
                goto salir;
            }
        }
    }
    return; // Si finaliza los dor for sale de la función.
    salir:printf("Con cero se termina la carga de valores\n");
}

void imprimir(int mat[FILAS][COLUMNAS])
{
    for(int f=0; f<3; f++)
    {
        for (int c=0; c<3; c++)
        {
            printf(" [%i] ", mat[f][c]);
        }
        printf("\n");
    }
}
int main()
{
    int matriz[FILAS][COLUMNAS]={{0,0,0},{0,0,0},{0,0,0}};
    cargar(matriz);
    imprimir(matriz);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Luego de la palabra clave goto debemos indicar el nombre de una sección de nuestra función donde debe seguir el flujo de nuestro programa:

```
if (mat[f][c]==0)
{
    goto salir;
}
```

En otra parte de la función que puede estar antes o después del comando goto indicamos el mismo nombre de la etiqueta seguida de dos puntos:

```
salir:printf("Con cero se termina la carga de valores\n");
```

En nuestro programa en particular se el operador nunca carga un valor cero luego de terminar la ejecución de los dos for encuentra el comando return saliendo de la función y no ejecutando la instrucción posterior a la etiqueta salir:

Si no disponemos la instrucción return y el operador carga todos los valores distintos a cero al finalizar los dos for ejecuta la punción printf sin importar que está precedida por la etiqueta salir.

Capítulo 192.- Comando goto – 2

Problema propuesto

Definir una matriz de 5 filas por 10 columnas. Cargar valores aleatorios comprendidos entre 1 y 10.

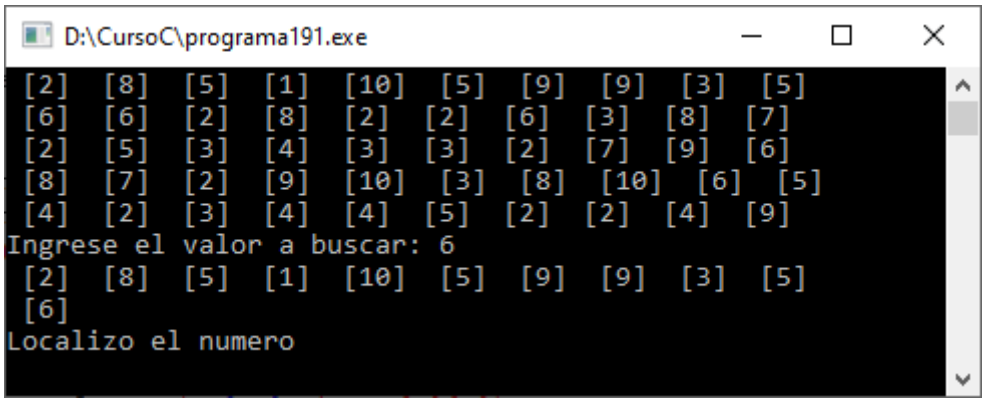
Ingresar por teclado un número y verificar si se encuentra en la matriz. No buscar más si se lo ha encontrado, salir de los ciclos mediante un goto.

```
#include<stdio.h>
#include<conio.h>

#define FILA 5
#define COLUMNA 10

void cargar(int mat[FILA][COLUMNA])
{
    for(int f=0; f<FILA; f++)
    {
        for(int c=0; c<COLUMNA; c++)
        {
            mat[f][c]=rand()% 10 + 1;
        }
    }
}

void imprimir(int mat[FILA][COLUMNA])
{
    for(int f=0; f<FILA; f++)
    {
        for(int c=0; c<COLUMNA; c++)
        {
            printf(" [%i] ", mat[f][c]);
        }
        printf("\n");
    }
}
```

```

void buscar(int mat[FILA][COLUMNA])
{
    int valorBuscar;
    printf("Ingrese el valor a buscar: ");
    scanf("%i", &valorBuscar);
    for(int f=0; f<FILA; f++)
    {
        for(int c=0; c<COLUMNA; c++)
        {
            printf(" [%i] ", mat[f][c]);
            if (valorBuscar==mat[f][c])
                goto salir;
        }
        printf("\n");
    }
```

```
45          return;
46          salir:printf("\nLocalizo el numero");
47      }
48
49      int main()
50      {
51          int matriz[FILA][COLUMNA];
52          cargar(matriz);
53          imprimir(matriz);
54          buscar(matriz);
55          getch();
56          return 0;
57      }
58
```

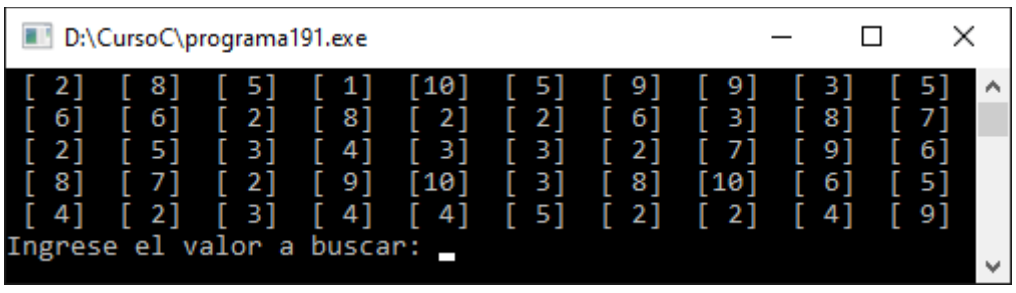
Si ejecutamos este será el resultado:



Si queremos que el número %i tenga dos dígitos realizaremos lo siguiente:

```
18      void imprimir(int mat[FILA][COLUMNA])
19      {
20          for(int f=0; f<FILA; f++)
21          {
22              for(int c=0; c<COLUMNA; c++)
23              {
24                  printf(" [%2i] ", mat[f][c]);
25              }
26              printf("\n");
27          }
28      }
```

Cuando ejecutemos:



Capítulo 193.- función exit para terminar un programa

La función exit interrumpe la ejecución del programa en forma inmediata y retorna un entero al proceso que llamó al programa (normalmente el valor devuelto lo recibe el sistema operativo).

La sintaxis de esta función es:

```
void exit(int codigo)
```

Para poder llamar a esta función debemos importar el archivo de inclusión:

```
#include<stdlib.h>
```

El valor devuelto por convención es un 0 si el programa finaliza en forma correcta o un valor distinto a cero indicando distintos códigos de error que los puede interpretar quién ejecutó el programa.

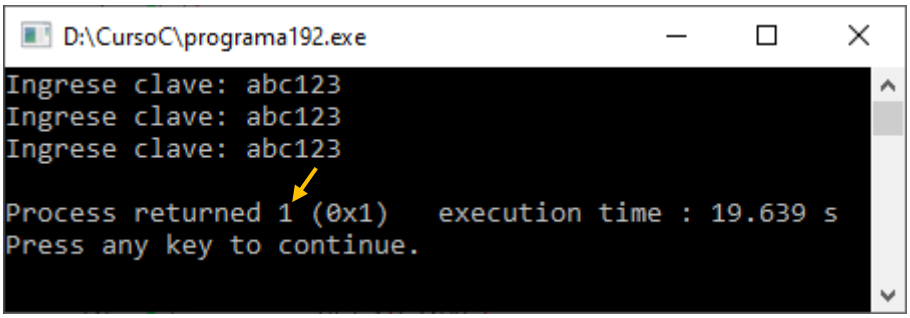
Es importante notar que a exit la podemos llamar en cualquier parte de nuestro programa e inmediatamente el programa se detendrá y no ejecutará más comandos del mismo.

Problema

Desarrollar un programa que permita ingresar una clave de usuario. Si la ingresa en forma incorrecta 3 veces proceder a detener el programa en forma inmediatamente.

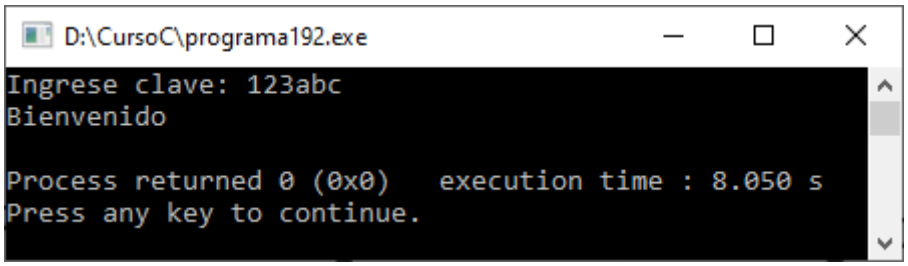
```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>

void ingresoClave()
{
    char clave[50];
    int intentos=0;
    do{
        if(intentos==3)
        {
            exit(1);
        }
        printf("Ingrese clave: ");
        gets(clave);
        intentos++;

    }while(strcmp(clave,"123abc")!=0);
}

int main()
{

    ingresoClave();
    printf("Bienvenido\n");
    getch();
    return 0;
}
```

Vamos a ejecutar e introducir la clave 3 veces incorrectamente.



Nos retorna un 1.

Ahora vamos a ejecutar de nuevo e introducir la contraseña correcta:



Retorna un 0.

Capítulo 194.- Función system

La función system nos permite ejecutar otro programa. Debemos pasa como parámetro una cadena con el nombre del programa a ejecutar, debe ser un programa ejecutable el mismo.

Problema

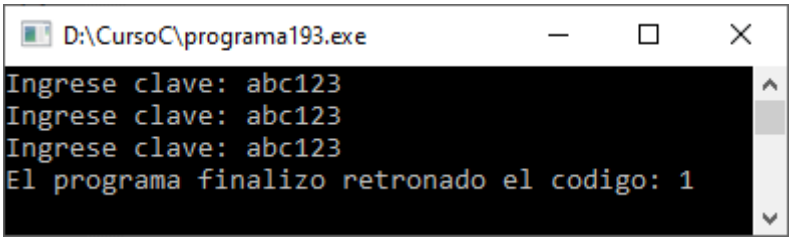
Implementar un programa que llame al programa del ingreso de la clave que hicimos en el capítulo anterior (Deberás llamar el programa con terminación .exe) mediante la función system.

Imprimir el código devuelto.

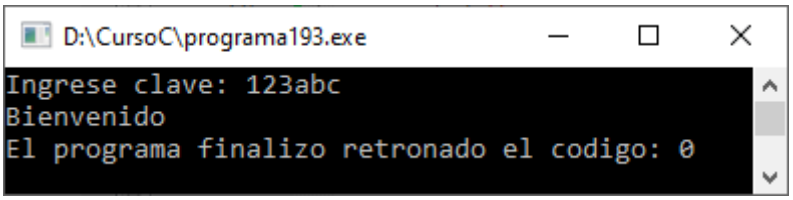
```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

int main()
{
    int codigo;
    codigo=system("programa192.exe");
    printf("El programa finalizo retronado el codigo: %i", codigo);
    getch();
    return 0;
}
```

En nuestro ejemplo el programa que llamamos es programa192.exe.

Vamos a ejecutar e introducir la clave erróneamente tres veces:



Ejecutamos de nuevo y esta vez introducimos la clave correcta:



Para poder probar este programa ya debimos compilar el programa y haber generado el exe.

Los archivos exe se deben encontrar en la misma carpeta ya que la función system le pasamos solamente el nombre del archivo a ejecutar y no el path donde se encuentra:

```
codigo=system("programa192.exe");
```

Si indicamos el path el código quedaría:

```
codigo=system("c:\\programasc\\programa192.exe");
```

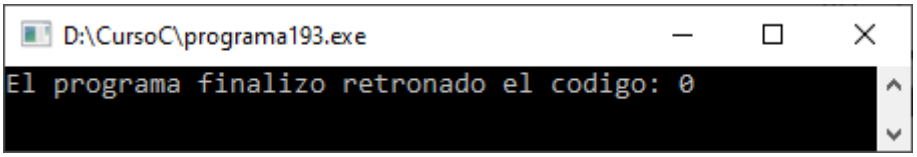
Es importante notar que cuando finaliza el programa, el valor devuelto se almacena en la variable codigo:

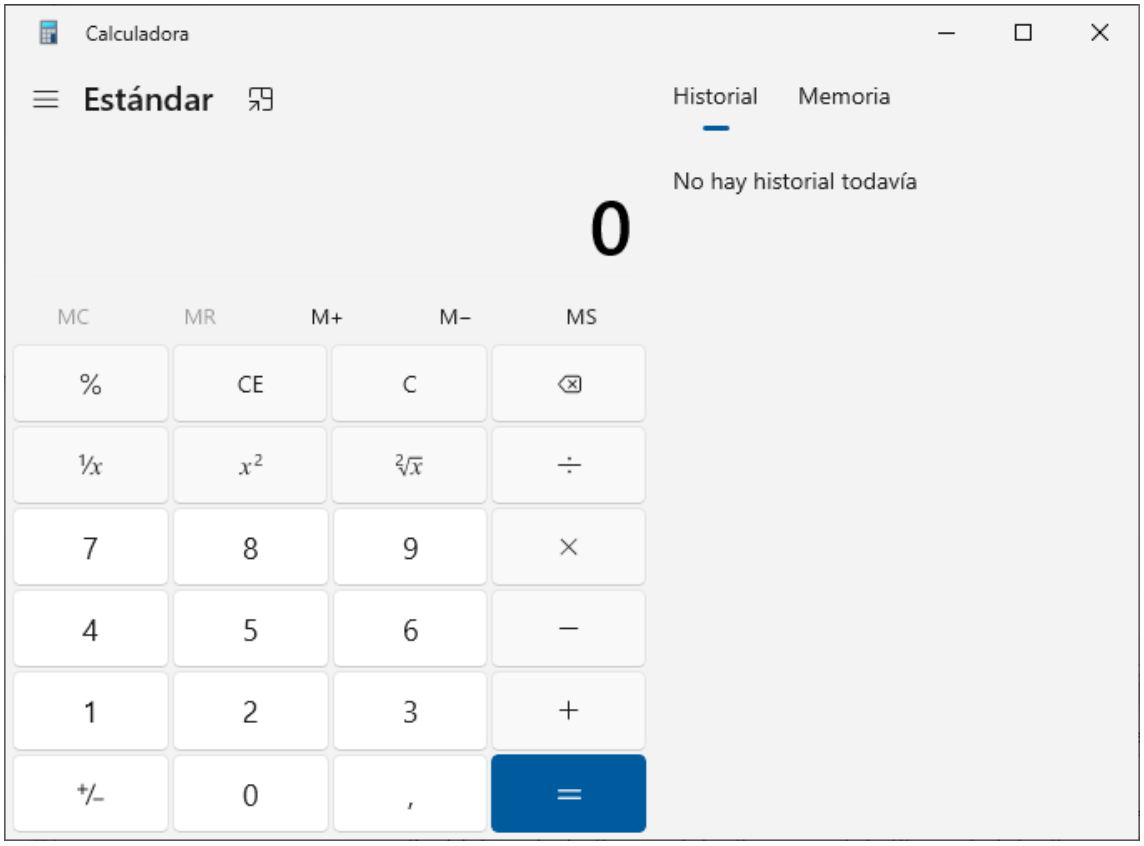
```
codigo=system("programa192.exe");
printf("El programa finalizo retronado el codigo: %i", codigo);
```

Podemos mencionar que si finaliza por la ejecución del exit el valor devuelto es un 1 y si finaliza cuando termina la función main retorna un 0.

```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<stdlib.h>
4
5   int main()
6   {
7       int codigo;
8       codigo=system("calc.exe");
9       printf("El programa finalizo retronado el codigo: %i", codigo);
10      getch();
11      return 0;
12  }
```

Le estamos pidiendo que ejecute la calculadora de Windows.





Capítulo 195.- Definición de constantes const

Si necesitamos almacenar un valor en una variable que nunca será cambiado, el lenguaje C nos permite definir una constante.

```
const int impuesto=21;
```

La definición como constante del identificador impuesto asegura que cualquier intento de asignarle otro valor generará un error cuando compilemos el programa:

```
impuesto=25;
```

Se puede definir constantes de cualquier tipo de dato que nos provee el lenguaje C.
Una constante a diferencia de una macro ocupa un espacio durante la ejecución del programa:

```
#define PI 3.1416

const float pi=3.1416
```

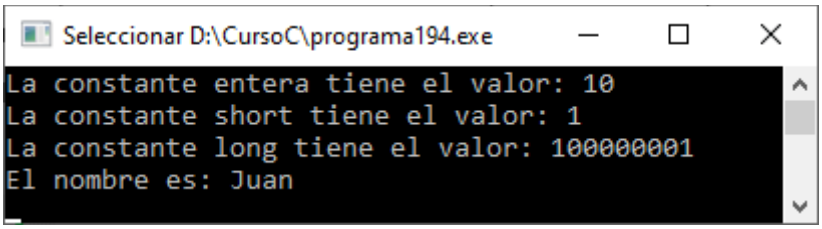
Es importante notar que pi reserva espacio para un valor float durante la ejecución del programa, en cambio la macro PI cuando compilamos se sustituye todas las partes que aparece PI por el valor 3.1416.

Problema

Definir constantes de distinto tipo e imprimir sus valores.

```
#include<stdio.h>
#include<conio.h>

struct persona{
    char nombre[50];
    int edad;
};

int main()
{
    const int valor1=10;
    const short valor2=1;
    const long valor3=100000001;
    const struct persona per1={"Juan", 34};
    const char letra='A';
    const char titulo[7]="Inicio";
    printf("La constante entera tiene el valor: %i\n", valor1);
    printf("La constante short tiene el valor: %i\n", valor2);
    printf("La constante long tiene el valor: %li\n", valor3);
    printf("El nombre es: %s\n", per1.nombre);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 196.- Modificador const en los parámetros de una función

Cuando un parámetro no debe ser modificado en conveniente agregarle el modificador const para evitar que por descuido le asignemos un valor.

Si consultamos el código fuente del sistema operativo Linux veremos que utiliza el modificador const en numerosas funciones:

```
static void copy_workqueue_attrs(struct workqueue_attrs *to,
                                 const struct workqueue_attrs *from)
{
        to->nice = from->nice;
        cpumask_copy(to->cpumask, from->cpumask);
        /*
         * Unlike hash and equality test, this function doesn't ignore
         * ->no_numa as it is used for both pool and wq attrs.  Instead,
         * get_unbound_pool() explicitly clears ->no_numa after copying.
         */
        to->no_numa = from->no_numa;
}
```

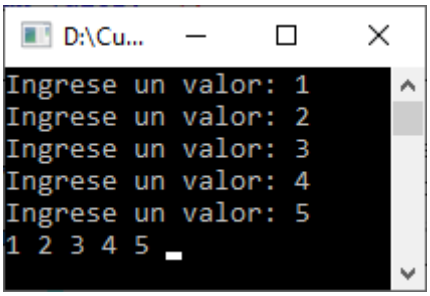
Problema

Cargar un vector de 5 enteros y luego imprimirlo.

```
1  #include<stdio.h>
2  #include<conio.h>
3
4  #define CANTIDAD 5
5
6  void cargar(int vec[CANTIDAD])
7  {
8      for (int f=0; f<CANTIDAD; f++)
9      {
10         printf("Ingrese un valor: ");
11         scanf("%i", &vec[f]);
12     }
13 }
14
15 void imprimir(const int vec[CANTIDAD])
16 {
17     for(int f=0; f<CANTIDAD; f++)
18     {
19         printf("%i ",vec[f]);
20     }
21 }
22
23 int main()
24 {
25     int vec[CANTIDAD];
26     cargar(vec);
27     imprimir(vec);
28     getch();
29     return 0;
30 }
31
```

Si en esta función intentamos cambiar los valores del vector nos dará un error.

Si ejecutamos este será el resultado:



Capítulo 197.- Empleo de llaves opcionales en estructuras condicionales y repetitivas

Hasta ahora todos los problemas planteados hemos utilizado llaves de apertura y cierre para indicar los bloques de las estructuras condicionales y repetitivas.

Cuando uno comienza a programar es más conveniente concentrarse en la lógica y sintaxis básica que nos provee el lenguaje que ver todas las variantes.

Ahora ya podemos introducir la sintaxis alternativa cuando tenemos una sola instrucción dentro de una estructura condicional o repetitiva.

Las llaves de apertura y cierre de una estructura condicional y repetitiva en el lenguaje C es opcional si hay una sola instrucción.

Si consultamos el código fuente del sistema operativo Linux veremos que cuando hay estructuras repetitivas con una sola instrucción no se disponen las llaves ya que son opcionales:

```
int cgroup_add_dfl_cftypes(struct cgroup_subsys *ss, struct cftype
{
        struct cftype *cft;

        for (cft = cfts; cft && cft->name[0] != '\0'; cft++)
                cft->flags |= __CFTYPE_ONLY_ON_DFL;
        return cgroup_add_cftypes(ss, cfts);
}
```

Pero si la estructura repetitiva tiene más de una instrucción veremos las llaves que encierran el bloque a repetir:

```
static int allocate_cgrp_cset_links(int count, struct list_head *tmp
{
        struct cgrp_cset_link *link;
        int i;

        INIT_LIST_HEAD(tmp_links);

        for (i = 0; i < count; i++) {
                link = kzalloc(sizeof(*link), GFP_KERNEL);
                if (!link) {
                        free_cgrp_cset_links(tmp_links);
                        return -ENOMEM;
                }
                list_add(&link->cset_link, tmp_links);
        }
        return 0;
```

Problema

Cargar dos enteros y mostrar el mayor.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   int main()
5   {
6       int v1, v2;
7       printf("Ingrese primer valor: ");
```

```
8       scanf("%i", &v1);
9       printf("Ingrese segundo valor: ");
10      scanf("%i", &v2);
11      if (v1>v2)
12          printf("El valor mayor es %i", v1);
13      else
14          printf("El valor mayor es %i", v2);
15      getch();
16      return 0;
17  }
```

Si por cualquier situación hay que agregar una segunda instrucción dentro del verdadero o falso debemos obligatoriamente encerrar entre llaves el bloque.

