Introducción a la codificación

Cada lenguaje dispone de una *sintaxis* o conjunto de reglas con las que se indica de modo inequívoco las operaciones que se quiere realizar.
Los *lenguajes de alto nivel* son más o menos comprensibles para el usuario, pero no para el procesador; entonces para que el procesador pueda ejecutarlos es necesario traducirlos *a su propio lenguaje de máquina*.
Esta, es una de las tareas que realiza el entorno de desarrollo (Borland C, Turbo C o DevC) que me permite escribir el programa fuente, traducir el programa a lenguaje de máquina y ejecutarlo.
La tarea de traducción se suele descomponer en dos etapas, que se pueden realizar juntas o por separado.
El programa de alto nivel se suele almacenar en un fichero llamado ***programa fuente***, que en casi todos los *sistemas operativos* se caracterizan por una ***extensión*** especial. Así, todos los programas fuente de C terminan en ***.cpp***
Por ejemplo:
calculos.cpp , derivada.cpp
En la primera etapa el compilador realiza una traducción directa del programa a un lenguaje más próximo al del computador (este proceso se llama *compilado*), esta traducción da como resultado, un ***programa objeto*** con el mismo nombre que el original, pero con la extensión ***.obj***.
En una segunda etapa se realiza el proceso de *montaje* (enlazado) del programa, consistente en producir un ***programa ejecutable*** en lenguaje máquina, en el que están ya incorporados todos los otros módulos que aporta el sistema sin intervención explícita del programador (funciones de librería, recursos del sistema operativo, etc.).
En un PC con sistema operativo ***Windows*** el programa ejecutable se guarda en un programa con extensión *.***exe***.
Este programa es cargado por el sistema operativo en la memoria RAM cuando va a ser ejecutado.

Una de las ventajas más importantes de los lenguajes de alto nivel es la *portabilidad* de los programas fuente resultantes. Esto quiere decir que un programa desarrollado en una PC podrá ser ejecutado en una Macintosh o en una máquina UNIX, con mínimas modificaciones y una simple recompilación. El lenguaje C, originalmente desarrollado por D. Ritchie en los laboratorios Bell de la AT&T, fue posteriormente estandarizado por un comité del ANSI (American National Standard Institute) con objeto de garantizar su portabilidad entre distintos computadores, dando lugar al ***ANSI C***, que es la variante que actualmente se utiliza casi universalmente.

Concepto de "Función"

Las aplicaciones informáticas que habitualmente se utilizan, incluso a nivel de informática personal, suelen contener decenas y aún cientos de miles de líneas de código fuente.
A medida que los programas se van desarrollando y aumentan de tamaño, se convertirían rápidamente en sistemas poco manejables si no fuera por la

modularización, que es el proceso consistente en dividir un programa muy grande en una serie de bloques mucho más pequeños y manejables.
A estos bloques se les ha denominado de distintas formas (*subprogramas*, *subrutinas*, *Procedimientos*, *funciones*, etc.).
En C se hace uso del concepto de ***función*** (***function***). Sea cual sea la forma de denominar, la idea siempre es la misma: dividir un programa grande en un conjunto de subprogramas o funciones más pequeñas que son llamadas por un programa o función principal y éstas a su vez pueden llamar a otras funciones en algunos casos.
Para iniciar este camino rumbo a aprender a codificar en C nuestros algoritmos de resolución de problemas (para poder ejecutarlos y ponerlos a prueba).En primer término veremos como se acordaron en ANSI-C, algunos elementos de uso común.
Veamos como le damos un nombre a cualquier variable o función, arreglo, puntero, estructura, etc. A este nombre se lo denomina **IDENTIFICADOR.**
En C, un identificador es una combinación de caracteres siendo el primero una letra del alfabeto o un símbolo de subrayado y el resto cualquier letra del alfabeto, cualquier dígito numérico ó símbolo de subrayado. Tres reglas debemos tener en cuenta cuando les demos nombre a los identificadores:

1. Elegir un identificador o nombre de el elemento que nos de información acerca de lo que va a contener o descriptivo acerca de la función que va a cumplir y que no sea una palabra reservada para el uso de el compilador, por ejemplo si el algoritmo calcula el volumen de cubos a la variable que contiene la arista, no la voy a llamar Pepe, ni A, un Identificador adecuado seria arista o lado esto nos va a ayudar a que sea mas legible un programa.
2. El Compilador es sensible a mayúsculas y minúsculas. Usar **PRINCIPAL** para el nombre de una variable no es lo mismo que usar **principal**, como tampoco es lo mismo que usar **PrInCiPaL**. Los tres serian variables diferentes.
3. De acuerdo al estándar ANSI-C, al darle nombre a un identificador solo serán significativos los primeros 31 caracteres, todo carácter mas allá de este límite será ignorado por cualquier compilador que cumpla la norma ANSI-C

Un elemento importante es el símbolo de subrayado que puede utilizarse como parte del nombre de una variable, contribuyendo notablemente a la legibilidad del código resultante.

PALABRAS CLAVE DEL C

En C, como en cualquier otro lenguaje, existen una serie de palabras clave (*keywords*) que el usuario no puede utilizar como identificadores (nombres de variables y/o de funciones). A continuación se presenta la lista de las 32 palabras clave del ANSI C,a lo largo del cursado conoceremos sus significados (algunos compiladores añaden otras palabras clave, propias de cada uno de ellos. Es importante evitar usarlas como identificadores):

auto double int struct
break else long switch
case enum register typedef

char extern return union
const float short unsigned
continue for signed void
default goto sizeof volatile
do if static while

La función principal main( )

Todo programa C, desde el más pequeño hasta el más complejo, tiene un *programa principal* que es con el que se comienza la ejecución del programa y no necesariamente debe ser lo primero que este escrito. Este programa principal es una función, pero una función que está por encima de todas las demás. Esta función se llama **main()** y tiene la forma siguiente (la palabra *void* es opcional en este caso):

```
void main(void)
{
sentencia_1
sentencia_2
…
}
```

Las *llaves* {...} constituyen el modo utilizado por el lenguaje C para agrupar varias sentencias de modo que se comporten como una sentencia única (*sentencia compuesta* o *bloque*). Todo el cuerpo de la función debe ir comprendido entre las llaves de apertura y cierre.
Para codificar nuestros diagramas de flujo debemos poder mostrar por pantalla y leer datos desde el teclado para ello el C hace uso de dos funciones que pertenecen a la librería Standard "stdio.h" y son printf() y scanf() Estas funciones tienen estructuras similares dentro del paréntesis tienen dos partes bien diferenciadas , la cadena de control que va entre comillas dobles y separada por comas y la lista de variables que se reemplazaran una a una y en el orden correspondiente en cada uno de los especificadores de tipos de datos , los que veremos inicialmente son :

%d para enteros tipo int en C
%c para caracteres tipo char en C
%f para numeros racionales float en C

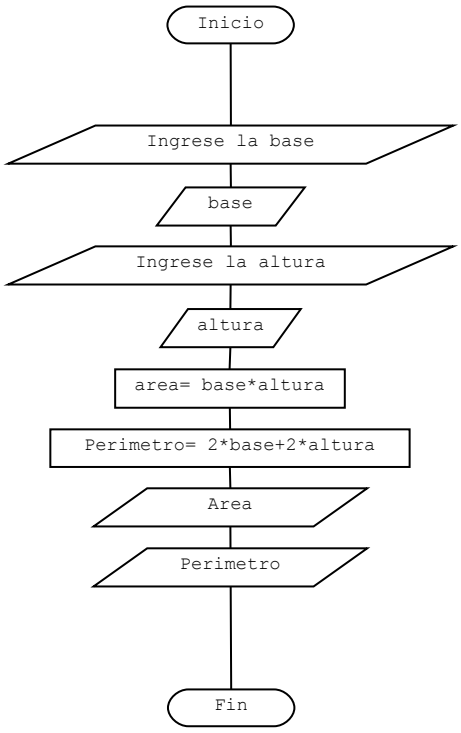
En el caso de printf despliega el texto en el monitor, que esta entre comillas y dentro del paréntesis que sigue a la palabra printf y el contenido de las variables se reemplazan de acuerdo a los especificadores de impresión y en el orden correspondiente dentro de la cadena de control.
En el caso de la función de entrada scanf la lista de variables debe contener el operador monario & antes de cada identificador ya que esta función necesita como argumento la dirección de memoria de cada variable.