Problema

Mostrar los números del 1 al 10.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   int main()
5   {
6       int x;
7       for(x=1; x<=10;x++)
8           printf("%i\n",x);
9       getch();
10      return 0;
11  }
```

La estructura repetitiva for tiene una sola instrucción por lo que esta permitido disponer dicha actividad sin encerrarla entre llaves:

```
for(x=1;x<=10;x++)
    printf("%i\n",x);
```

Recordemos que veníamos resolviendo con la sintaxis:

```
for(x=1;x<=10;x++)
{
    printf("%i\n",x);
}
```

Como vemos el código queda mucho más conciso sin las llaves.

Pero imaginemos que tenemos que modificar el programa para que muestre una línea de 10 guiones que separe cada número, si no tenemos en claro la existencia de las llaves podríamos pensar que si disponemos el código siguiente funcionaría:

```
for(x=1;x<=10;x++)
    printf("%i\n",x);
    printf("----------\n");
```

Aunque ahora ya sabemos perfectamente que todo lo que se permite dentro del for debe estar encerrado entre llaves si hay más de una instrucción:

```
for(x=1;x<=10;x++)
{
    printf("%i\n",x);
    printf("----------\n");
}
```

No introducimos esta sintaxis desde los primeros días que comenzamos a aprender a programar para evitar errores como el anterior. Dijimos que toda estructura repetitiva o condicional debe estar encerrada entre llaves.

Por último tener en cuenta que las llaves son opcionales solo para los bloques de las estructuras condicionales y repetitivas. Una función que tiene una única instrucción se debe tener llaves de apertura y cierre de dicha función.

Capítulo 198.- Declaración de enumeraciones – 1

Una enumeración es un tipo de dato que creamos asociando un conjunto de enteros con un conjunto de literales.

Veamos ejemplos de declaración de enumeraciones:

```
enum operaciones {SUMA, RESTA, MULTIPLICACION, DIVISION};
enum respuesta {NO, SI};
enum diassemana {LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO};
enum tipoventa {MAYORISTAS, MINORISTAS};
enum dispositivos {TECLADO, MOUSE, CAMARA};
enum boolean {FALSE, TRUE};
```

Como vemos la declaración de un tipo de dato enumerado se declara mediante la palabra clave enum y seguidamente el nombre del tipo de datos, luego entre llaves indicamos todos los valores que pueden almacenar una variable de este tipo. La primera constante o literal indicado toma el valor cero, la segunda el valor 1 y así sucesivamente.

Podemos declarar la emulación con la sintaxis:

```
enum operaciones {SUMA=0, RESTA=1, MULTIPLICACION=2, DIVISION=3};
```

Pero es opcional si queremos iniciar a partir de cero la emulación el propio compilador lo hace.

Luego para definir una variable de tipo enumeración utilizamos la sintaxis:

```
enum operaciones op;
```

Definimos la variable op de tipo operaciones. Luego en la variable op podemos almacenar cualquiera de los cuatros literales definidos en la emulación:

```
op=MULTIPLICACION;
```

Las emulaciones tienen por objetivo que el programa sea más legible para cuando tengamos que hacer cambios. Si no existieran las emulaciones podemos definir una variable entera y recordar que el 0=suma, 1=resta, 2=multiplicación y 3=división.

```
int op; //si guardo 0=suma, 1=resta, 2=multiplicacion y 3=division
op=2;
```

Podemos indicar los literales a partir de otro valor y no de cero.

```
enum diassemana {LUNES=1, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO};
```

Con la asignación de un 1 al literal LUNES los demás tomas valores consecutivos a partir de 1, es decir MARTES toma el valor 2, MIERCOLES toma el valor 3 y así sucesivamente.

Podemos asignar valores no consecutivos a cada uno:

```
enum dispositivos {TECLADO=1, MOUSE=10, CAMARA=100};
```

Si consultamos el código fuente del sistema operativo Linux en muchos de sus archivos encontraremos declaraciones de enumeraciones:

```
enum cgroup_filetype {
        CGROUP_FILE_PROCS,
        CGROUP_FILE_TASKS,
};
```

Problema

Crear y cargar un vector de 10 elementos con valores aleatorios entre 1 y 100. Ingresar por teclado un entero y mostrar un mensaje si el número está dentro del vector.

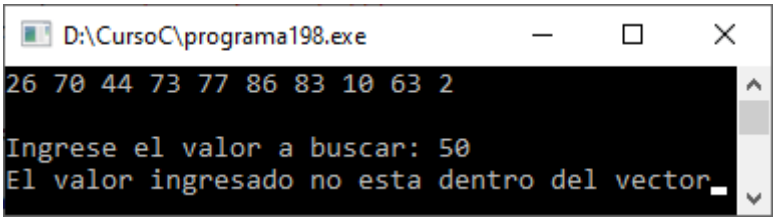
```
#include<stdio.h>
#include<conio.h>
#include<time.h>

#define CANTIDAD 10

enum texiste {NO, SI};

void cargar(int vec[CANTIDAD])
{
    int f;
    srand(time(NULL));
    for(f=0; f<CANTIDAD; f++)
        vec[f]=rand()%100+1;
}

void imprimir(int vec[CANTIDAD])
{
    int f;
    for(f=0; f<CANTIDAD; f++)
        printf("%i ", vec[f]);
    printf("\n\n");
}

void consulta(int vec[CANTIDAD])
{
    int f;
    int valor;
    printf("Ingrese el valor a buscar: ");
    scanf("%i", &valor);
    enum texiste existe=NO;
    for(f=0; f<CANTIDAD; f++)
        if(valor==vec[f])
            existe=SI;
    if(existe==SI)
        printf("El valor ingresado esta dentro el vector");
    else
        printf("El valor ingresado no esta dentro del vector");
}
```

Tipo de dato

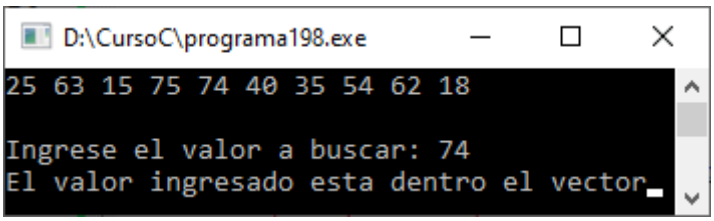
La variable existe que es de tipo enum texiste solo admite valores de SI o NO.

```
41  int main()
42  {
43      int vec[CANTIDAD];
44      cargar(vec);
45      imprimir(vec);
46      consulta(vec);
47      getch();
48      return 0;
49  }
50
```

Si ejecutamos este será el resultado introduciendo en valor que no está en el vector:



Ejecutamos de nuevo con un valor que si existe en el vector:




```
enum texiste {NO, SI};
```

En este ejemplo NO tiene el valor 0 y SI el valor 1 por este motivo podríamos poner:

```
void consulta(int vec[CANTIDAD])
{
    int f;
    int valor;
    printf("Ingrese el valor a buscar: ");
    scanf("%i", &valor);
    enum texiste existe=NO;
    for(f=0; f<CANTIDAD; f++)
        if(valor==vec[f])
            existe=SI;
    if(existe) <----
        printf("El valor ingresado esta dentro el vector");
    else
        printf("El valor ingresado no esta dentro del vector");
}
```

Y funcionará exactamente igual.

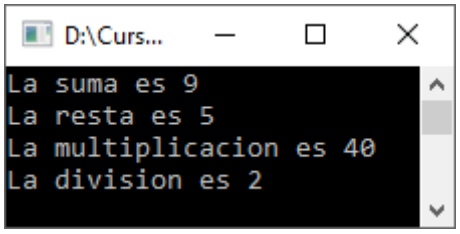
Capítulo 199.- Declaración de enumeraciones – 2

Problema

Confeccionar una función que retorne la suma, resta, multiplicación o división de dos enteros, le pasamos como parámetro los dos valores y un tipo de dato enumerado que indique que operación efectuar.

```
#include<stdio.h>
#include<conio.h>

enum operaciones {SUMA, RESTA, MULTIPLICACION, DIVISION};

void calcular(int num1, int num2, enum operaciones operar)
{
    int resultado;
    if (operar==SUMA)
        printf("La suma es %i\n", num1+num2);
    if (operar==RESTA)
        printf("La resta es %i\n", num1-num2);
    if (operar==MULTIPLICACION)
        printf("La multiplicacion es %i\n", num1*num2);
    if (operar==DIVISION)
        printf("La division es %i\n", num1/num2);
}

int main()
{
    calcular(2, 7, SUMA);
    calcular(10, 5, RESTA);
    calcular(5, 8, MULTIPLICACION);
    calcular(10, 5, DIVISION);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:

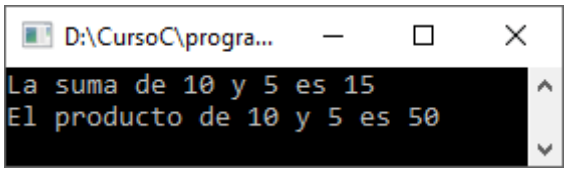


Otra posible solución:

```
#include<stdio.h>
#include<conio.h>

enum operacion {SUMAR, RESTAR, MULTIPLICAR, DIVIDIR};

int operar(int v1, int v2, enum operacion op)
{
    switch(op) {
        case SUMAR: return v1+v2;
```

```c
        case RESTAR: return v1-v2;
        case MULTIPLICAR: return v1*v2;
        case DIVIDIR: return v1/v2;
        default: printf("El programa se detubo por una operacion no valida");
                exit(1);
    }
}

int main()
{
    int valor1=10;
    int valor2=5;
    printf("La suma de %i y %i es %i\n",
           valor1, valor2, operar(valor1, valor2, SUMAR));
    printf("El producto de %i y %i es %i\n",
           valor1, valor2, operar(valor1, valor2, MULTIPLICAR));
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 200.- Declaración de uniones

La unión es una estructura de datos muy particular del lenguaje C. La declaración de una unión es similar a un registro (struct):

```
union dato {
    int x;
    char letra;
    char cadena[5];
};
```

como vemos cambiamos la palabra clave struct por union, hasta ahí la semejanza con los registros.

Cuando definimos una variable de tipo union utilizamos la sintaxis:

```
union dato d;
```

La característica fundamental es que se reserva el mismo espacio de memoria para los atributos x, letra y cadena[5].

En una variable de tipo struct cada campo reserva un espacio distinto de memoria en una unión todas comparten el mismo espacio. Se reserva par el tipo de dato mayor contenida en la union, si tenemos que x es un int y ocupa 4 bytes, letra es de tipo char y ocupa 1 byte y el vector ocupa 5 bytes luego significa que cuando defino una variable de tipo union dato se estará reservando 5 bytes de memoria.

Si consultamos el código fuente del sistema operativo Linux en muchos de sus archivos encontraremos declaraciones de uniones:

```
union {
    struct ahash_request ahreq;
    struct skcipher_request skreq;
}
```

Problema

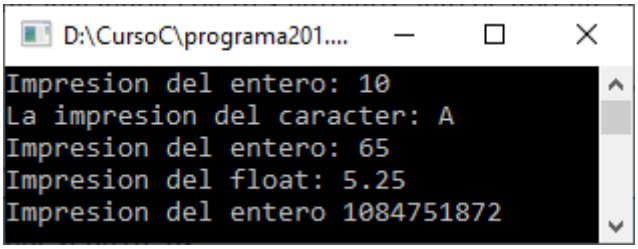
Declarar una unión con tres atributos, uno de tipo int, otro de tipo char y finalmente otgro de tipo float.

Definir una variable y guardar en distintos momentos del programa valores para dichos atributos.

```
1  #include<stdio.h>
2  #include<conio.h>
3
4  union dato {
5      int x;
6      char letra;
7      float z;
8  };
9
```

```c
int main()
{
    union dato d;
    d.x=10;
    printf("Impresion del entero: %i\n", d.x);
    d.letra='A';
    printf("La impresion del caracter: %c\n", d.letra);
    printf("Impresion del entero: %i\n",
           d.x); // El dato se ha modificado
    d.z=5.25;
    printf("Impresion del float: %0.2f\n", d.z);
    printf("Impresion del entero %i\n",
           d.x); // El dato se ha modificado nuevamente
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Declaramos una unión llamada dato:

```c
union dato {
    int x;
    char letra;
    float z;
};
```

En el main definimos una variable:

```c
int main()
{
    union dato d;
```

Modificamos el atributo x cargando en entero 10:

```c
d.x=10;
```

Si imprimimos el atributo x luego se muestra 10:

```c
printf("Impresion del entero:%i\n",d.x);
```

Cargamos ahora el atributo letra y almacenamos la 'A':

```c
d.letra='A';
```

Si imprimimos el atributo letra el resultado es el esperado, es decir la letra 'A':

```
printf("Impresion del caracter:%c\n",d.letra);
```

Pero si ahora accedemos al atributo x que habíamos almacenado el 10 veremos que ya no tiene dicho valor, esto es debido a que los tres atributos comparten el espacio de memoria:

```
printf("Impresion del entero:%i\n",d.x); //El dato se ha modificado
```

Lo mismo sucede cuando modificamos el atributo z y guardamos un valor:

```
    d.z=5.25;
    printf("Impresion del float:%0.2f\n",d.z);
    printf("Impresion del entero:%i\n",d.x); //El dato se ha modificado
nuevamente
```

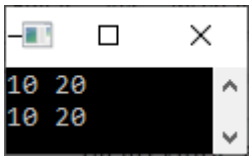
En algunas situaciones definimos uniones para poder acceder a los datos con dos modelos distintos. Veremos un ejemplo donde definimos una unión para acceder a dos enteros que representan a un punto en el plano mediante un registro con dos campos y mediante un vector de dos enteros.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   struct coordenada{
5       int x;
6       int y;
7   };
8
9   union punto{
10      int v[2];
11      struct coordenada co;
12  };
13
14  int main()
15  {
16      union punto p;
17      p.co.x=10;
18      p.co.y=20;
19      printf("%i %i\n", p.co.x, p.co.y); //10 20
20      printf("%i %i\n", p.v[0], p.v[1]); //10 20
21      getch();
22      return 0;
23  }
```

Declaramos una unión con dos elementos, uno de tipo vector y otro de tipo registro.

```
struct coordenada {
    int x;
    int y;
};

union punto {
    int v[2];
    struct coordenada co;
};
```

Comparten el bloque de memoria.



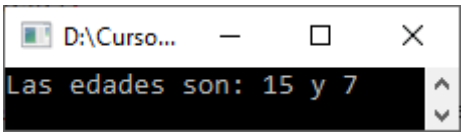
Capítulo 201.- Definición de nuevos nombres para tipos de datos existentes – typedef – 1

En el lenguaje C existen muchos tipos de datos primitivos (int, char, float, etc.) y estructura de datos.

Mediante la palabra clave typedef podemos definir un alias para un tipo de datos existente:

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   typedef int entero;
5
6   int main()
7   {
8       entero edad1, edad2;
9       edad1=15;
10      edad2=7;
11      printf("Las edades son: %i y %i", edad1, edad2);
12      getch();
13      return 0;
14  }
15
```

Si ejecutamos este será el resultado:



La sintaxis para declarar un nuevo nombre para un tipo de dato existente es indicar luego de la palabrea clave typedef el tipo de dato y el nuevo nombre:

```
typedef int entero;
```

Luego de esta declaración podemos definir variables de tipo enero:

```
entero edad1,edad2;
```

Tener en cuenta que en edad1 y edad2 se pueden guardar valores de tipo int.

Decimos que el nombre "entero" es un alias del tipo de dato "int".

La declaración de tipos tiene por objetivo hacer que nuestro programa sea más legible.

typedef con struct

Podemos declarar un alias para un tipo de dato registro.

Si consultamos el código fuente del sistema operativo Linux en muchos de sus archivos encontraremos el uso de typedef:

```
typedef unsigned long long   __u64;
typedef long long            __s64;
typedef unsigned int         __u32;
```

```
typedef struct {
    u64 cpu;
    u64 loop;
    u64 interval;
    u64 err_type_info;
    u64 err_struct_info;
    u64 err_data_buffer[ERR_DATA_BUFFER_SIZE];
} parameters_t;
```

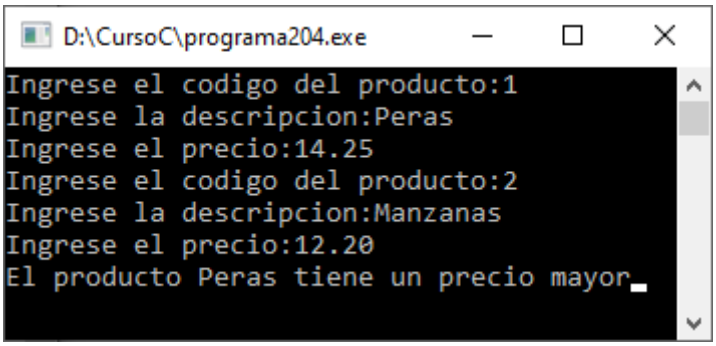
Problema

Declarar un registro que permita almacenar el código, descripción y precio de un producto. Crear un alias para dicho tipo. Definir dos variables, cargarlas e imprimir el nombre del producto que tiene mayor precio.

```
1  #include<stdio.h>
2  #include<conio.h>
3
4  typedef struct {
5      int codigo;
6      char descripcion[41];
7      float precio;
8  } tproducto;
9
10 int main()
11 {
12     tproducto pro1,pro2;
13     printf("Ingrese el codigo del producto:");
14     scanf("%i",&pro1.codigo);
15     fflush(stdin);
16     printf("Ingrese la descripcion:");
17     gets(pro1.descripcion);
18     printf("Ingrese el precio:");
19     scanf("%f",&pro1.precio);
20     printf("Ingrese el codigo del producto:");
21     scanf("%i",&pro2.codigo);
22     fflush(stdin);
23     printf("Ingrese la descripcion:");
24     gets(pro2.descripcion);
25     printf("Ingrese el precio:");
26     scanf("%f",&pro2.precio);
27     if (pro1.precio>pro2.precio)
28     {
29         printf("El producto %s tiene un precio mayor",pro1.descripcion);
30     }
31     else
32     {
33         if (pro2.precio>pro1.precio)
34         {
35             printf("El producto %s tiene un precio mayor",pro2.descripcion);
36         }
37         else
38         {
39             printf("Tienen igual precio");
40         }
41     }
42     getch();
43     return 0;
44 }
```

Creamos el alias al hacer el registro.

Si ejecutamos este será el resultado:



De otra forma:

```c
#include<stdio.h>
#include<conio.h>

struct producto{
    int codigo;
    char descripcion[41];
    float precio;
};

typedef struct producto tproducto;

int main()
{
    tproducto pro1,pro2;
    printf("Ingrese el codigo del producto:");
    scanf("%i",&pro1.codigo);
    fflush(stdin);
    printf("Ingrese la descripcion:");
    gets(pro1.descripcion);
    printf("Ingrese el precio:");
    scanf("%f",&pro1.precio);
    printf("Ingrese el codigo del producto:");
    scanf("%i",&pro2.codigo);
    fflush(stdin);
    printf("Ingrese la descripcion:");
    gets(pro2.descripcion);
    printf("Ingrese el precio:");
    scanf("%f",&pro2.precio);
    if (pro1.precio>pro2.precio)
    {
        printf("El producto %s tiene un precio mayor",pro1.descripcion);
    }
    else
    {
        if (pro2.precio>pro1.precio)
        {
            printf("El producto %s tiene un precio mayor",pro2.descripcion);
        }
        else
        {
            printf("Tienen igual precio");
        }
    }
    getch();
    return 0;
}
```

Primero creamos el registro y a continuación creamos e alias.

El resultado será el mismo.

Capítulo 202.- Definición de nuevos nombres para tipos de datos existentes – typedef – 2

typedef con array

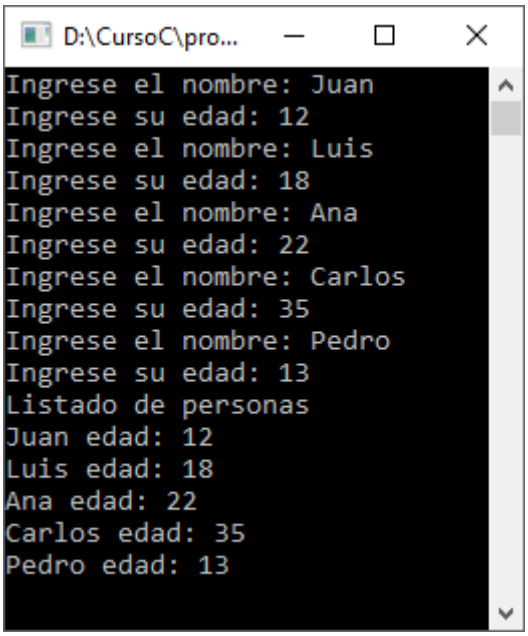
Podemos declarar un alias par un tipo de array de registros.

Problema

Definir un alias de un vector de 5 elementos con componentes de tipo registro que almacenan el nombre de la persona y su edad.

```
#include<stdio.h>
#include<conio.h>

#define TAM 5

struct persona{
    char nombre[40];
    int edad;
};

typedef struct persona tpersona[TAM];

void cargar(tpersona per)
{
    for (int x=0; x<TAM; x++)
    {
        fflush(stdin);
        printf("Ingrese el nombre: ");
        gets(per[x].nombre);
        printf("Ingrese su edad: ");
        scanf("%i",&per[x].edad);
    }
}

void imprimir(tpersona per)
{
     printf("Listado de personas\n");
     for (int x=0; x<TAM; x++)
     {
         printf("%s edad: %i\n",per[x].nombre, per[x].edad);
     }
     printf("\n");
}

int main()
{
    tpersona personas;
    cargar(personas);
    imprimir(personas);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 203.- Definición de nuevos nombres para tipos de datos existentes – typedef – 3

Problema

Confeccionar un programa que administre una lista tipo pila (se debe poder insertar, extraer e imprimir los datos de la pila).

La estructura del nodo es la siguiente:

```
struct nodo {
    int info;
    struct nodo *sig;
};
```

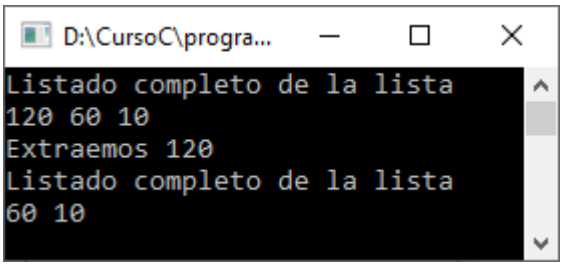
Luego se crear un alias tipo de dato puntero a nodo:

```
typedef struct nodo * tnodo;
```

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

struct nodo {
    int info;
    struct nodo *sig;
};

typedef struct nodo * tnodo;

tnodo raiz=NULL;

void insertar(int x)
{
    tnodo nuevo;
    nuevo=malloc(sizeof(struct nodo));
    nuevo->info=x;
    if (raiz==NULL)
    {
        raiz=nuevo;
        nuevo->sig=NULL;
    }
    else
    {
        nuevo->sig=raiz;
        raiz=nuevo;
    }
}

```

```
31  void imprimir()
32  {
33      tnodo reco=raiz;
34      printf("Listado completo de la lista\n");
35      while (reco!=NULL)
36      {
37          printf("%i ", reco->info);
38          reco=reco->sig;
39      }
40  }
41
42  int extraer()
43  {
44      if (raiz!=NULL)
45      {
46          int informacion=raiz->info;
47          tnodo bor=raiz;
48          raiz=raiz->sig;
49          free(bor);
50          return informacion;
51      }
52       else
53          return -1;
54  }
55
56  void liberar()
57  {
58      tnodo reco=raiz;
59      tnodo bor;
60      while(reco!=NULL)
61      {
62          bor=reco;
63          reco=reco->sig;
64          free(bor);
65      }
66  }
67
68  int main()
69  {
70      insertar(10);
71      insertar(60);
72      insertar(120);
73      imprimir();
74      printf("\nExtraemos %i\n", extraer());
75      imprimir();
76      liberar();
77      getch();
78      return 0;
79  }
```

Si ejecutamos este será el resultado:



Capítulo 204.- Variables locales static – 1

Hemos visto que cuando definimos una variable local su valor se pierde inmediatamente cuando finaliza la función.

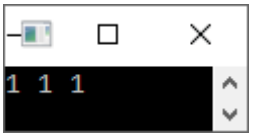
En el lenguaje C podemos definir variables locales que no pierdan su valor entre llamadas mediante el modificador static.

Veamos con dos programas la diferencia entre definir una variable local y una variable local static.

Programa

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   void imprimir()
5   {
6       int x=0;
7       x++;
8       printf("%i ", x);
9   }
10
11  int main()
12  {
13      imprimir();
14      imprimir();
15      imprimir();
16      getch();
17      return 0;
18  }
```

Como sabemos el resultado de ejecutar el programa es:



Cuando se define una variable local como estatic se reserva espacio para la misma antes de ser llamada a la función y se inicializa con el valor que le asignemos. Esto significa que la variable local x existe previo a llamar a la función.

El contenido de la variable local static solo se le puede modificar o consultar dentro de dicha función.

Es muy importante tener en cuenta que en cada llamada a la función imprimir no se está definiendo una variable local x e inicializándola en cero, podemos decir que la línea:

```
static int x=0;
```

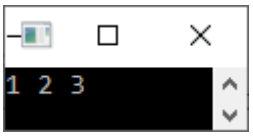
Se ejecuta antes de ser llamada la función. Cuando llamamos a la función ejecutan las líneas:

```
x++;
printf("%i ",x);
```

Es decir que en cada llamada a la función imprimir se incrementa en uno la variable static x y e muestra su contenido.

```c
#include<stdio.h>
#include<conio.h>

void imprimir()
{
    static int x;
    x++;
    printf("%i ", x);
}

int main()
{
    imprimir();
    imprimir();
    imprimir();
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Si bien este problema se puede resolver utilizando una variable global es más conveniente resolverlo con una variable local static:

Ejemplo con variable global:

```c
#include<stdio.h>
#include<conio.h>

int x=0;

void imprimir()
{
    x++;
    printf("%i ", x);
}

int main()
{
    imprimir();
    imprimir();
    imprimir();
    getch();
    return 0;
}
```

Si ejecutamos el resultado será el mismo.

Problema

Generar un número aleatorio entre 1 y 100. Pedir que el operador lo adivine mostrando un mensaje si gano o si el número a adivinar es menor o mayor al propuesto.

Permitir solo tres intentos.

```c
#include<stdio.h>
#include<conio.h>
#include<time.h>

int verificarFin()
{
    static int intentos=3;
    intentos--;
    if(intentos==0)
    {
        printf("Perdio, solo tenia 3 intentos");
        return 1;
    }
    return 0;
}

int jugar()
{
    srand(time(NULL));
    int numero = rand() % 100 + 1;
    int valor;
    do{
        printf("Adivina el numero que pense entre 1 y 100: ");
        scanf("%i", &valor);
        if(valor==numero)
        {
            printf("Ganaste");
            break;
        }
        else
        {
            if (valor<numero)
                printf("El numero que pense es mayor\n");
            else
                printf("El numero que pense es menor\n");
        }
    } while(verificarFin()==0);
}

int main()
{
    jugar();
    getch();
    return 0;
}
```

```
D:\CursoC\programa208.exe
Adivina el numero que pense entre 1 y 100: 20
El numero que pense es mayor
Adivina el numero que pense entre 1 y 100: 80
El numero que pense es mayor
Adivina el numero que pense entre 1 y 100: 90
Ganaste
```

Si ejecutamos este será el resultado:

Capítulo 205.- Variables locales static – 2

Problema propuesto

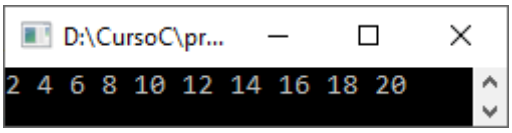
Confeccionar una función que retorne el próximo múltiplo de 2. La primera vez debe mostrar un dos, la segunda ver un 4 y así sucesivamente.

```
int proximoMultiplo2()
{
   ...
}
```

Llamar a la función 10 veces desde la main.

```
1  #include<stdio.h>
2  #include<conio.h>
3
4
5  int proximoMultiplo2()
6  {
7      static int multiplo;
8      multiplo+=2;
9      return multiplo;
10 }
11
12 int main()
13 {
14     for (int x=0; x<10; x++)
15         printf("%i ", proximoMultiplo2());
16     getch();
17     return 0;
18 }
19
```

Si ejecutamos este será el resultado:



Capítulo 206.- Aplicaciones en C con más de un archivo fuente (proyectos con múltiples archivos)

En programas medianos o grandes es inviable disponer todo el código fuente en un único archivo. Yendo a un ejemplo extremo el sistema operativo Linux está compuesto por más de 50000 archivos.

En este concepto veremos cómo implementar una aplicación que tenga dos archivos *.c y que luego se compilen y generen un único archivo ejecutable.

Utilizando Code::Blocks nos facilita esta actividad mediante la creación de un proyecto.

Problema

Se tiene la siguiente declaración:

```
typedef struct {
    int codigo;
    char descripcion[41];
    float precio;
} tproducto;
```

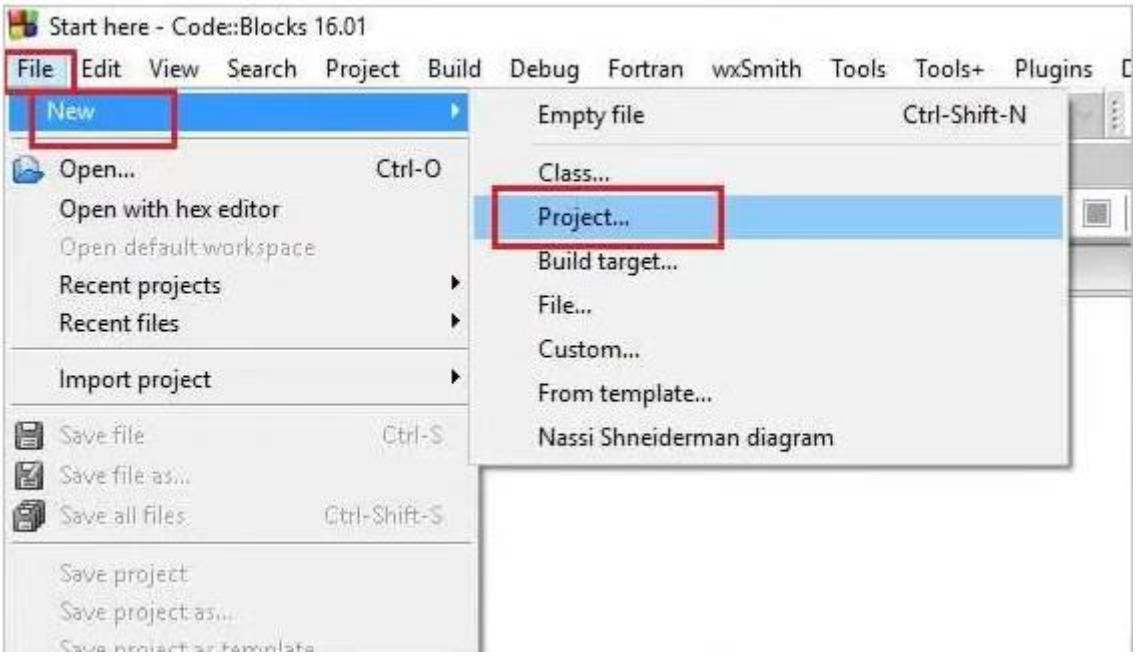
Definir un vector de 4 elementos de tipo tproducto.

Implementar una aplicación que muestre un menú de opciones que permita:

1. Carga del vector.
2. Impresión del vector.
3. Finalizar

Como dijimos resolveremos este problema disponiendo los algoritmos de carga e impresión del vector en un archivo y en otro archivo el menú de opciones.

Para crear un proyecto en Code::Blocks elegimos la opción:



Aparece un diálogo donde elegimos la opción "Empety Project" (proyecto vacío).

Aparecerá una serie de diálogos para configurar el proyecto.

Uno de los pasos importantes es indicar el nombre del proyecto y la carpeta donde se debe almacenar el proyecto.

En el diálogo siguiente dejamos por defecto los datos propuestos.

Ahora tenemos que crear los dos archivos con extensión .c que tendrán los algoritmos propiamente dicho.

Creamos dos archivos llmados "main.c" que será el principal que contendrá la función main y otro archivo llamado "productos.c" que tendrá los algoritmos para poder cargar e imprimir el vector.

Estando seleccionado el proyecto001 elegimos desde el menú de opciones File -> Empty file.

Aparecerá un diálogo para confirmar si el archivo lo queremos agregar al proyecto seleccionado:

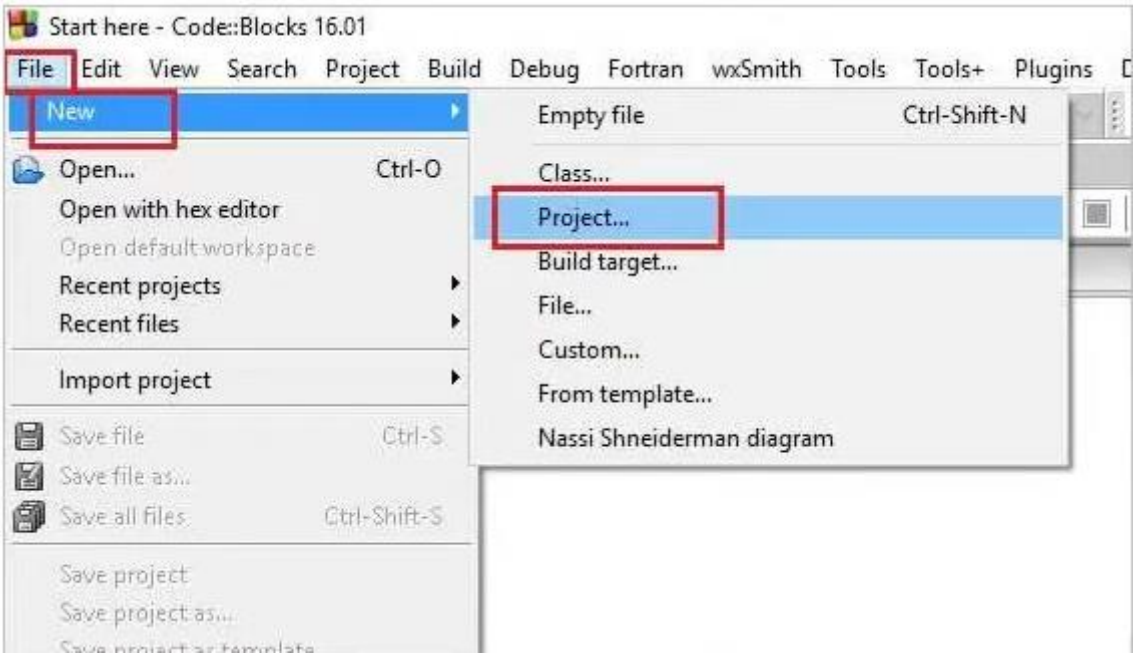
Indicamos seguidamente el nombre del archivo a crear, en nuestro caso lo llamamos "main.c":

Después de esto ya tenemos agregado a nuestro proyecto el archivo "main.c".

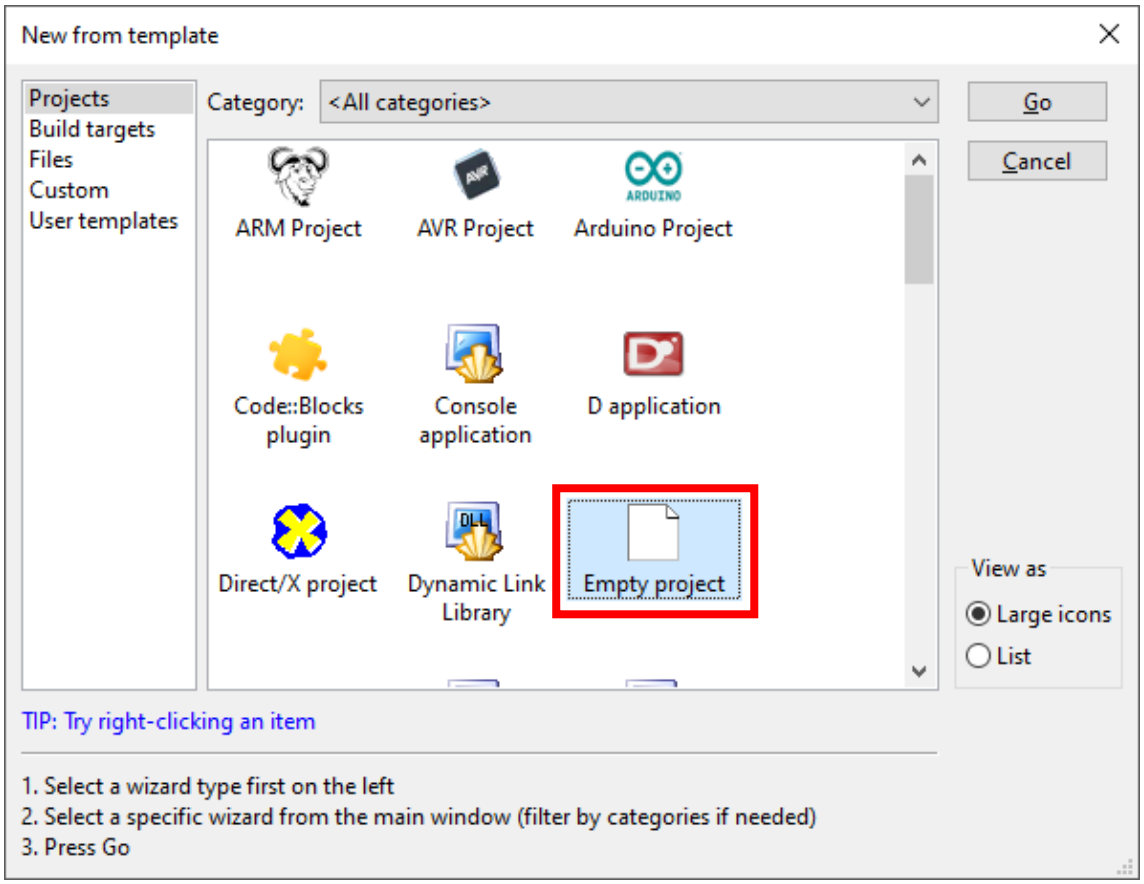
Efectuamos los mismos pasos para agregar el segundo archivo "productos.c":

Ahora crearemos un tercer archivo llamado "productos.h", tener en cuenta que la extensión es h (heder) los paso son los mismos hechos anteriormente (luego veremos cual es el objetivo de este tercer archivo).

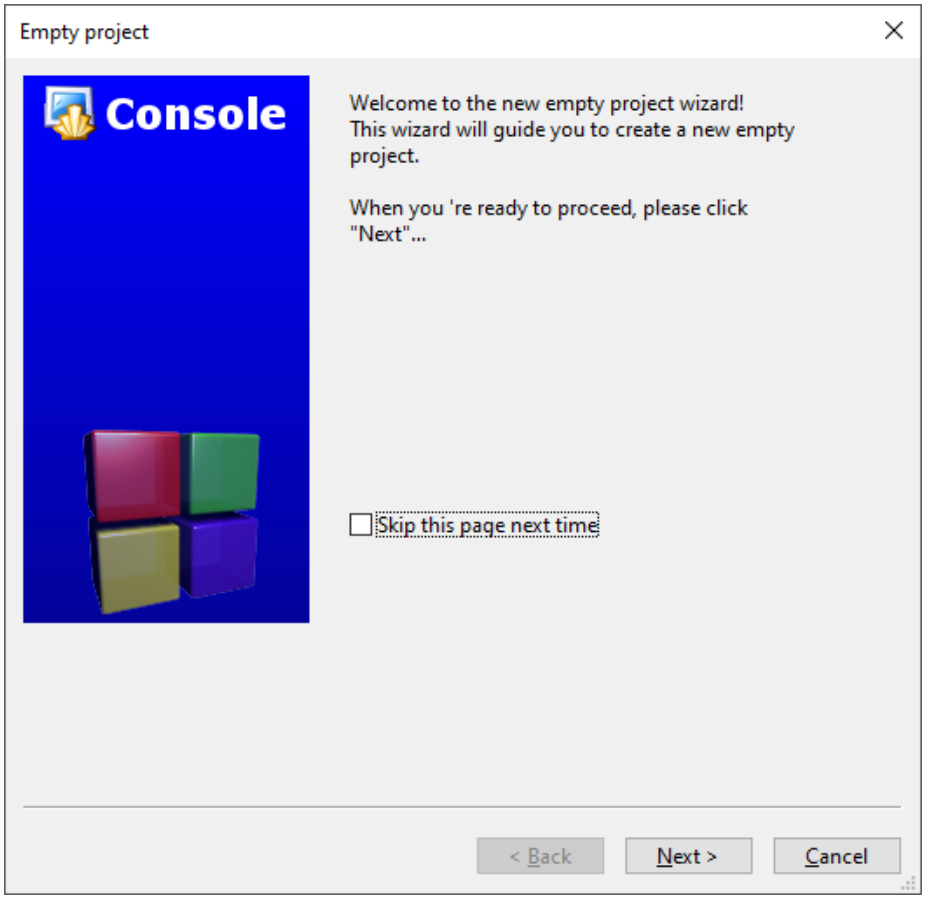
Vamos a realizar los pasos:



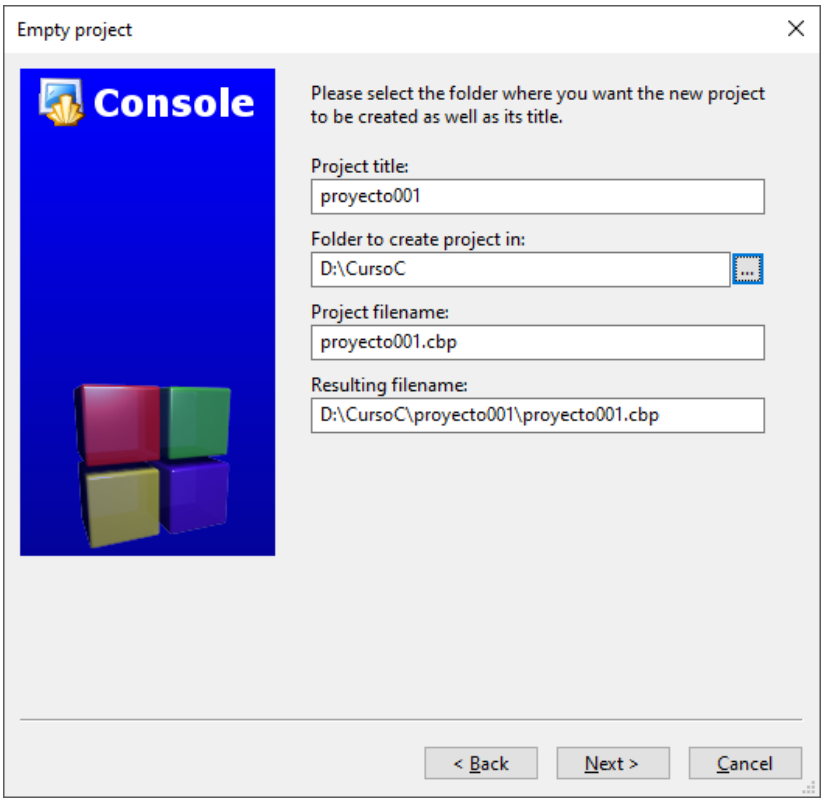
Del menú seleccionamos File -> New -> Projecct…



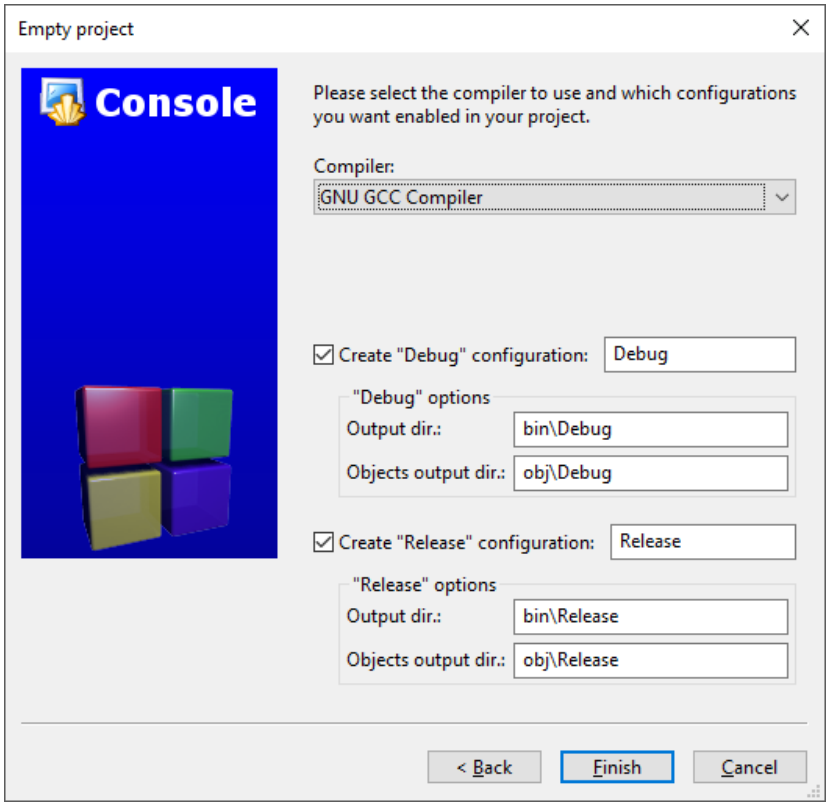
Seleccionaremos Empty Project, seguido del botón Go.



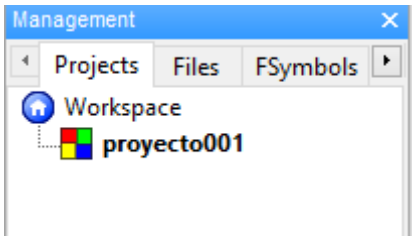
Le daremos al botón Next.