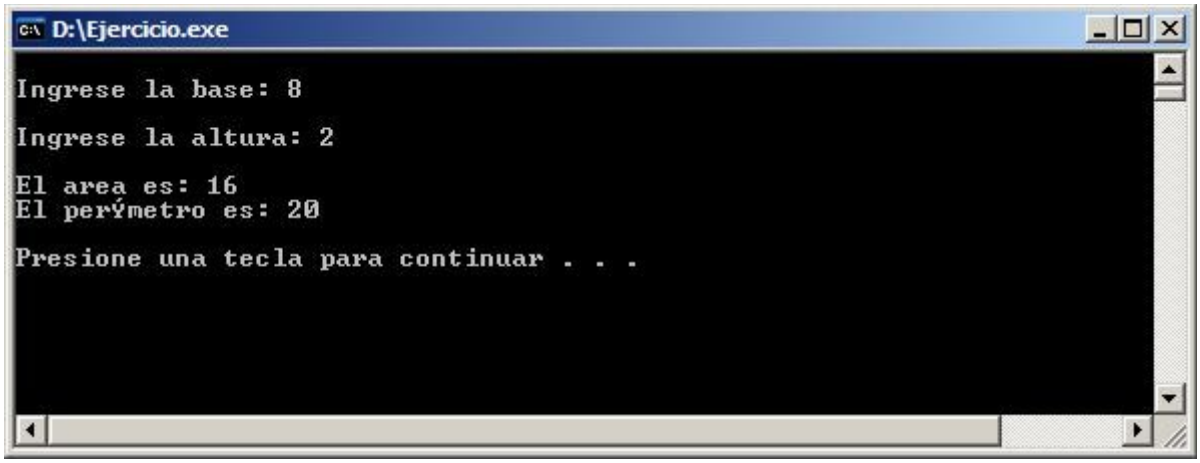
Observe que al final del enunciado se ha puesto el símbolo de punto y coma. C utiliza el punto y coma para indicarle al compilador que una línea ejecutable está completa.

Veamos ahora un ejemplo de aplicación interesante con una estructura secuencial:

EJEMPLO 1: Ingresar los valores de los lados de un rectángulo y calcular su área y su perímetro.

```
#include <stdio.h>

main()
{
int base, altura, area, perimetro;

printf("\nIngrese la base: ");

scanf("%d", &base);

printf("\nIngrese la altura: ");

scanf("%d", &altura);

area=base*altura;

perimetro=2*base+2*altura;

printf("\nEl area es: %d", area);

printf("\nEl perimetro es: %d", perimetro);

printf("\n\n");

system("PAUSE");
return 0;
}
```





Este código incluye enunciados ejecutables además del enunciado obligatorio **return**. El ejecutable contiene llamadas a funciones incluidas como parte de su librería C, la funciones **printf () y scanf()** están definidas en el archivo de cabecera stdio.h. , son accesibles mediante la primera instrucción

la línea superior será ejecutada en primer lugar seguidas de las otras líneas ejecutables en el orden en que aparecen, note el carácter dentro de la cadena de control de printf la primera línea ejecutable, la diagonal invertida (\ ) es utilizada para indicar que sigue un carácter especial de control llamado secuencia de escape . En este caso, la "n" indica la petición de una nueva línea de texto, es una indicación para regresar el cursor al lado izquierdo del monitor y a la vez moverlo una línea abajo.