Damos nombre al proyecto 'proyecto001', le decimos en que carpeta lo queremos ubicar, y nos ha agregado Project filename y Resulting filename, seguido del botón next.

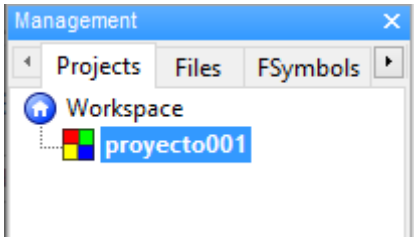


Dejamos los valores por defecto seguido del botón Finish.

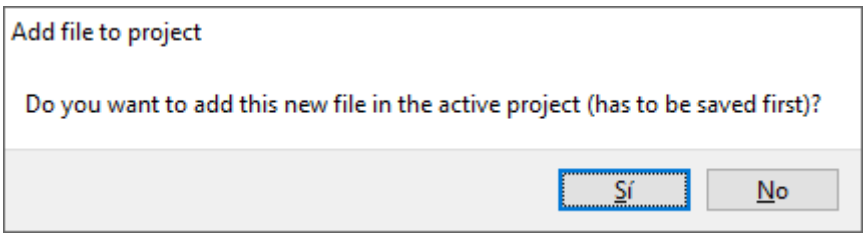


Ya hemos creado el proyecto001.

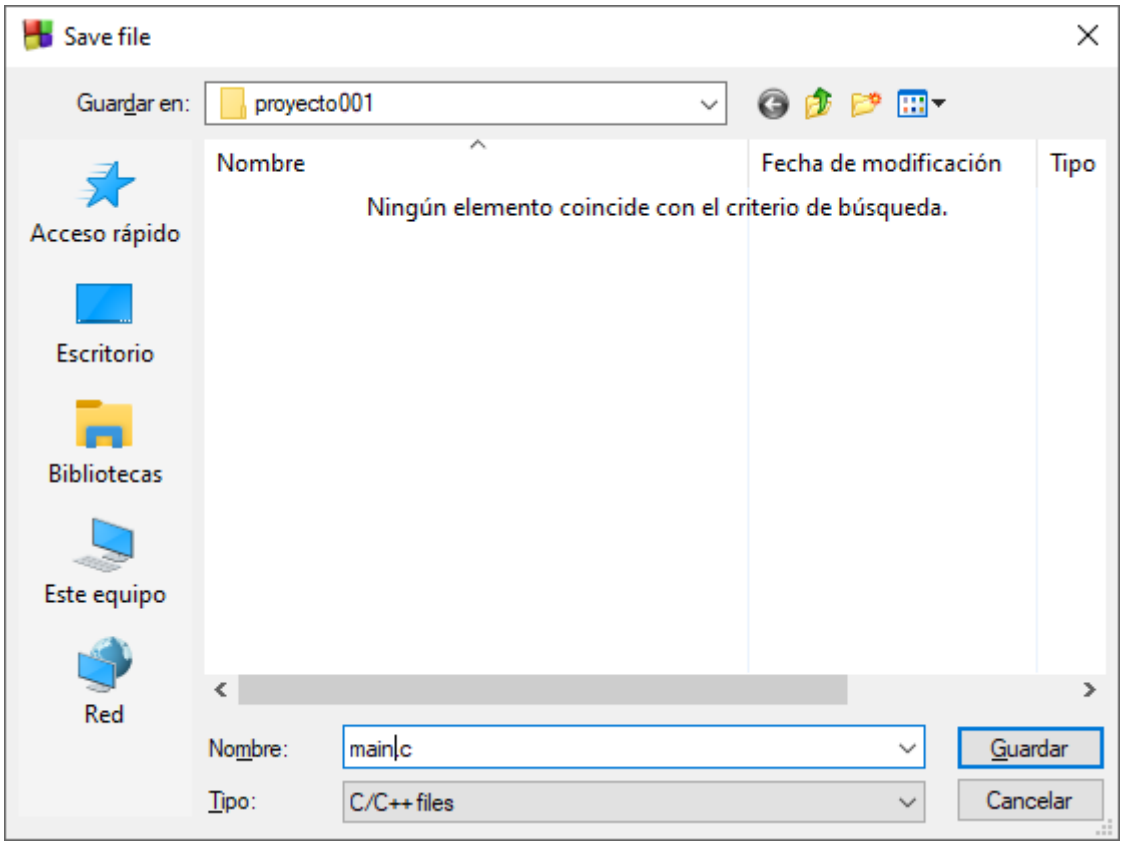
Ahora vamos a insertar los dos archivos con extensión c:



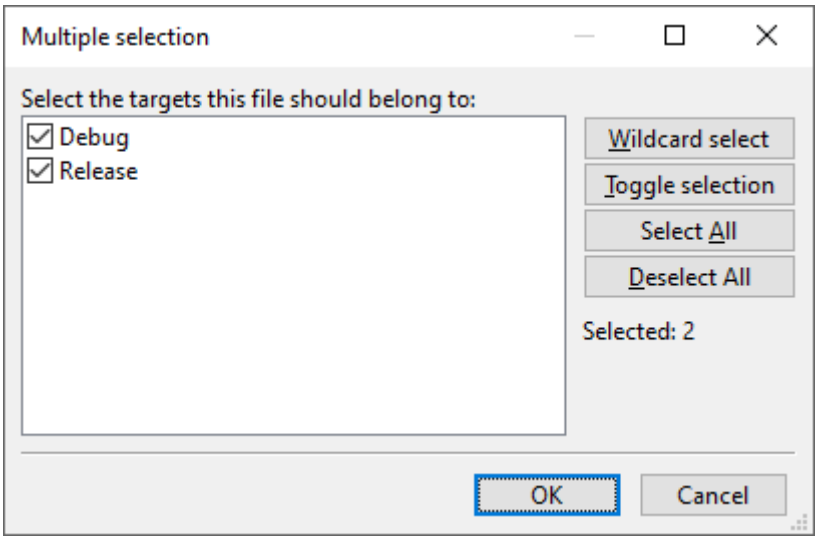
Seleccionamos el proyecto y del menú File -> New -> Empty file.



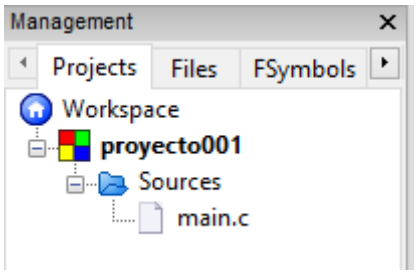
Nos pregunta si queremos añadir el archivo al proyecto seleccionado, contestaremos Sí.



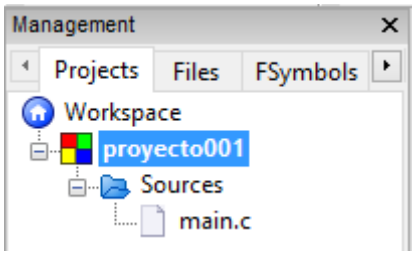
El primer archivo ser llamara main.c, seleccionaremos el botón Gurdar.



Dejamos los valores por defecto seguido del botón OK.

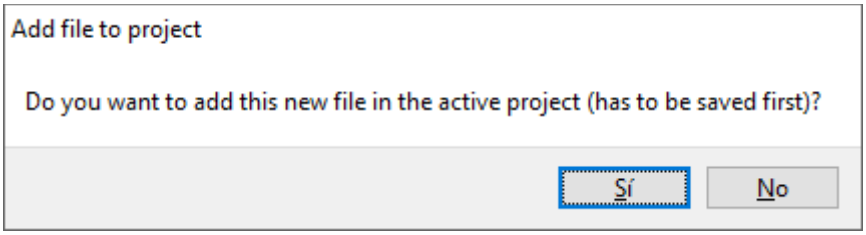


Nuevamente seleccionamos el proyecto para agregar el segundo archivo.

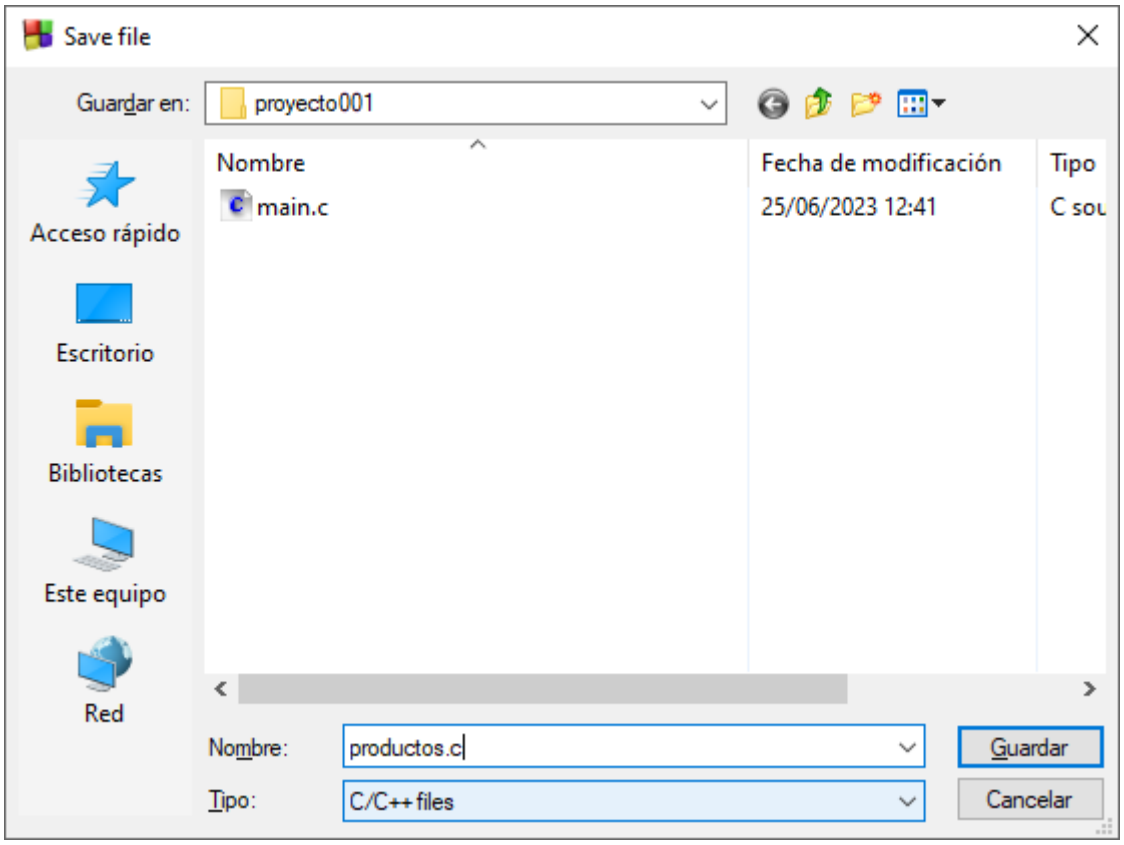


Y repetimos los pasos anteriores para agregar el segundo archivo.

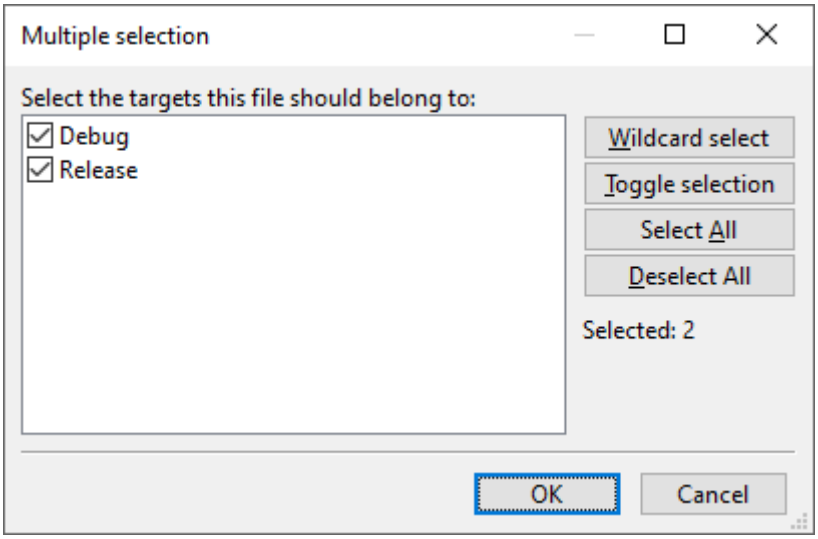
Del menú File seleccionaremos New y de este Empty file.



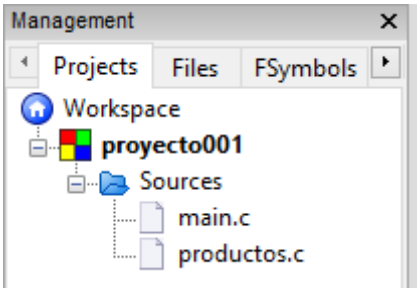
Seleccionaremos el botón Si.



Le pondremos el nombre de productos.c seguido del botón Guardar.

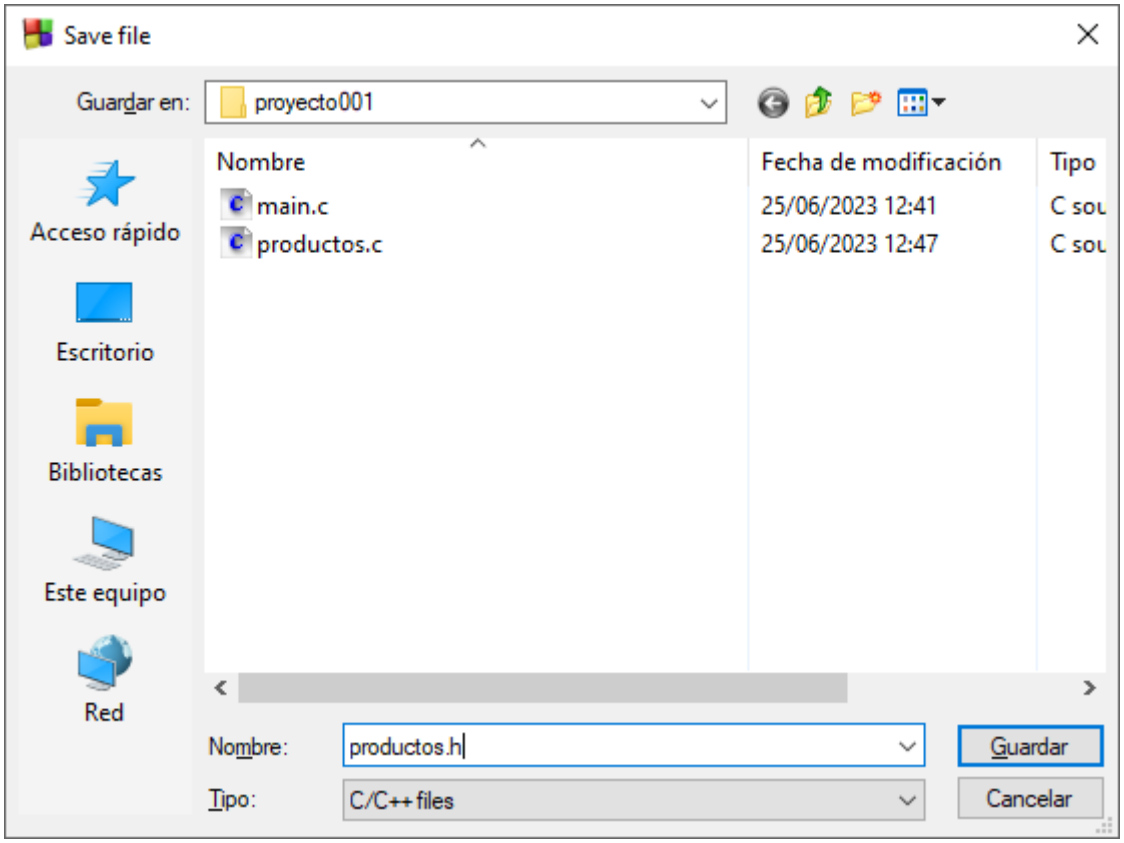


Dejamos los valores por defecto seguido del botón OK.

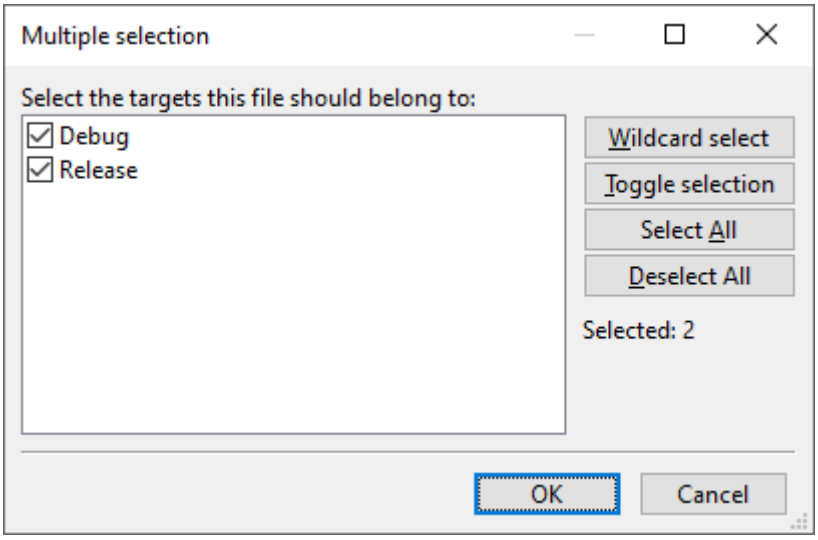


Ahora vamos a crear un tercer archivo llamado 'productos.h', que nos permitirá la comunicación entre los dos archivos.

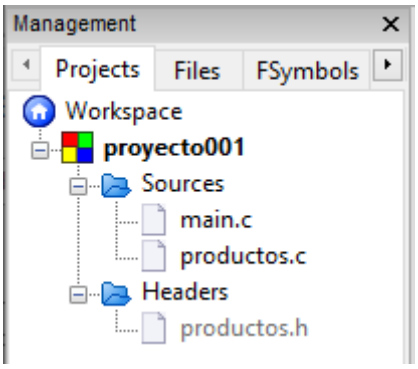
Repetiremos los pasos anteriores:



seleccionamos el botón Guardar.



Le damos al botón OK.



Este será el resultado final.

Primero codificaremos el archivo productos.h.

En este tipo de archivo declaramos tipos de datos, macros, emulaciones, uniones, prototipos de funciones, etc. que sean de uso común a varios archivos de nuestro programa.

Archivo: productos.h

```
1   typedef struct {
2       int codigo;
3       char descripcion[41];
4       float precio
5   } tproducto;
6
7   #define CANT 4
8
9   void cargar(tproducto producto[CANT]);
10  void imprimir(tproducto producto[CANT]);
```

En este archivo declaramos el tipo de dato tproduto y la macro CANT.

Además declaramos dos prototipos de las funciones que se implementarán en el archivo productos.c.

Los prototipos son las cabeceras de las funciones.

Archivo: main.c

```
1   #include<stdio.h>
2   #include"productos.h"
3
4   int main()
5   {
6       tproducto productos[CANT];
7       int opcion;
8       do{
9           printf("1.- Carga de productos.\n");
10          printf("2.- Listado de productos.\n");
11          printf("3.- Finalizar programa..\n\n");
12          printf("Elija su opcion: ");
13          scanf("%i", &opcion);
14          switch (opcion) {
15              case 1: cargar(productos);
16                      break;
```

```
17              case 2: imprimir(productos);
18                      break;
19          }
20      }while(opcion!=3);
21      return 0;
22  }
```

En la función main definimos un vector de tipo producto:

```
int main()
{
    tproducto productos[CANT];
```

¿Cómo podemos definir una variable de un tipo de dato que no está definido en el archivo?

La respuesta está en que hemos incluido el archivo que contiene dicha declaración del tipo de dato:

```
#include"productos.h"
```

La sintaxis para incluir archivos.h creados por nosotros y contenidos en el proyecto es encerrarlos entre comillas dobles y no con los caracteres <>.

El tercer archivo "productos.c" tiene por objetivo cargar e imprimir el vector.