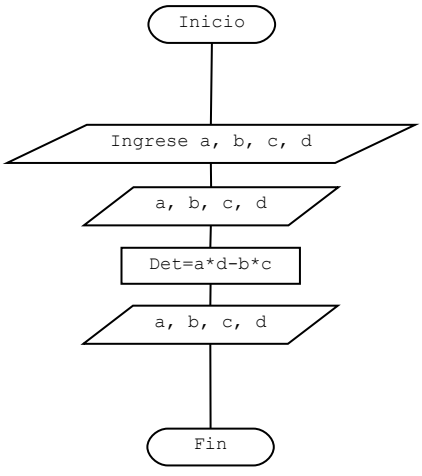
En cuanto a las operaciones, utilizaremos los operadores aritméticos y los relacionales o lógicos ya vistos en la unidad anterior ya que son los mismos que usamos en el diagrama y el pseudocódigo.

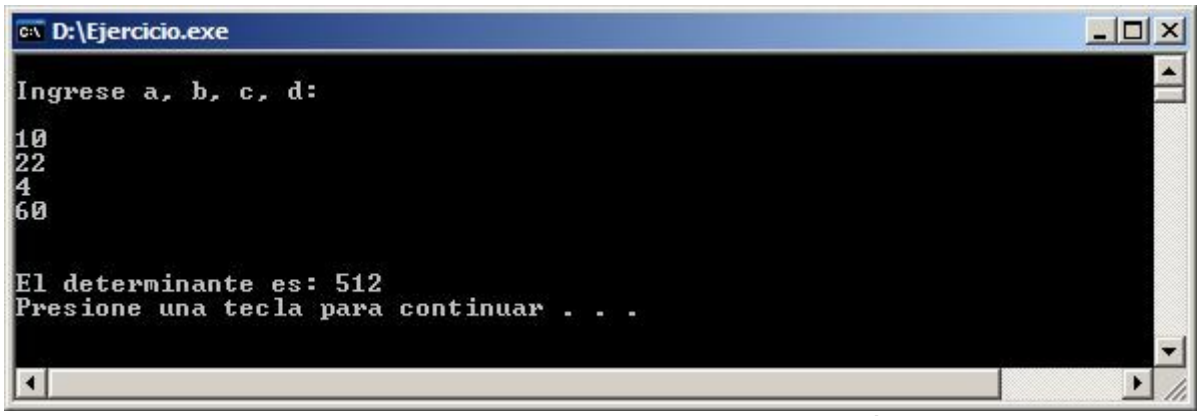
Aquí va otro ejemplo:

EJEMPLO 2: Ingresar los valores de los 4 elementos de una matriz, y calcular su determinante.

$$M = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \therefore \qquad \text{DET } M = a \cdot d - c \cdot b$$

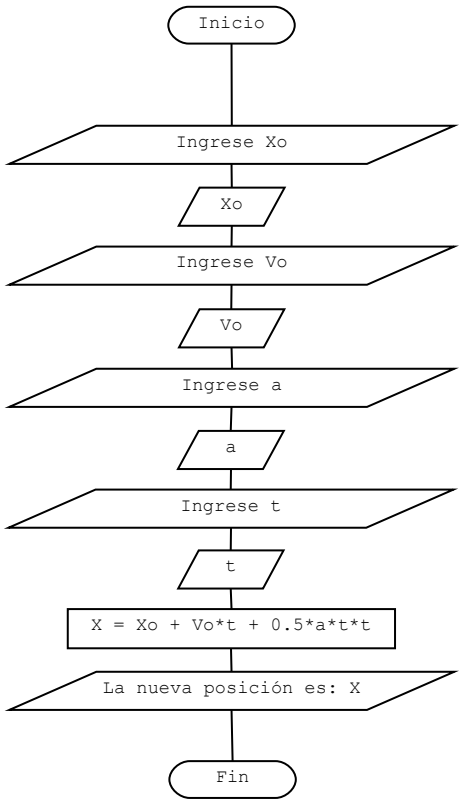
```
#include <stdio.h>

main()
{
int a, b, c, d, det;

printf("\nIngrese a, b, c, d:\n\n");

scanf("%d%d%d%d", &a,&b,&c,&d);

det=a*d – b*c;

printf("\n\nEl determinante es: %d", det);

system("PAUSE");
return 0;
}
```