Archivo: productos.c

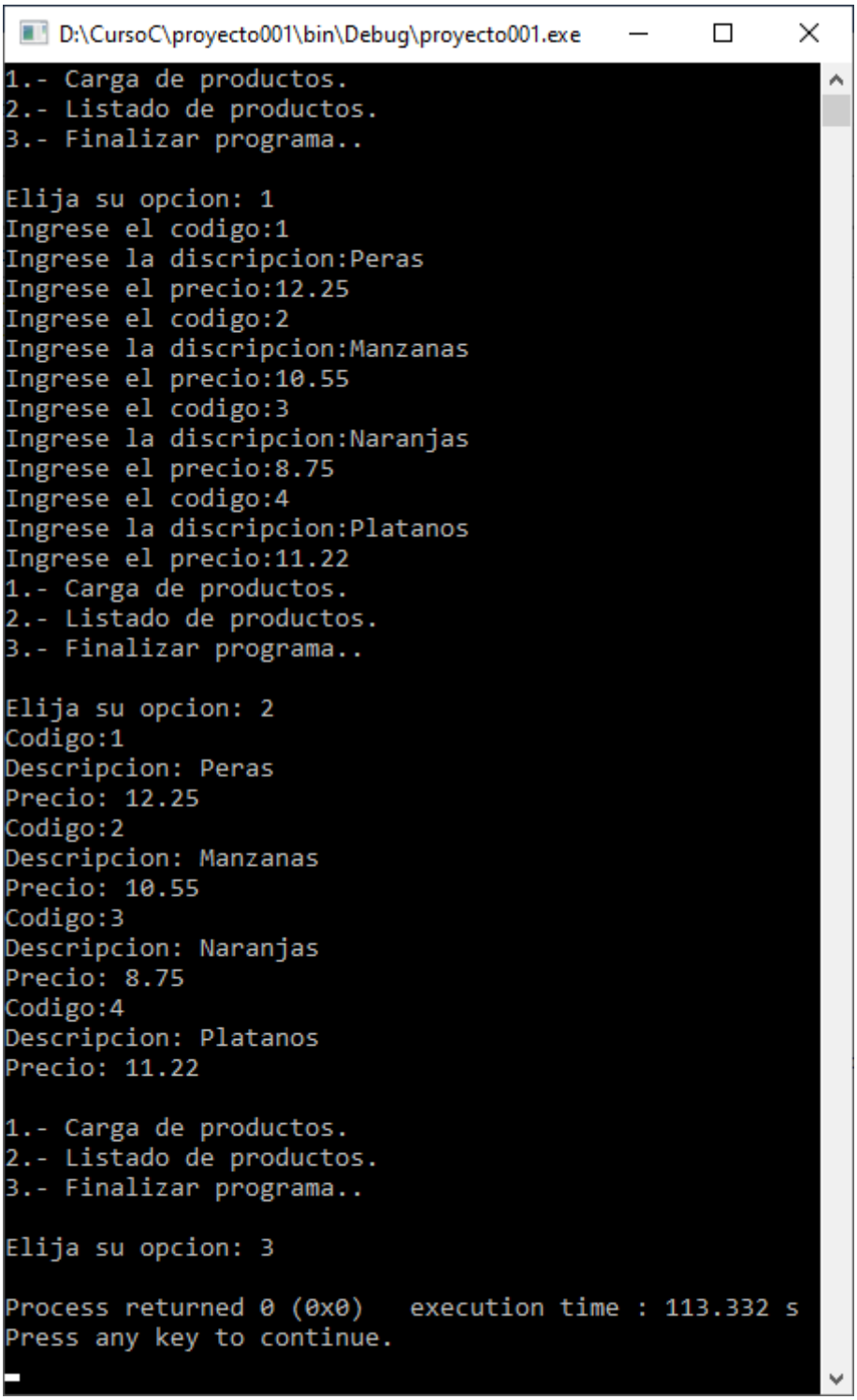
```
1   #include<stdio.h>
2   #include"productos.h"
3
4   void cargar(tproducto productos[CANT])
5   {
6       int f;
7       for(f=0; f<CANT; f++)
8       {
9           printf("Ingrese el codigo:");
10          scanf("%i", &productos[f].codigo);
11          fflush(stdin);
12          printf("Ingrese la discripcion:");
13          gets(productos[f].descripcion);
14          printf("Ingrese el precio:");
15          scanf("%f", &productos[f].precio);
16      }
17  }
18
19  void imprimir(tproducto productos[CANT])
20  {
21      int f;
22      for(f=0; f<CANT; f++)
23      {
24          printf("Codigo:%i\n", productos[f].codigo);
25          printf("Descripcion: %s\n", productos[f].descripcion);
26          printf("Precio: %0.2f\n", productos[f].precio);
27      }
28      printf("\n");
29  }
```

En este archivo también trabajamos con el tipo de dato "tproducto" y la macro CANT por lo que primero hacemos la inclusión del archivo "productos.h":

También incluimos el archivo "stdio.h" ya que es ese archivo se declara el prototipo de la función printf y scanf:

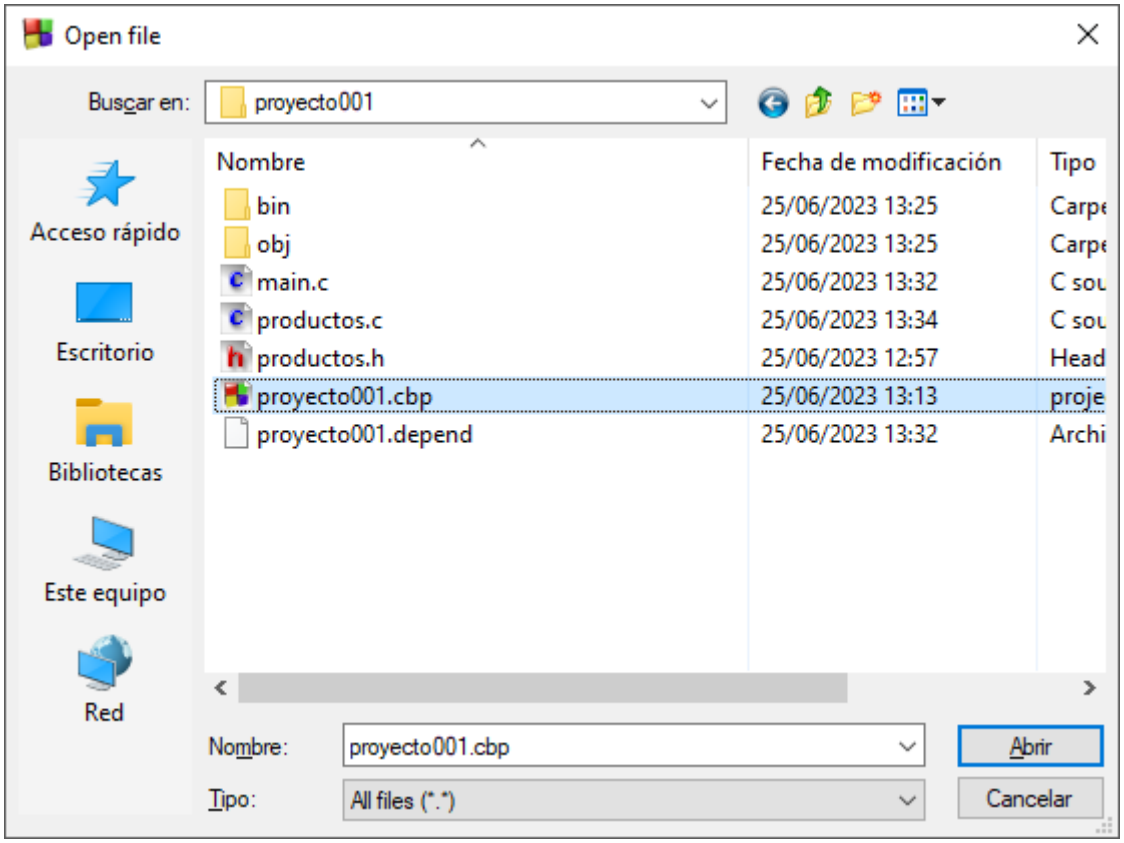
```
#include<stdio.h>
```

Vamos a ejecutar: (Generará un archivo compilado llamado proyecto001.exe.

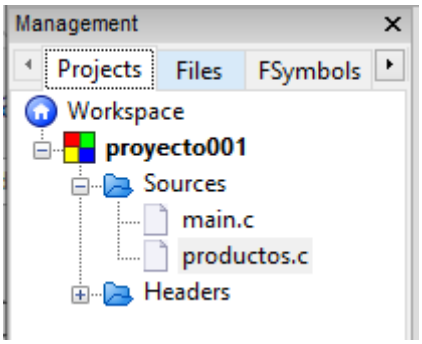


Si en un futuro queremos recuperar este proyecto realizaremos los siguientes pasos:

Del menú File seleccionaremos Open.



Seleccionaremos el archivo proyecto001.cbp seguido del botón abrir.



Ya lo podemos ejecutar de nuevo a modificarlo.

Capítulo 207.- Definición de funciones y variables static

Cuando tenemos proyectos grandes cada uno de los archivos *.c tiene un objetivo definido y lo llevan a cabo implementando funciones.

Las funciones que implementan un archivo *.c son accedidas desde otros archivos.

Cuando queremos implementar funciones que solo pueden ser llamadas desde funciones del mismo archivo debemos anteceder el modificador static previo al nombre de la función.

Una función static aplicada a una variable global definida en el archivo también solo puede ser accedida por las funciones de ese archivo.

Problema

Se tiene la siguiente declaración:

```
typedef struct {
    int codigo;
    char descripcion[41];
    float precio;
} tproducto;
```

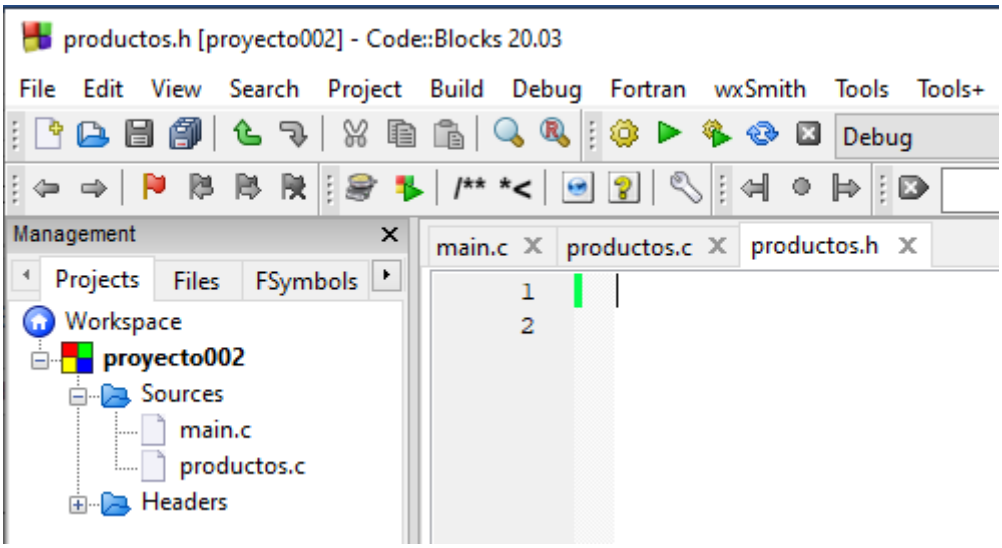
Definir un vector de 4 elementos de tipo tproducto.

Implementar una aplicación que muestre un menú de opciones que permita:

1. Carga del vector.
2. Impresión del vector.
3. Finalizar

Mostrar un título y una línea de asteriscos cuando se carga e imprima el vector.

Crearemos un proyecto en Code::Blocks dando los pasos similares al capítulo anterior ky lo llamaremos proyecto002.



Los contenidos de los tres archivos del proyecto son:

Archivo: producto.h

```c
typedef struct {
    int codigo;
    char descripcion[41];
    float precio;
} tproducto;

#define CANT 4

void cargar(tproducto productos[CANT]);
void imprimir(tproducto productos[CANT]);
```

Archivo: productos.c

```c
#include<stdio.h>
#include"productos.h"

static void mostrarTitulo(char *tit)
{
    printf("%s\n", tit);
}

static void mostrarAsteriscos()
{
    printf("************************************************\n\n");
}

void cargar(tproducto productos[CANT])
{
    int f;
    mostrarTitulo("Carga de datos de productos");
    for(f=0; f<CANT; f++)
    {
        printf("Ingrese el codigo: ");
        scanf("%i", &productos[f].codigo);
        fflush(stdin);
        printf("Ingrese al descripcion: ");
        gets(productos[f].descripcion);
        printf("Ingrese el precio: ");
        scanf("%f", &productos[f].precio);
    }
    mostrarAsteriscos();
}

void imprimir(tproducto productos[CANT])
{
    int f;
    mostrarTitulo("Listado de productos");
    for(f=0; f<CANT; f++)
    {
        printf("Codigo: %i\n", productos[f].codigo);
        printf("Descripcion: %s\n", productos[f].descripcion);
        printf("Precio: %0.2f\n", productos[f].precio);
    }
    mostrarAsteriscos();
}
```

En este archivo tenemos los cambios con respecto al problema del capítulo anterior.

Las funciones mostrarTitulo y mostrarAsteriscos queremos que solo puedan ser llamadas desde otras funciones del mismo archivo productos.c, para este le agregamos el modificador static:

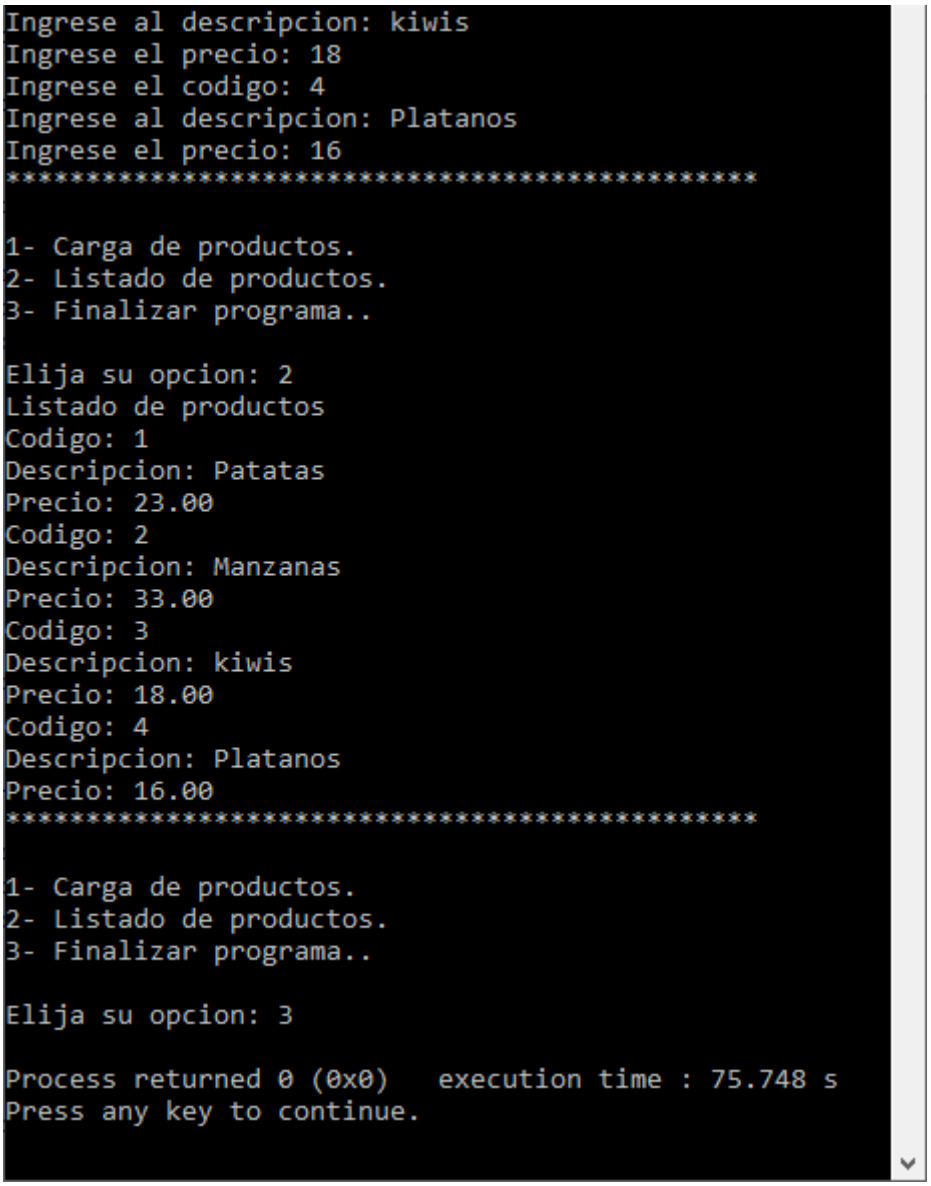
```
static void mostrarTitulo(char *tit)
{
    printf("%s\n",tit);
}

static void mostrarAsteriscos()
{
    printf("**********************************************\n\n");
}
```

Deben estar declaradas en forma previa a cuando son llamadas.

Archivo: main.c

```
#include<stdio.h>
#include"productos.h"

int main()
{
    tproducto productos[CANT];
    int opcion;
    do{
        printf("1- Carga de productos.\n");
        printf("2- Listado de productos.\n");
        printf("3- Finalizar programa..\n\n");
        printf("Elija su opcion: ");
        scanf("%i", &opcion);
        switch (opcion){
            case 1:cargar(productos);
                   break;
            case 2:imprimir(productos);
                   break;
        }
    }while(opcion!=3);
    return 0;
}
```

```
D:\CursoC\proyecto002\bin\Debug\proyecto002.exe
1- Carga de productos.
2- Listado de productos.
3- Finalizar programa..

Elija su opcion: 1
Carga de datos de productos
Ingrese el codigo: 1
Ingrese al descripcion: Patatas
Ingrese el precio: 23
Ingrese el codigo: 2
Ingrese al descripcion: Manzanas
Ingrese el precio: 33
Ingrese el codigo: 3
```



En el archivo main.c no podemos llamar a las funciones mostrarTitulo y mostrarAsteriscos.

Si intentamos llamarlas:

```
int main()
{
    tproducto productos[CANT];
    int opcion;
    mostrarAsteriscos(); //error
    do {
        printf("1-Carga de productos.\n");
```

Aparecerá un error sintáctico en pantalla.

Si consultamos el código fuente del sistema operativo Linux en muchos de sus archivos encontraremos definiciones de funciones de tipo static:

```c
static int do_fsync(unsigned int fd, int datasync)
{
    struct fd f = fdget(fd);
    int ret = -EBADF;

    if (f.file) {
        ret = vfs_fsync(f.file, datasync);
        fdput(f);
    }
    return ret;
}
```

Capítulo 208.- Variables globales en el modificador extern

Cuando tenemos una aplicación constituida por varios archivos *.c y queremos a una variable global de un archivo a otro archivo debemos declarar la variable global con el modificador estern.

Cuando una variable global se declara como extern, el compilador no crea un espacio para ella en memoria, sino que, tan solo tiene en cuenta que dicha variable ya ha sido declarada en otro archivo del programa.

Problema

Confeccionar un programa que administre una lista de tipo pila (se debe poder, extraer e imprimir los datos de la pila).
La estructura del nodo es el siguiente:

```
struct nodo {
    int info;
    struct nodo *sig;
};
```

Luego se crea un alias para tipo de dato puntero a nodo:

```
typedef struct nodo * tnodo;
```

Crea un proyecto llamado: proyecto003 e implementa todas las funciones para administrar la pila en un archivo separado. En el archivo principal definir una variable global que apunte al primer nodo de la lista. Acceder a dicho puntero desde los dos archivos.

Archivo: pila.h

```
1  struct nodo {
2      int info;
3      struct nodo *sig;
4  };
5
6  typedef struct nodo * tnodo;
7
8  void insertar(int x);
9  void imprimir();
10 int extraer();
11 void liberar();
```

Archivo: main.c

```
1  #include<stdio.h>
2  #include<conio.h>
3  #include"pila.h"
4
5  tnodo raiz=NULL;
6
7  int main()
8  {
9      insertar(10);
10     insertar(40);
11     insertar(10);
12     imprimir();
```

```
13      printf("Extraemos de la pila: %i\n", extraer());
14      imprimir();
15      liberar();
16      getch();
17      return 0;
18  }
```

En este archivo definimos una variable global llamada raiz de tipo tnodo y se la inicializa con el valor NULL:

```
tnodo raiz=NULL;
```

Archivo: pila.c

```
1   #include<stdio.h>
2   #include<stdlib.h>
3   #include"pila.h"
4
5   extern tnodo raiz;
6
7   void insertar(int x)
8   {
9       tnodo nuevo;
10      nuevo = malloc(sizeof(struct nodo));
11      nuevo->info=x;
12      if (raiz==NULL)
13      {
14          raiz=nuevo;
15          nuevo->sig=NULL;
16      }
17      else
18      {
19          nuevo->sig=raiz;
20          raiz=nuevo;
21      }
22  }
23
24  void imprimir()
25  {
26      tnodo reco=raiz;
27      printf("Lista completa.\n");
28      while (reco!=NULL)
29      {
30          printf("%i ", reco->info);
31          reco=reco->sig;
32      }
33      printf("\n");
34  }
35
```

```
36  int extraer()
37  {
38      if (raiz!=NULL)
39      {
40          int informacion =raiz->info;
41          struct nodo *bor = raiz;
42          raiz=raiz->sig;
43          free(bor);
44          return informacion;
45      }
46      else
47      {
48          return -1;
49      }
50  }
51
52  void liberar()
53  {
54      tnodo reco = raiz;
55      tnodo bor;
56      while (reco!=NULL)
57      {
58          bor = reco;
59          reco = reco->sig;
60          free(bor);
61      }
62  }
```

Los algoritmos ya han sido vistos anteriormente, lo nuevo es como accedemos al puntero raiz definido en el otro archivo (main.c)

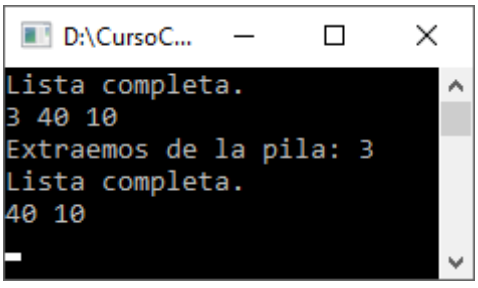
Mediante la palabra clave extern indicamos al compilador que hay otro archivo que define la variable raiz y queremos acceder a la misma en este archivo.

```
extern tnodo raiz;
```

Si consultamos el código fuente del sistema operativo Linux en muchos de sus archivos encontraremos especificación a variables de tipo extern:

```
extern struct kmem_cache *blk_requestq_cachep;
extern struct kmem_cache *request_cachep;
extern struct kobj_type blk_queue_ktype;
extern struct ida blk_queue_ida;
```

Si ejecutamos este será el resultado:



Capítulo 209.- Modificador inline en la definición de funciones

El lenguaje C se utiliza fundamentalmente en el desarrollo de software de base: sistemas operativos, navegadores web, gestores para visualizar videos, procesadores de texto, etc.

Es fundamental que este tipo de programas utilice en forma muy eficiente los recursos de la computadora donde se ejecuta.

Hemos visto que es responsabilidad del programador acceder correctamente a los componentes de un vector, gestionar la administración de la memoria, etc.

Otra herramienta más en busca de hacer que nuestro programa sea lo más eficiente es permitirnos indicar al compilador que una función se inserte su código en lugar de ser llamada.

Para indicar al compilador que inserte el algoritmo de la función se utiliza la palabra clave inline previo a la definición de la misma.

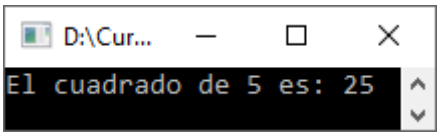
El modificador inline se hace fundamentalmente en funciones que realiza algoritmos muy cortos.

Problema

Confeccionar una función que retorne el cuadrado de un número. Indicar que dicha función sea de tipo inline.

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   static inline int calcularCuadrado(int x)
5   {
6       return x * x;
7   }
8
9   int main()
10  {
11      printf("El cuadrado de 5 es: %i", calcularCuadrado(5));
12      getch();
13      return 0;
14  }
```

Este será el resultado:



Con el modificador inline en la definición calcularCuadrado estamos solicitando al compilador que en lugar de llamar a la función sustituya el algoritmo de la misma, como si nos hubiéramos codificado:

```
printf("El cuadrado de 5 es: %i", 5 * 5);
```

Ahorrarse el llamado a una función hace que nuestro programa se ejecute en forma más rápida.

Si consultamos el código fuente del sistema operativo Linux en muchos de sus archivos encontraremos definiciones de funciones inline:

```
static inline void bsg_set_block(struct bsg_device *bd, struct file
{
    if (file->f_flags & O_NONBLOCK)
        clear_bit(BSG_F_BLOCK, &bd->flags);
    else
        set_bit(BSG_F_BLOCK, &bd->flags);
}
```

Capítulo 210.- Archivos binarios: Creación y grabación de tipo de datos primitivos (fopen, fwrite, fclose)

Hemos aprendido distintas estructuras de datos que nos permiten administrar información en la memoria ram de nuestro ordenador.

Ahora veremos otra estructura de datos que nos permite almacenar datos en una memoria secundaria de nuestro ordenador como podría ser el disco duro, pendrive, etc.

Esta estructura de datos son los archivos. Un archivo veremos que es independiente a nuestro programa y podrá ser accedido luego por otros programas.

Un archivo podemos imaginarlo en forma similar a un vector. También en un archivo los distintos componentes (bytes) se ubican uno a continuación del otro, yal igual que en los vectores, los componentes de un archivo en C también son automáticamente numeraos con índices, desde el 0 (cero) en adelante. Pero la forma de procesar un archivo difiere totalmente con respecto a un vector.

Podemos clasificar los archivos según el formato como se guardan los datos en:

- Archivos binarios
- Archivos de texto

En este y próximos capítulos veremos cómo administrar los archivos binarios y luego cuando veamos los archivos de texto veremos sus semejanzas y diferencias.

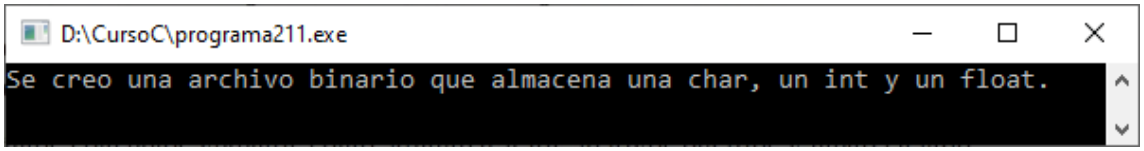
Problema

Crear un archivo binario y almacenar un carácter, un entero y un float. Indicar en el disco duro el archivo creado y cuál es su tamaño en bytes.

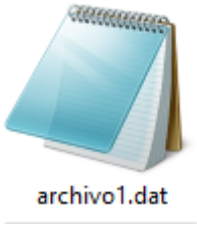
```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

int main()
{
    FILE *arch;
    arch=fopen("archivo1.dat", "wb");
    if (arch==NULL)
        exit(1);

    char letra='A';
    fwrite(&letra, sizeof(char), 1, arch);
    int valor1=12;
    fwrite(&valor1, sizeof(int), 1, arch);
    float valor2=5.25;
    fwrite(&valor2, sizeof(float), 1, arch);

    fclose(arch);
    printf("Se creo una archivo binario que ");
    printf("almacena una char, un int y un float.");
```

```
23      return 0;
24  }
25
```

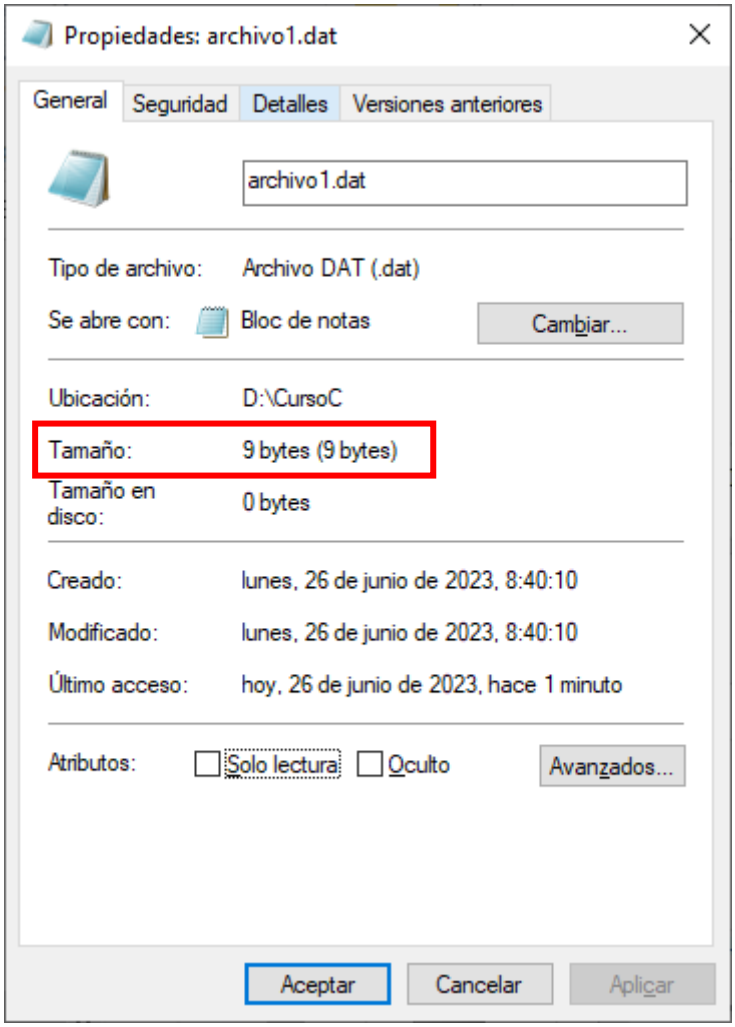
Este será si ejecutamos:



Y ha creado el siguiente archivo:



Si le damos con el botón derecho y seleccionamos propiedades:

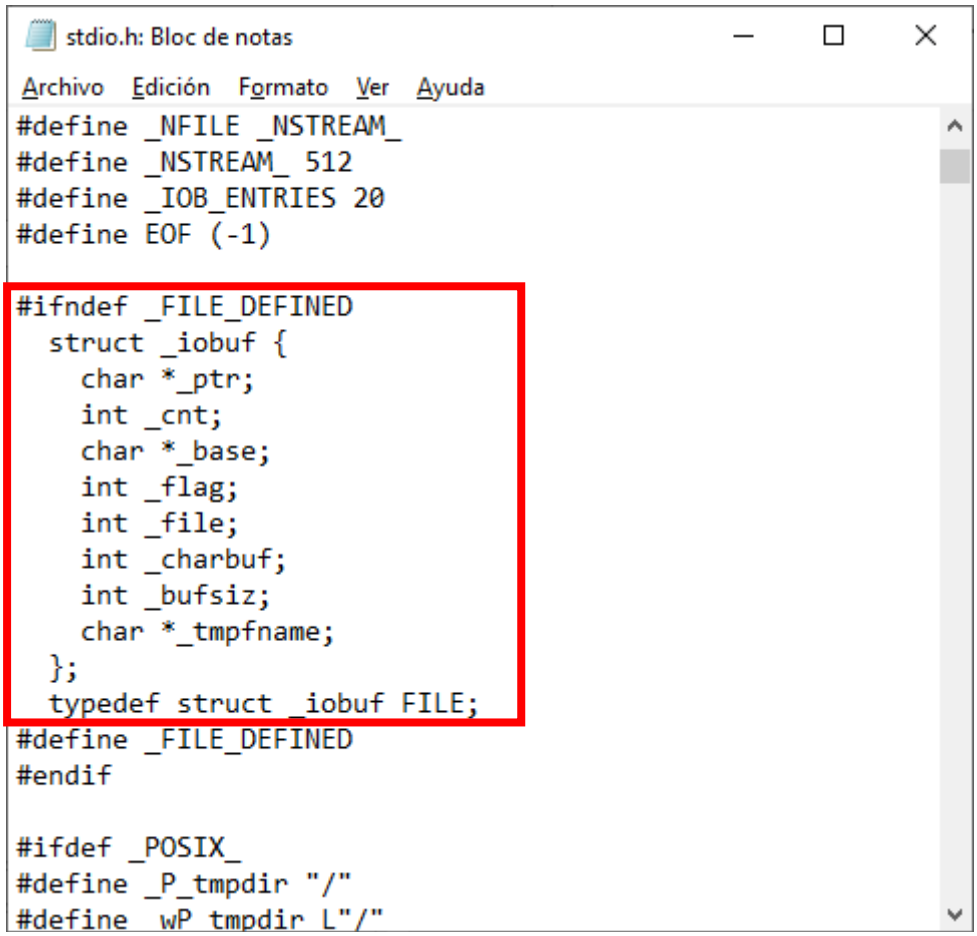


Podremos ver el tamaño del archivo.

Existe un conjunto de funciones que nos permiten administrar archivos binarios en el lenguaje C y se encuentran declaradas en el archivo de inclusión: #include<stdio.h>.

Además para trabajar con archivos debemos definir un puntero del tipo FILE que se encuentra declarado también en #include<stdio.h>.

Podemos abrir con un editor de texto el archivo stdio.h (se encuentra en C:\Program Files\CodeBlocks\MinGW\x86_64-w64-mingw32\include) y ubicar en su interior la declaración del tipo FILE.



La estructura de este registro no nos importa ya que son las funciones que provee la librería stdio.h las que las utilizan.

Entonces volvamos a nuestro programa, lo primero que hacemos es definir un puntero del tipo FILE:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

int main()
{
    FILE *arch;
```

Una vez definida la variable arch procedemos a llamar a la función fopen con do parámetros que indican el nombre físico del archivo ("archivo1.dat" y la forma de apertura y tipo de archivo:

```
arch=fopen("archivo1.dat","wb");
```

El primer parámetro indicamos el nombre del archivo a crear, debe ser un nombre válido para el sistema operativo donde se ejecuta.

Con "wb" estamos pidiendo a fopen que cree un archivo binario (si ya existe se borra el actual y se crea uno nuevo vacío).

Finalmente la función fopen retorna un puntero de tipo FILE y se almacena en la variable arch (arch es el nombre lógico del archivo que hacemos referencia dentro de nuestro programa y se relaciona con el archivo "archivo1.dat").

Es importante controlar si el archivo no se pudo crear verificando si el puntero almacena el valor NULL:

```
if (arch==NULL)
    exit(1);
```

El archivo podría no crearse porque el disco está lleno, porque hay otro archivo con el mismo nombre y esta siendo procesado por otro programa, etc.

Si en arch se almacena un valor distinto a NULL luego ya tenemos creado el archivo "archivo1.dat" listo para grabar datos en el mismo.

Inmediatamente luego que se crea el archivo no contiene información, para almacenar datos en el archivo utilizaremos la función fwrite:

```
char letra='A';
fwrite(&letra, sizeof(char), 1, arch);
```

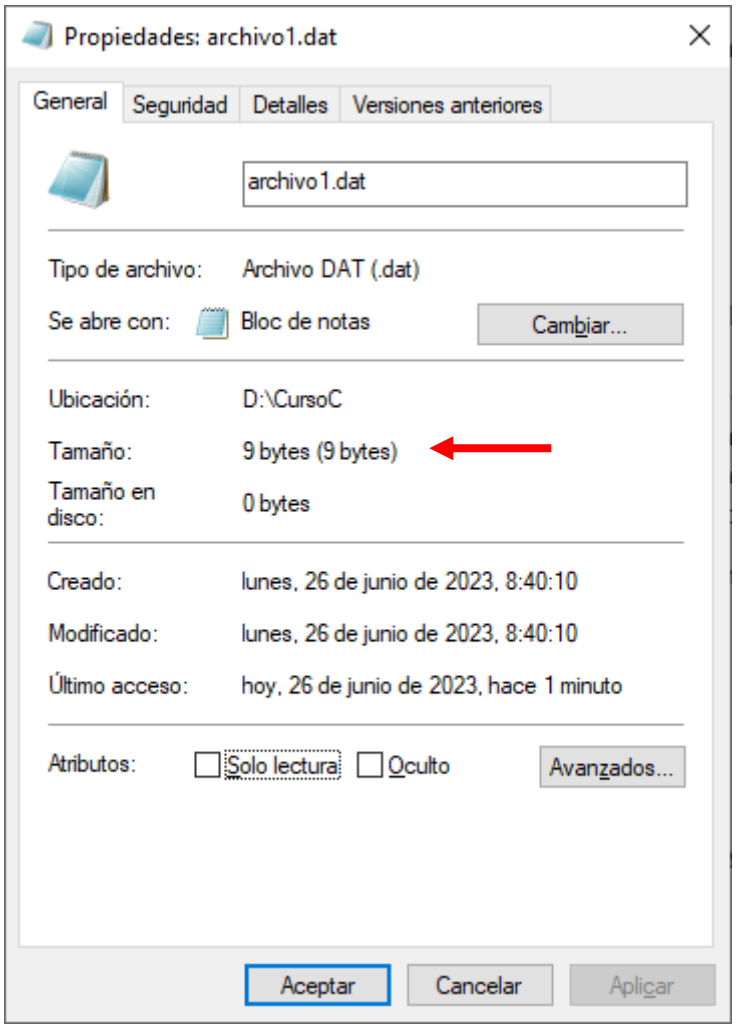
La función fwrite tiene 4 parámetros:

- Al primer parámetro hay que pasar la dirección de la variable a grabar el archivo.
- Al segundo parámetro hay que pasar la cantidad de bytes del tipo de dato a grabar. Normalmente utilizamos el operador sizeof para recuperar el tamaño.
- El tercer parámetro le pasamos un 1 ya que guardamos solo una variable (tendrá sentido cuando grabemos todo un vector con una sola llamada).
- El cuarto parámetro es el nombre lógico del archivo.

Si volvemos a llamar a la función fwrite el nuevo dato se almacena en el archivo a continuación del último grabado:

```
int valor1=12;
fwrite(&valor1, sizeof(int), 1, arch);
float valor2=5.25;
fwrite(&valor2, sizeof(float), 1, arch);
```

Como hemos grabado un char (1 byte), un int (4 bytes) y un float (4 bytes) luego podemos verificar que el archivo creado tiene 9 bytes de tamaño:



Cuando no necesitamos trabajar más con el archivo llamamos a la función fclose para liberar el archivo y que puede ser procesado por otro programa:

```
fclose(arch);
```

Es fundamental entender que el archivo "archivo1.dat" no se pierde cuando finaliza la ejecución del programa.

Capítulo 211.- Archivos binarios: lectura de datos primitivos (fread)

Vimos en el capítulo anterior como crear archivos binarios y grabar tipos de datos primitivos empleando las funciones que provee el lenguaje C.

Ahora veremos como leer los datos contenidos en un archivo binario.

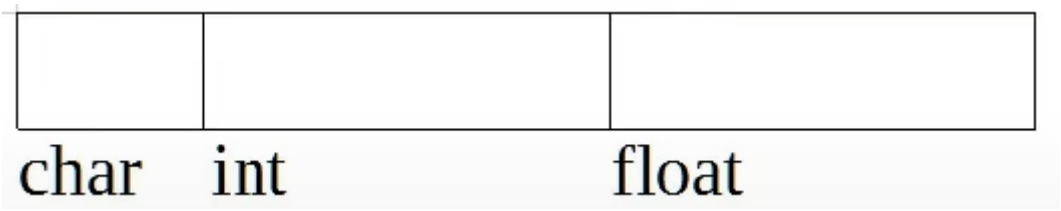
Para trabajar con los datos almacenados en un archivo debemos hacer una copia de los mismo en la memoria ram.

Debemos conocer los tipos de datos almacenados en el archivo binario y el orden exacto como están guardados.

Problema

Abrir el archivo binario que se creó en el capítulo anterior: "archivo1.dat" y leer sus datos.

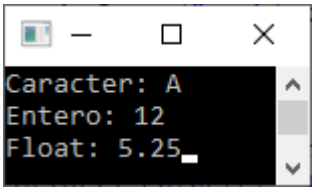
Recordamos que el archivo almacena un carácter, un entero y un float en este orden:



Si no conocemos la estructura interna del archivo binario (en nuestro ejemplo un char, un int y finalmente un float) será imposible recuperar correctamente la información almacenada.

```c
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

int main()
{
    FILE *arch;
    arch=fopen("archivo1.dat", "rb");
    if(arch==NULL)
        exit(1);
    char c;
    fread(&c, sizeof(char), 1, arch);
    printf("Caracter: %c\n", c);
    int v1;
    fread(&v1, sizeof(int), 1, arch);
    printf("Entero: %i\n", v1);
    float v2;
    fread(&v2, sizeof(float), 1, arch);
    printf("Float: %0.2f",v2);
    fclose(arch);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 212.- Archivos binarios: desplazamiento del puntero de archivo (fseek) – 1

En el capítulo anterior vimos que cuando abrimos un archivo binario para su lectura el puntero de archivo se posicione en el byte 0 y a medida que llamamos a la función fread el puntero del archivo se desplaza.

Veremos ahora una función que nos permite desplazar el puntero de archivo a cualquier posición previo a la lectura.

Con el desplazamiento del puntero de archivo no estamos obligados a leer en forma secuencial los datos.

La función que nos permite desplazar el puntero del archivo se llama fseek y tiene tres parámetros:

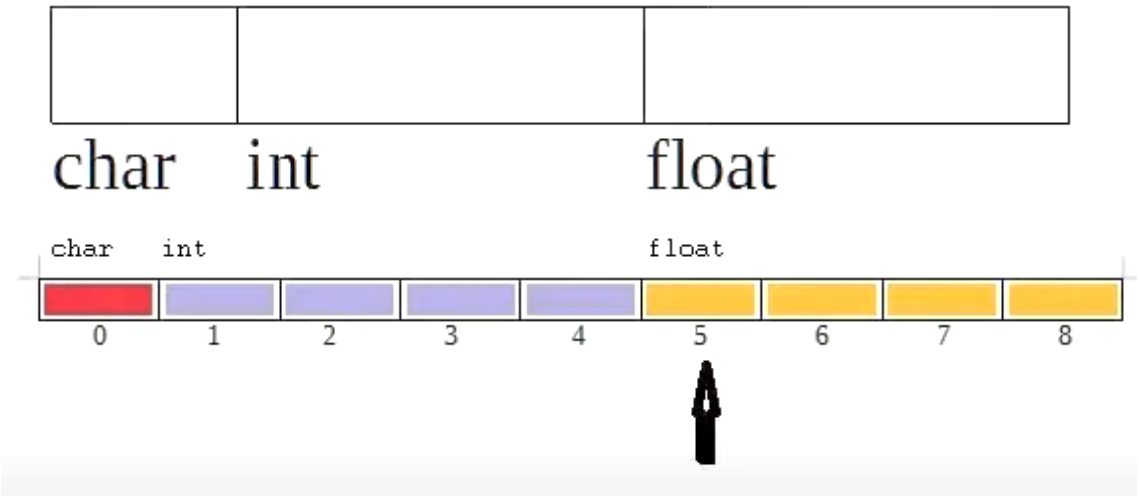
```
fseek([nombre lógico del archivo], [cantidad de bytes a desplazarse], [ubicación
desde donde comenzar a desplazarse])
```

El tercer parámetro pude tomar alguno de estos tres valores (son tres macros definidas en el archivo stdio.h):

- SEEK_SET Reposicionar comenzando desde el principio del archivo (es lo mismo pone un 0).
- SEEK_CUR Reposicionar comenzando desde la posición actual del puntero de archivo (es lo mismo que poner un 1).
- SEEK_END Reposicionar comenzando desde el final del archivo (es lo mismo que poner un 2).

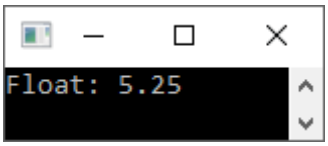
Problema

Abrir el archivo binario que se creó en el capítulo anterior: "archivo1.dat" y leer solo los bytes donde se almacena el valor float. Recordemos que el archivo almacena un carácter, un entero y un float en este orden.



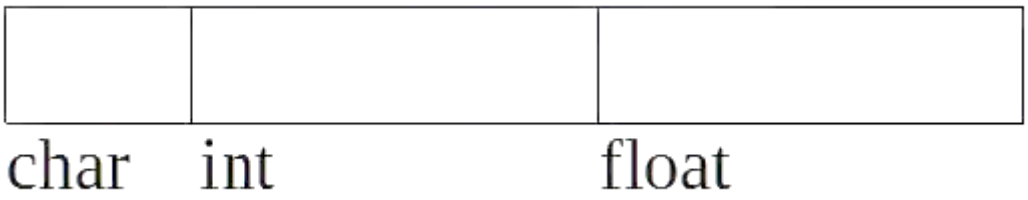
```c
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

int main()
{
    FILE *arch;
    arch=fopen("archivo1.dat", "rb");
    if (arch==NULL)
        exit(1);
    fseek(arch, 5, SEEK_SET);
    float v2;
    fread(&v2, sizeof(float), 1, arch);
    printf("Float: %0.2f", v2);
    fclose(arch);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Problema

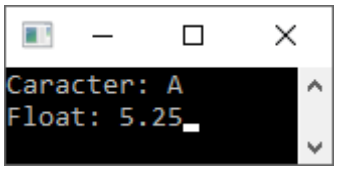
Abrir el archivo binario: "archivo1.dat" y leer e imprimir el char y el float.




```c
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

int main()
{
    FILE *arch;
    arch=fopen("archivo1.dat", "rb");
    if (arch==NULL)
        exit(1);
    char c;
    fread(&c, sizeof(char), 1, arch);
    printf("Caracter: %c\n", c);
    fseek(arch, 4, SEEK_CUR);
    float v2;
```

```
16      fread(&v2, sizeof(float), 1, arch);
17      printf("Float: %0.2f", v2);
18      fclose(arch);
19      getch();
20      return 0;
21  }
```

Si ejecutamos este será el resultado:



Capítulo 213.- Archivos binarios: desplazamiento del puntero de archivo (fseek) – 2

Problemas propuestos

1.- Crear un archivo llamado "archivo2.dat". Solicitar la carga de dos enteros y dos float por teclado. Almacenar en el archivo los dos enteros y seguidamente los dos float.

```c
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

int main()
{
    FILE *arch;
    arch=fopen("archivo2.dat", "wb");
    if(arch==NULL)
        exit(1);
    int numInt1;
    printf("Ingrese el primer valor entero: ");
    scanf("%i", &numInt1);
    fwrite(&numInt1, sizeof(int), 1, arch);
    int numInt2;
    printf("Ingrese el segundo valor entero: ");
    scanf("%i", &numInt2);
    fwrite(&numInt2, sizeof(int), 1, arch);
    float numFloat1;
    printf("Ingrese el primer valor float: ");
    scanf("%f", &numFloat1);
    fwrite(&numFloat1, sizeof(float), 1, arch);
    float numFloat2;
    printf("Ingrese el segundo valor float: ");
    scanf("%f", &numFloat2);
    fwrite(&numFloat2, sizeof(float), 1, arch);

    fclose(arch);
    printf("Se creo el archivo binario 'archivo2.dat'");
    return 0;
}
```

Cuando ejecutamos este será el resultado:

```
D:\CursoC\programa215_1.exe
Ingrese el primer valor entero: 5
Ingrese el segundo valor entero: 10
Ingrese el primer valor float: 20.55
Ingrese el segundo valor float: 30.75
Se creo el archivo binario 'archivo2.dat'
Process returned 0 (0x0)   execution time : 16.245 s
Press any key to continue.
```

archivo2.dat

Otra posible solución:

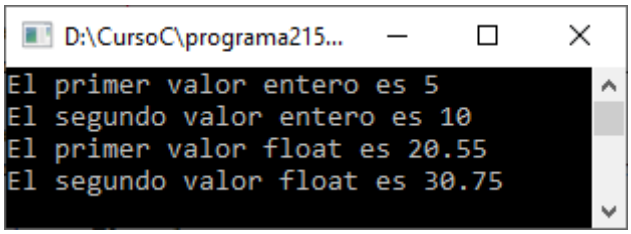
```c
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

int main()
{
    int v1,v2;
    float f1,f2;
    printf("Ingrese el primer entero:");
    scanf("%i",&v1);
    printf("Ingrese el segundo valor:");
    scanf("%i",&v2);
    printf("Ingrese el primer float:");
    scanf("%f",&f1);
    printf("Ingrese el segundo float:");
    scanf("%f",&f2);
    FILE *arch;
    arch=fopen("archivo2.dat","wb");
    if(arch==NULL)
        exit(1);
    fwrite(&v1,sizeof(int),1,arch);
    fwrite(&v2,sizeof(int),1,arch);
    fwrite(&f1,sizeof(float),1,arch);
    fwrite(&f2,sizeof(float),1,arch);
    fclose(arch);
    printf("Se creo el archivo binario y se almacenaron los 4 datos");
    getch();
    return 0;
}
```

2.- Abrir el archivo "archivo2.dat" creado en el ejercicio anterior y proceder a su lectura e impresión.

```c
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

int main()
{
    FILE *arch;
    arch=fopen("archivo2.dat", "rb");
    if (arch==NULL)
        exit(1);
    int numInt1;
    fread(&numInt1, sizeof(int), 1, arch);
    printf("El primer valor entero es %i\n", numInt1);
    int numInt2;
    fread(&numInt2, sizeof(int), 1, arch);
    printf("El segundo valor entero es %i\n", numInt2);

    float numFloat1;
    fread(&numFloat1, sizeof(float), 1, arch);
    printf("El primer valor float es %0.2f\n", numFloat1);
    float numFloat2;
    fread(&numFloat2, sizeof(float), 1, arch);
    printf("El segundo valor float es %0.2f\n", numFloat2);
```

```
24
25          fclose(arch);
26          getch();
27          return 0;
28      }
```

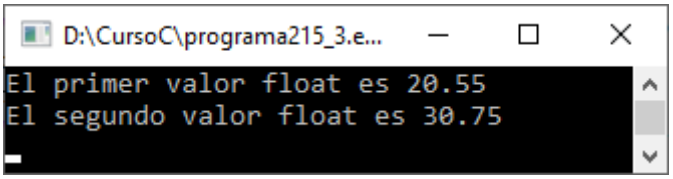
Si ejecutamos este será el resultado:



3.- Abrir el archivo "archivo2.dat" e imprimir solo los dos valores de tipo float almacenados.

```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<stdlib.h>
4
5   int main()
6   {
7       FILE *arch;
8       arch=fopen("archivo2.dat", "rb");
9       if (arch==NULL)
10          exit(1);
11
12      fseek(arch, 8, SEEK_SET);
13      float numFloat1;
14      fread(&numFloat1, sizeof(float), 1, arch);
15      printf("El primer valor float es %0.2f\n", numFloat1);
16      float numFloat2;
17      fread(&numFloat2, sizeof(float), 1, arch);
18      printf("El segundo valor float es %0.2f\n", numFloat2);
19
20      fclose(arch);
21      getch();
22      return 0;
23  }
```

Si ejecutamos este será el resultado:



Capítulo 214.- Archivos binarios: agregar datos

Hemos visto que cuando necesitamos crear un archivo lo hacemos pasando a la función fopen el valor "wb". Con esto indicamos que se cree el archivo y si ya existe lo borre y lo cree nuevamente vacío.

En muchas situaciones podemos necesitar agregar datos al archivo sin tenerlo que borrar, para esto debemos pasar a la función fopen el valor "ab" (append binary).

Problema

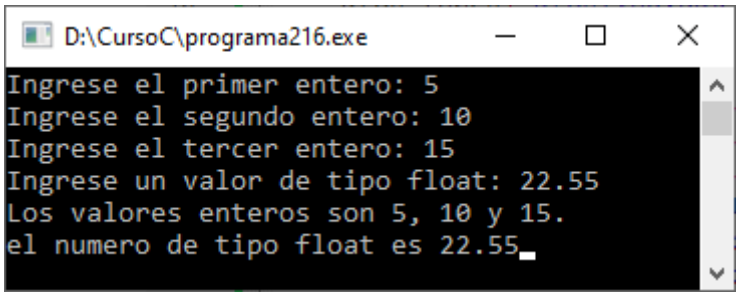
Confeccionar un programa con tres funciones:

1.- Creación de un archivo binario llamado "archivo3.dat" y almacenar 3 enteros. Seguidamente cerrar el archivo.
2.- Abrir nuevamente el archivo pero en modo añadir y almacenar 1 float.
3.- finalmente abrir el archivo para lectura y mostrar todos sus datos.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

void crear()
{
    int num1, num2, num3;
    FILE *arch;
    arch=fopen("archivo3.dat", "wb");
    if(arch==NULL)
        exit(1);
    printf("Ingrese el primer entero: ");
    scanf("%i", &num1);
    printf("Ingrese el segundo entero: ");
    scanf("%i", &num2);
    printf("Ingrese el tercer entero: ");
    scanf("%i", &num3);
    fwrite(&num1, sizeof(int), 1, arch);
    fwrite(&num2, sizeof(int), 1, arch);
    fwrite(&num3, sizeof(int), 1, arch);
    fclose(arch);
}

void agregar()
{
    float numero1;
    FILE *arch;
    arch=fopen("archivo3.dat", "ab");
    if(arch==NULL)
        exit(1);
    printf("Ingrese un valor de tipo float: ");
    scanf("%f", &numero1);
    fwrite(&numero1, sizeof(float), 1, arch);
    fclose(arch);
}
```

```
36
37  void lectura()
38  {
39      int num1, num2, num3;
40      float numero1;
41      FILE *arch;
42      arch=fopen("archivo3.dat", "rb");
43      if(arch==NULL)
44          exit(1);
45      fread(&num1, sizeof(int), 1, arch);
46      fread(&num2, sizeof(int), 1, arch);
47      fread(&num3, sizeof(int), 1, arch);
48      fread(&numero1, sizeof(float), 1, arch);
49      printf("Los valores enteros son %i, %i y %i.\n", num1, num2, num3);
50      printf("el numero de tipo float es %0.2f", numero1);
51      fclose(arch);
52  }
53
54  int main()
55  {
56      crear();
57      agregar();
58      lectura();
59      getch();
60      return 0;
61  }
62
```

Si ejecutamos este será el resultado:



Capítulo 215.- Archivos binarios: modificar datos

Hemos visto hasta ahora como se crea un archivo binario desde C, como se lo lee, como se agregan datos al final. Nos está quedando como modificar datos almacenados en el archivo.

Listado completo de modos de apertura de archivos binarios:

- "rb" Sólo lectura. El archivo debe existir previamente. Si no existe, fopen retorna NULL.
- "wb" Sólo escritura. Si el archivo no existe, lo crea. Si existe, destruye su contenido y lo abre vacío.
- "ab" Sólo escritura, pero en modo append (o sea: añadir). Los datos que se graban, lo hacen al final. Crear el archivo si no existe y no destruye el contenido si lo hubiera.
- "r+b" Permite lectura y escritura. El archivo debe existir previamente.
- "w+b" Permite escritura y lectura. Crea el archivo si no existe, pero destruye su contenido si ya existía, y lo abre vacío.
- "a+b" Permite añadir al final, en modo lectura – escritura. Crea el archivo si no existe. No destruye el contenido si hubiera.

Para poder modificar datos de un archivo binario procedemos a su apertura en modo "r+b".

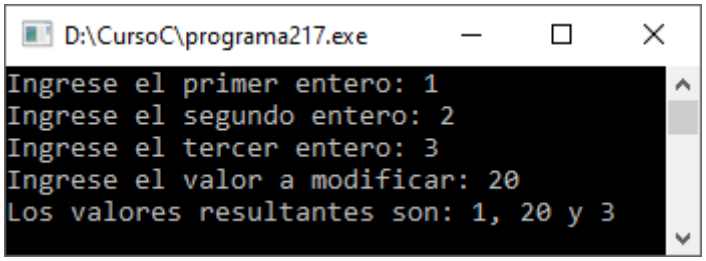
Problema

Confeccionar un programa con tres funciones:
1.- Creación de un archivo binario llamado "archivo4.dat" y almacenar 3 enteros. Seguidamente cerrar el archivo.
2.- Abrir nuevamente el archivo pero en modo "r+b" y modificamos el segundo entero almacenado en el archivo.
3.- Finalmente abrir el archivo para lectura y mostrar todos sus datos.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

void crear()
{
    int num1, num2, num3;
    FILE *arch;
    arch=fopen("archivo4.dat", "wb");
    if (arch==NULL)
        exit(1);
    printf("Ingrese el primer entero: ");
    scanf("%i", &num1);
    printf("Ingrese el segundo entero: ");
    scanf("%i", &num2);
    printf("Ingrese el tercer entero: ");
    scanf("%i", &num3);
    fwrite(&num1, sizeof(int), 1, arch);
    fwrite(&num2, sizeof(int), 1, arch);
    fwrite(&num3, sizeof(int), 1, arch);
    fclose(arch);
}
```

```c

void modificar()
{
    int num4;
    FILE *arch;
    arch=fopen("archivo4.dat", "r+b");
    if (arch==NULL)
        exit(1);
    printf("Ingrese el valor a modificar: ");
    scanf("%i", &num4);
    fseek(arch, 4, SEEK_SET);
    fwrite(&num4, sizeof(int), 1, arch);
    fclose(arch);
}

void lectura()
{
    int num1, num2, num3;
    FILE *arch;
    arch=fopen("archivo4.dat", "rb");
    if (arch==NULL)
        exit(1);
    fread(&num1, sizeof(int), 1, arch);
    fread(&num2, sizeof(int), 1, arch);
    fread(&num3, sizeof(int), 1, arch);
    printf("Los valores resultantes son: %i, %i y %i",
           num1, num2, num3);
    fclose(arch);
    printf("\n");
}

int main()
{
    crear();
    modificar();
    lectura();
    getch();
    return 0;
}

```

Si ejecutamos este será el resultado:



Capítulo 216.- Archivos binarios: identificar final de archivo (feof) – 1

Hemos visto la función fread que nos permite recuperar datos de un archivo binario. Ahora veremos otra función disponible en stdio.h que nos permite que mientras utilizamos la función fread verificar si se ha llegado al final del archivo.

```
int feof([nombre lógico del archivo])
```

Esta función devuelve 0 si no es fin de archivo y un valor distinto a 0 si se ha llegado al final del archivo.

Problema

Confeccionar un programa con dos funciones:

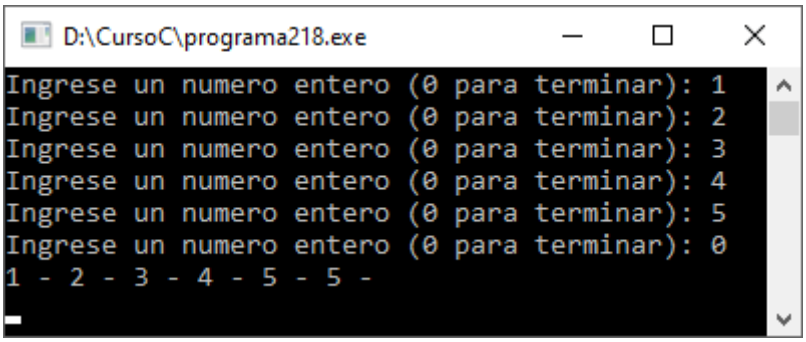
1.- Creación de un archivo binario llamado "archivo5.dat" y almacenar valores enteros ingresados por teclado.

2.- Imprimir el archivo en forma completa.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

void almacenar()
{
    int num;
    FILE *arch;
    arch=fopen("archivo5.dat", "wb");
    if (arch==NULL)
        exit(1);
    do{
        printf("Ingrese un numero entero (0 para terminar): ");
        scanf("%i", &num);
        if (num!=0)
        {
            fwrite(&num, sizeof(int), 1, arch);
        }

    }while(num!=0);
    fclose(arch);
}

void imprimir()
{
    int num;
    FILE *arch;
    arch=fopen("archivo5.dat", "rb");
    if(arch==NULL)
        exit(1);
    while (!feof(arch))
    {
        fread(&num, sizeof(int), 1, arch);
        printf("%i - ", num);
    }
```

```
36          printf("\n");
37          fclose(arch);
38      }
39
40
41      int main()
42      {
43          almacenar();
44          imprimir();
45          getch();
46          return 0;
47      }
48
```

Si ejecutamos este será el resultado:



Capítulo 217.- Archivos binarios: identificar final de archivo (feof) – 2

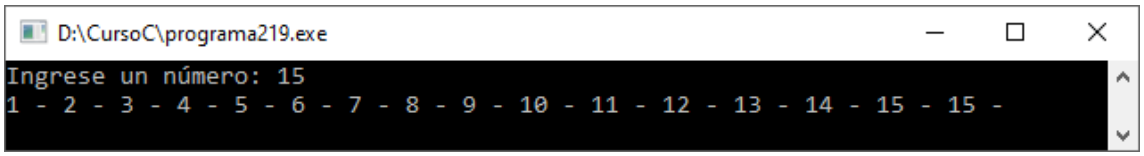
Problema propuesto

Confeccionar un programa que solicite el ingreso de un entero por teclado y luego grabe en un archivo los números comprendidos entre el 1 y el número ingresado de uno en uno. Ej. Si se ingresa un 10 luego se graban en el archivo 1, 2, 3, 4,5 6, 7, 8, 9 y 10 (nombre del archivo: "archivo6.dat").

Imprimir luego el contenido del archivo.

```c
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<locale.h>

void cargar()
{
    int num;
    printf("Ingrese un número: ");
    scanf("%i", &num);
    FILE  *arch;
    arch = fopen("archivo6.dat", "wb");
    if(arch==NULL)
        exit(1);
    for (int x=1; x<=num; x++)
    {
        fwrite(&x, sizeof(int), 1, arch);
    }
    fclose(arch);
}

void imprimir()
{
    int num;
    FILE *arch;
    arch=fopen("archivo6.dat", "rb");
    if(arch==NULL)
        exit(1);
    while(!feof(arch))
    {
        fread(&num, sizeof(int), 1, arch);
        printf("%i - ", num);
    }
    fclose(arch);
}

int main()
{
    setlocale(LC_ALL, "");
    cargar();
    imprimir();
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



Capítulo 218.- Archivos binarios: posición actual del puntero de archivo – ftell – 1

Hemos dicho en capítulos anteriores que cuando trabajamos con archivos aparece el puntero de archivo que nos indica donde estamos posicionados para proceder a la lectura de un dato o grabación.

Mediante la función fseek podemos desplazar el puntero de archivo a cualquier byte del mismo, como vimos si queremos grabar datos al final lo desplazamos con el fseek fácilmente al final.

Ahora veremos que hay una función que nos informa en que byte del archivo está el puntero:

```
int ftell([nombre lógico del archivo])
```

Problema

Confeccionar un programa con dos funciones:

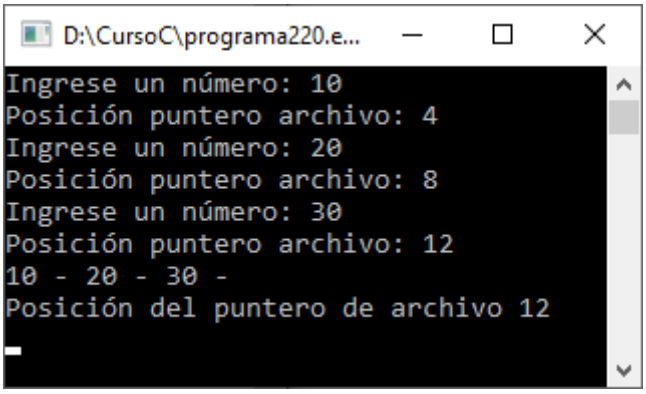
1.- Creación de un archivo binario llamado "archivo7.dat" y grabar 3 enteros, después de grabar cada dato imprimir la posición del puntero de archivo.

2.- Imprimir el archivo en forma completa, mostrar además la posición del puntero de archivo después de leer.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<locale.h>

void creacion()
{
    FILE *arch;
    arch = fopen("archivo7.dat", "wb");
    if(arch==NULL)
        exit(1);
    int num;
    for (int x=0; x<3; x++)
    {
        printf("Ingrese un número: ");
        scanf("%i", &num);
        fwrite(&num, sizeof(int), 1, arch);
        printf("Posición puntero archivo: %i\n", ftell(arch));
    }
    fclose(arch);
}

```

```c
void imprimir()
{
    int num;
    FILE *arch;
    arch = fopen("archivo7.dat", "rb");
    if(arch==NULL)
        exit(1);
    fread(&num, sizeof(int), 1, arch);
    while(!feof(arch))
    {
        printf("%i - ", num);
        fread(&num, sizeof(int), 1, arch);
    }
    printf("\n");
    printf("Posición del puntero de archivo %i\n", ftell(arch));
    fclose(arch);
}

int main()
{
    setlocale(LC_ALL,"");
    creacion();
    imprimir();
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



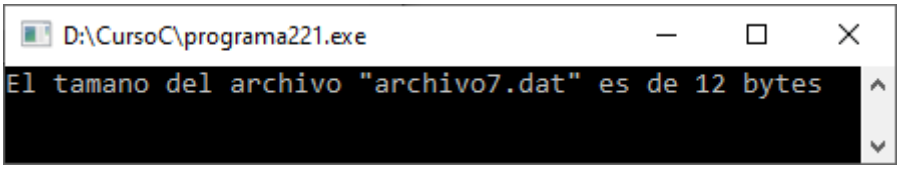
Problema

Confeccionar un programa que imprima el tamaño en bytes del archivo "archivo7.dat" creado en el programa anterior.

```c
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

int main()
{
    FILE *arch;
    arch = fopen("archivo7.dat", "rb");
    if(arch==NULL)
        exit(1);
    fseek(arch, 0, SEEK_END);
```

```
12      printf("El tamano del archivo \"archivo7.dat\" es de ");
13      printf("%i bytes", ftell(arch));
14      fclose(arch);
15      getch();
16      return 0;
17  }
18
```

Si ejecutamos este será el resultado:



Capítulo 219.- Archivos binarios: grabar y leer vectores completos en un archivo.

Hasta ahora siempre que hemos utilizado la función fread y fwrite en el tercer parámetro hemos pasado el valor 1:

```
char letra='A';
fwrite(&letra, sizeof(char), 1, arch);

int valor1=12;
fwrite(&valor1, sizeof(int), 1, arch);

char c;
fread(&c, sizeof(char), 1, arch);
```

Veremos ahora como necesitamos grabar o leer un vector en una sola llamada a la función fread o fwrite indicando en dicho tercer parámetro el tamaño del vector.

Problema

Confeccionar un programa con dos funciones:

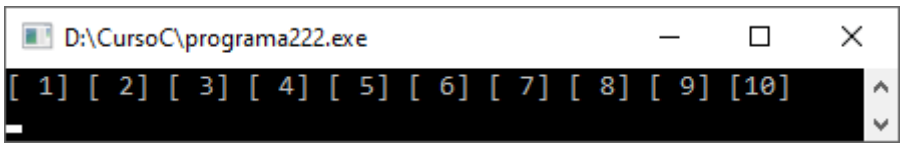
1.- Crear un vector de 10 enteros y grabar dicho vector en un archivo binario llamado "archivo8.dat".

2.- Recuperar todos los datos del archivo creado anteriormente y almacenar los datos en un vector.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

void cargar(int vec[10])
{
    FILE *arch;
    arch = fopen("archivo8.dat", "wb");
    if (arch==NULL)
        exit(1);
    fwrite(vec, sizeof(int), 10, arch);
    fclose(arch);
}

void retornar(int vec[10])
{
    int v[10];
    FILE *arch;
    arch = fopen("archivo8.dat", "rb");
    if (arch==NULL)
        exit(1);
    fread(&v, sizeof(int), 10, arch);
    fclose(arch);
    for (int x=0; x<10; x++)
```

```
25      {
26          printf("[%2i] ",v[x]);
27      }
28      printf("\n");
29  }
30  int main()
31  {
32      int vector[10] = {1,2,3,4,5,6,7,8,9,10};
33      cargar(vector);
34      retornar(vector);
35      getch();
36      return 0;
37  }
```

Si ejecutamos este será el resultado:



Capítulo 220.- Archivos binarios: agregar, consultar y modificar registros (struct) en un archivo

hemos visto que en un archivo binario podemos guardar tipos de datos primitivos y vectores, ahora veremos como administrar un archivo binario almacenando registros (struct).

Problema

Se tiene el siguiente struct:

```
typedef struct {
    int codigo;
    char descripcion[41];
    float precio;
} tproducto;
```

Elabora el programa que permita:
1.- Crear un archivo binario llamado "producto.dat".
2.- Carga de registros de tipo tproducto.
3.- Listado completo de productos.
4.- Consulta de un producto por su código.
5.- Modificar el precio de un producto.