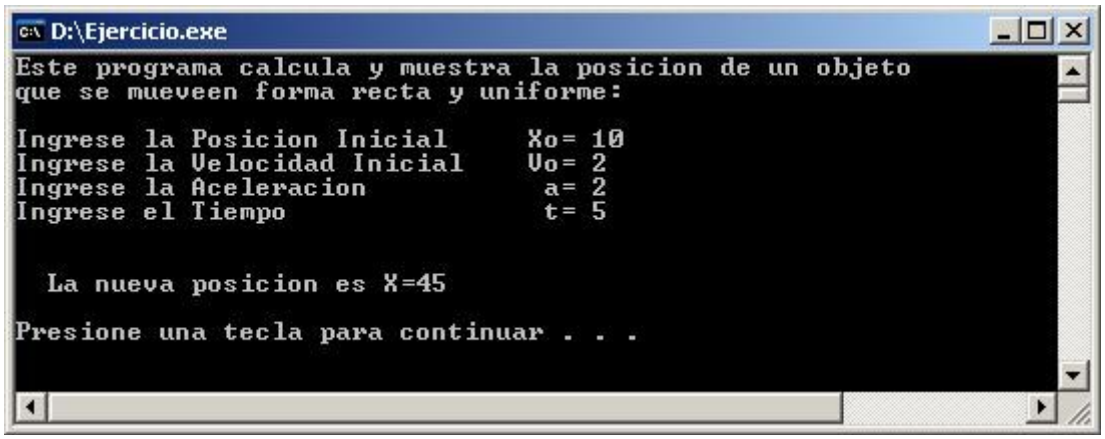




EJEMPLO 3: Diseñe un programa que permita calcular la posición de un objeto que se mueve en forma recta y uniforme, MRU (Movimiento Rectilíneo Uniforme) donde $X(t)=Xo+Vo*t+1/2*a*t^2$.

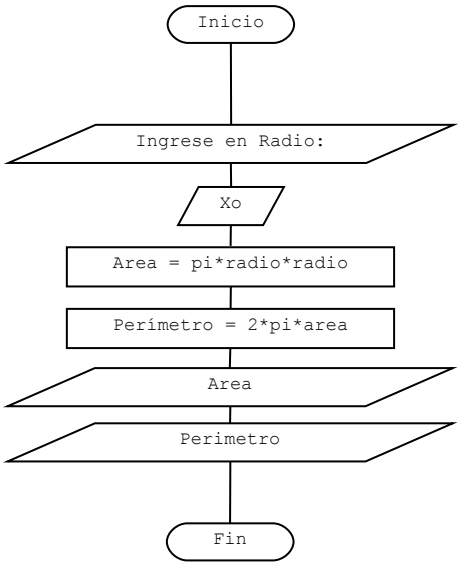
```
#include <stdio.h>

main()
{
int X, Xo, Vo, a, t;

printf("Ingrese la Posición Inicial \tXo= ");

scanf("%d", &Xo);

printf("Ingrese la Velocidad Inicial \tVo= ");

scanf("%d", &Vo);

printf("Ingrese la Aceleración \t a= ");

scanf("%d", &a);

printf("Ingrese en tiempo  \t\t T= ");

scanf("%d", &t);

X = Xo + Vo*t + 0.5*a*t*t;

printf("\n\n  La nueva posicion es X=%d\n\n", X);

system("PAUSE")
return 0;
}
```