```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<stdlib.h>
4   #include<locale.h>
5   #include<string.h>
6
7   typedef struct {
8       int codigo;
9       char descripcion[41];
10      float precio;
11  }tproducto;
12
13  void crear()
14  {
15      FILE *arch;
16      arch = fopen("producto.dat", "wb");
17      if(arch==NULL)
18          exit(1);
19      fclose(arch);
20      printf("Archivo \"producto.dat\" creado correctamente.");
21      system("pause");
22  }
23
```

```c
void agregarRegistros()
{
    FILE *arch;
    arch = fopen("producto.dat", "ab");
    if(arch==NULL)
        exit(1);
    tproducto pro;
    printf("Ingrese el código: ");
    scanf("%i",&pro.codigo);
    fflush(stdin);
    printf("Ingrese la descripción: ");
    gets(pro.descripcion);
    printf("Ingrese el precio: ");
    scanf("%f",&pro.precio);
    fwrite(&pro, sizeof(tproducto), 1, arch);
    fclose(arch);
    printf("Registro agregado correctamente.");
    system("pause");
}

void imprimir()
{
    tproducto pro;
    FILE *arch;
    arch = fopen("producto.dat", "rb");
    if(arch==NULL)
        exit(1);
    printf("\n                  Listado general\n");
    printf("-------------------------------------------------------------\n");
    fread(&pro, sizeof(tproducto), 1, arch);
    while(!feof(arch))
    {
        printf("     Código: %i, Descripción: %s, precio: %0.2f\n",
               pro.codigo, pro.descripcion, pro.precio);
        fread(&pro, sizeof(tproducto), 1, arch);
    }
    fclose(arch);
    printf("-------------------------------------------------------------\n");
    printf("\n");
    system("pause");
}

void consulta()
{
    tproducto pro;
    FILE *arch;
    arch = fopen("producto.dat", "rb");
    if(arch==NULL)
        exit(1);
    int codi;
    printf("Ingrese el código a consultar: ");
    scanf("%i", &codi);
    printf("\n           Resultado de la consulta\n");
    printf("-------------------------------------------------------------\n");
    fread(&pro, sizeof(tproducto), 1, arch);
    int encontrado=0;
    while(!feof(arch))
    {
        if (codi==pro.codigo)
        {
            encontrado=1;
            printf("     Código: %i, Descripción: %s, precio: %0.2f\n",
              pro.codigo, pro.descripcion, pro.precio);
        }
        fread(&pro, sizeof(tproducto), 1, arch);
    }
    fclose(arch);
```

```
91      if(encontrado==0)
92          printf("  Este artículo no está disponeble....\n");
93      printf("\n");
94      system("pause");
95  }
96
97  void modificar()
98  {
99      tproducto pro;
100     tproducto prol;
101      FILE *arch;
102      arch = fopen("producto.dat", "r+b");
103      if(arch==NULL)
104          exit(1);
105      int codi;
106      printf("Ingrese el código a modificar: ");
107      scanf("%i", &codi);
108      printf("\n        Artículo a moidficar su precio\n");
109      printf("-------------------------------------------------------\n");
110      fread(&pro, sizeof(tproducto), 1, arch);
111      int encontrado=0;
112      int posicion;
113      while(!feof(arch))
114      {
115          if (codi==pro.codigo)
116          {
117              encontrado=1;
118              posicion= ftell(arch)-sizeof(tproducto);
119              printf("     Código: %i, Descripción: %s, precio: %0.2f\n",
120                 pro.codigo, pro.descripcion, pro.precio);
121                 prol.codigo=pro.codigo;
122                 strcpy(prol.descripcion,pro.descripcion);
123
124          }
125          fread(&pro, sizeof(tproducto), 1, arch);
126      }
127      if (encontrado==1)
128      {
129          float pre;
130          printf("Ingrese el nuevo precio: ");
131          scanf("%f", &pre);
132          prol.precio=pre;
133          fseek(arch, posicion, SEEK_SET);
134          fwrite(&prol, sizeof(tproducto), 1, arch);
135          printf("Precio actualizado correctamente...\n");
136      }
137     else
138     {
139         printf("  Este artículo no está disponeble....\n");
140         printf("\n");
141     }
142     fclose(arch);
143     system("pause");
144 }
145
146
147 void menu()
148 {
149      int op;
150      do{
151          system("Cls");
152          printf("|------------------------ MENU-------------------------|\n");
153          printf("| 1.- Cargar un archivo binario llamado \"producto.dat\"  |\n");
154          printf("| 2.- Carga de registos de tipo tproducto                |\n");
155          printf("| 3.- Listado completo de productos                      |\n");
156          printf("| 4.- Consulta de un producto por su código              |\n");
157          printf("| 5.- Modificar el precio de un producto                 |\n");
```

```
158             printf("| 6.- Salir                                                    |\n");
159             printf("|-------------------------------------------------------------|\n\n");
160             printf("¿Que opción deseas realizar: ");
161             fflush(stdin);
162             scanf("%i", &op);
163             switch(op){
164             case 1: crear();
165                     break;
166             case 2: agregarRegistros();
167                     break;
168             case 3: imprimir();
169                     break;
170             case 4: consulta();
171                     break;
172             case 5: modificar();
173                     break;
174             case 6: system("Cls");
175                     break;
176             default:
177                 printf("La opción introducida es incorrecta");
178                 system("pause");
179             }
180         }while(op!=6);
181
182     }
183
184 int main()
185 {
186     setlocale(LC_ALL, "");
187     menu();
188     getch();
189     return 0;
190 }
191
```

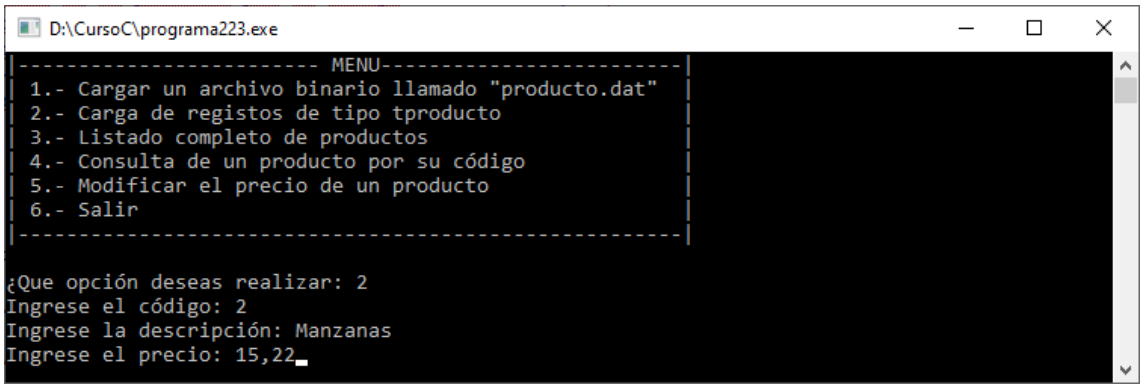
Si ejecutamos este será el resultado:

```
D:\CursoC\programa223.exe
|------------------------ MENU-------------------------|
| 1.- Cargar un archivo binario llamado "producto.dat"  |
| 2.- Carga de registos de tipo tproducto               |
| 3.- Listado completo de productos                     |
| 4.- Consulta de un producto por su código             |
| 5.- Modificar el precio de un producto                |
| 6.- Salir                                             |
|-------------------------------------------------------|

¿Que opción deseas realizar: 1
Archivo "producto.dat" creado correctamente.Presione una tecla para continuar . . .
```

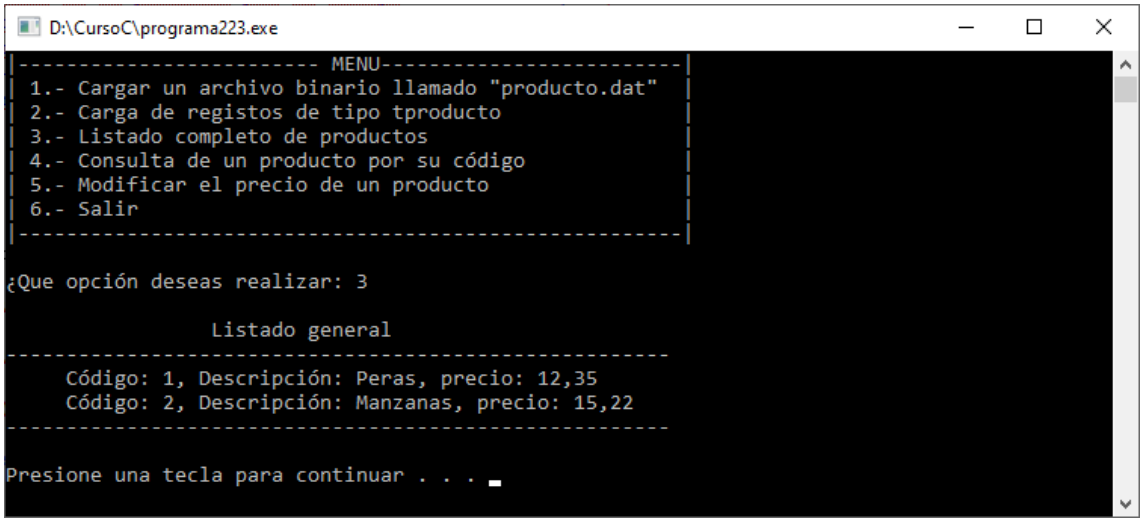
Seleccionamos la opción uno para crear el archivo binario.

```
D:\CursoC\programa223.exe
|------------------------ MENU-------------------------|
| 1.- Cargar un archivo binario llamado "producto.dat"  |
| 2.- Carga de registos de tipo tproducto               |
| 3.- Listado completo de productos                     |
| 4.- Consulta de un producto por su código             |
| 5.- Modificar el precio de un producto                |
| 6.- Salir                                             |
|-------------------------------------------------------|

¿Que opción deseas realizar: 2
Ingrese el código: 1
Ingrese la descripción: Peras
Ingrese el precio: 12,35
```

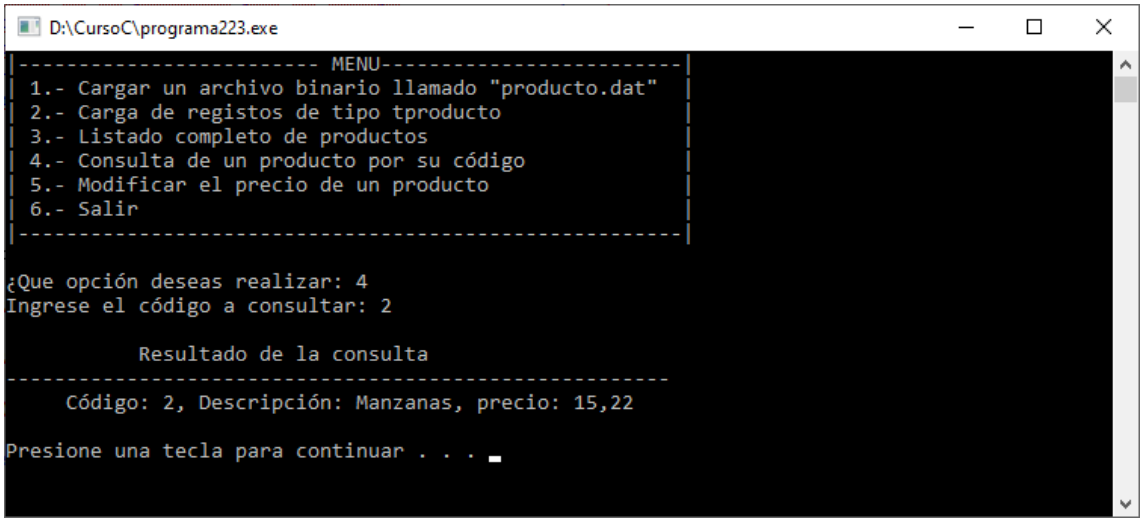
Seleccionamos la opción 2 para agregar un artículo.



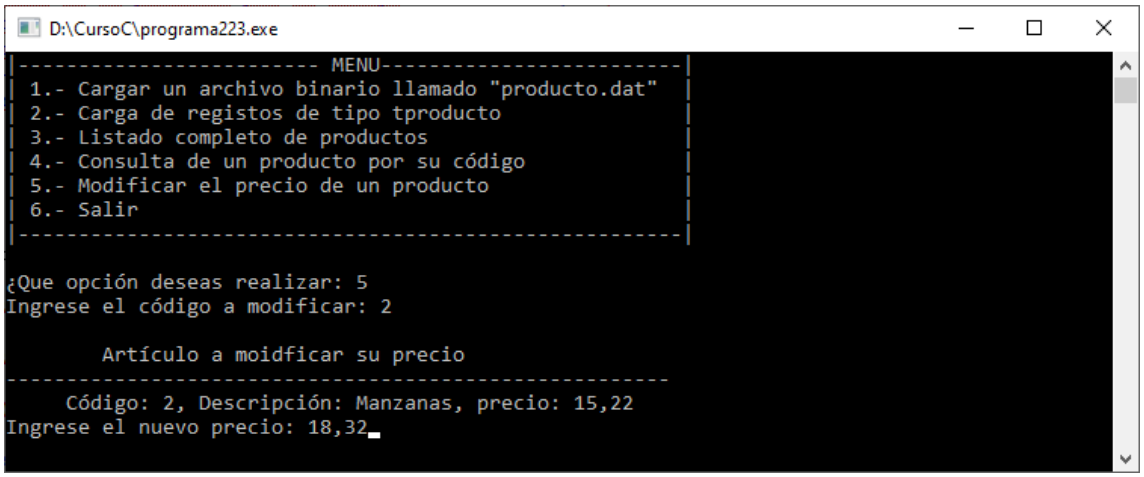
Seleccionamos la opción 2 de nuevo para agregar un segundo artículo.



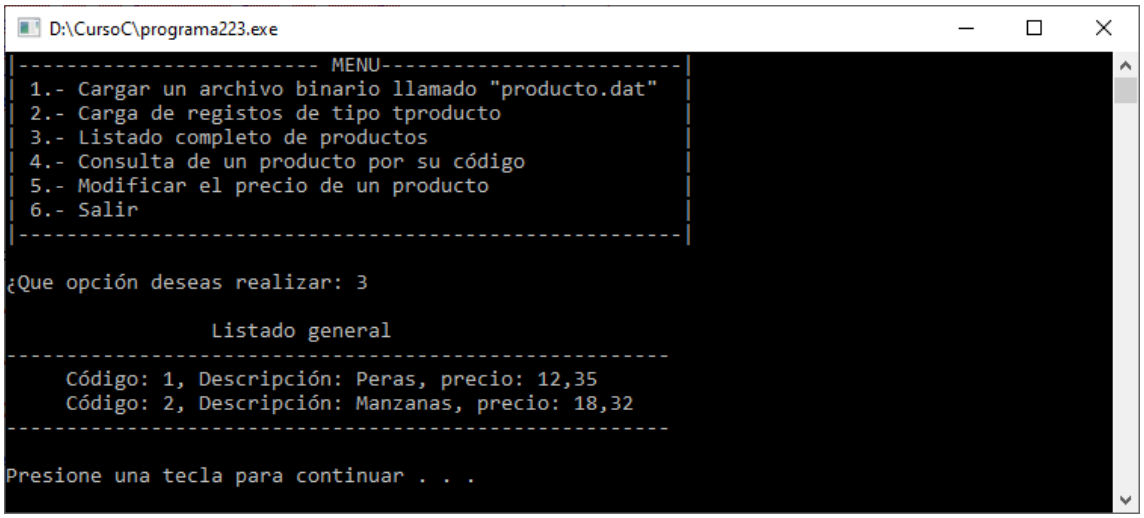
Seleccionamos la opción 3 para consultar por todos los artículos.



Seleccionamos la opción 5 para consular un artículo por su código.



Seleccionamos la opción 5 para modificar el precio de un artículo a partir de su código.



seleccionamos de nuevo la opción 3 para comprobar si el código 2 ha cambiado su precio.

Por último seleccionaremos la opción 6 para finalizar con la ejecución del programa.

Capítulo 221.- Archivos de texto: Creación y grabación de datos

Ahora veremos otro formato de archivo llamado "archivo de texto".

Un archivo de texto contiene solo caracteres legibles por el ser humano con la salvedad del salto de línea que le sirve a un editor de texto para comenzar a mostrar los siguientes caracteres una línea más abajo.

Es un formato muy utilizado los podemos encontrar en archivos de configuración, los archivos de código fuente en C u otros lenguajes, las páginas HTML, archivos XML, etc.

Un archivo de texto lo podemos editar (crear, agregar, modificar y borrar desde cualquier editor de texto).

El lenguaje C provee una serie de funciones para procesar en forma sencilla un archivo de texto.

Listado completo de modos de apertura de archivos de texto:

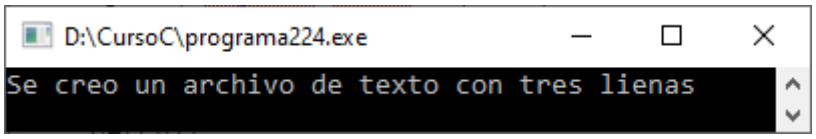
- "rt" Sólo lectura. El archivo debe existir previamente. Si no existe, fopen retorna NULL.
- "wt" Sólo escritura. Si el archivo no existe, lo crea. Si existe, destruye su contenido y lo abre vacío.
- "at" Sólo escritura, pero en modo append (o sea añadir): Los datos se graban, lo hacen al final. Crea el archivo si no existe, y no destruye su contenido si lo tiene.
- "r+t" Permite lectura y escritura. El archivo debe existir previamente.
- "w+t" Permite escritura y lectura. Crea el archivo si no existe, pero destruye su contenido si ya existe, y lo abre vacío.
- "a+t" Permite añadir al final, en modo lectura – escritura. Crea el archivo si no existe. no destruye su contenido si lo tiene.

Problema

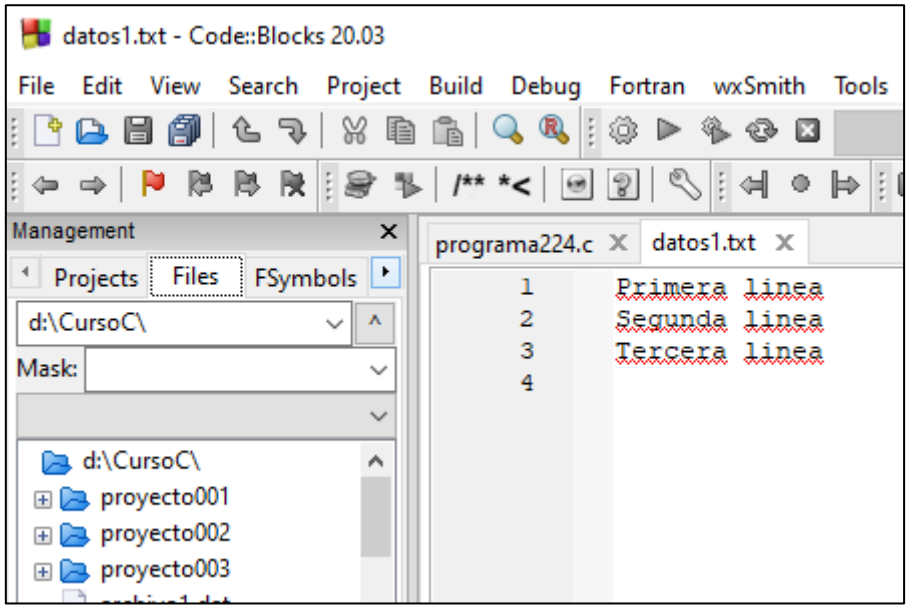
Crear un archivo de texto llamado "datos1.txt" y almacenar tres líneas. Acceder a su contenido luego con al menos dos editores de texto.

```
#include<conio.h>
#include<stdlib.h>
#include<stdio.h>

int main()
{
    FILE *arch;
    arch=fopen("datos1.txt", "wt");
    if (arch==NULL)
        exit(1);
    fputs("Primera linea\n",arch);
    fputs("Segunda linea\n",arch);
    fputs("Tercera linea\n",arch);
    fclose(arch);
    printf("Se creo un archivo de texto con tres lienas");
    getch();
    return 0;
}
```

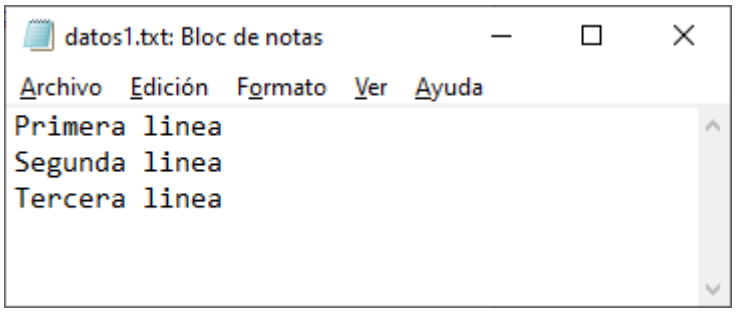
Si ejecutamos este será el resultado:



Lo abrimos con el mismo editor que utilizamos para programar:



Ahora con el bloc de notas:



Capítulo 222.- Archivo de texto: Lectura – 1

Vimos en el capítulo anterior como crear y grabar datos en un archivo de texto. La visualización de su contenido la hicimos desde un editor de texto.

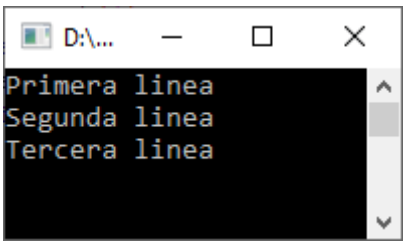
Ahora veremos como acceder al contenido del archivo desde nuestro programa.

Problema

Abrir el archivo "datos.txt" en modo de lectura, leer todas sus líneas y mostrar su contenido por pantalla.

```
#include<conio.h>
#include<stdlib.h>
#include<stdio.h>

int main()
{
    FILE *arch;
    arch=fopen("datos1.txt", "rt");
    if (arch==NULL)
        exit(1);
    char linea[100];
    fgets(linea, 100,arch);
    while (!feof(arch))
    {
        printf("%s", linea);
        fgets(linea,100, arch);
    }
    fclose(arch);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



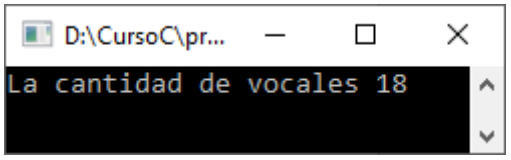
Capítulo 223.- Archivo de texto: Lectura – 2

Problemas propuestos

- Proceder a la apertura y lectura del archivo "datos1.txt". Imprimir la cantidad de vocales almacenadas en el mismo.

```c
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>

int main()
{
    FILE *arch;
    arch=fopen("datos1.txt", "rt");
    if (arch==NULL)
        exit(1);
    int cant=0;
    char caracter=fgetc(arch);
    while(!feof(arch))
    {
        switch(caracter){
        case 'a':;
        case 'e':;
        case 'i':;
        case 'o':;
        case 'u':;
        case 'A':;
        case 'E':;
        case 'I':;
        case 'O':;
        case 'U':cant++;
                 break;
        }
        caracter=fgetc(arch);
    }
    fclose(arch);
    printf("La cantidad de vocales %i", cant);
    getch();
    return 0;
}
```

Si ejecutamos este será el resultado:



- Proceder a la apertura y lectura del archivo "datos1.txt". Imprimir la cantidad de líneas que contiene.

```c
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>

int main()
{
    FILE *arch;
    arch=fopen("datos1.txt", "rt");
    if (arch==NULL)
        exit(1);
    int cant=0;
    char caracter=fgetc(arch);
    while(!feof(arch))
    {
        if(caracter=='\n')
            cant++;

        caracter=fgetc(arch);
    }
    fclose(arch);
    printf("La cantidad de lineas son %i", cant);
    getch();
    return 0;
}

```

Si ejecutamos este será el resultado: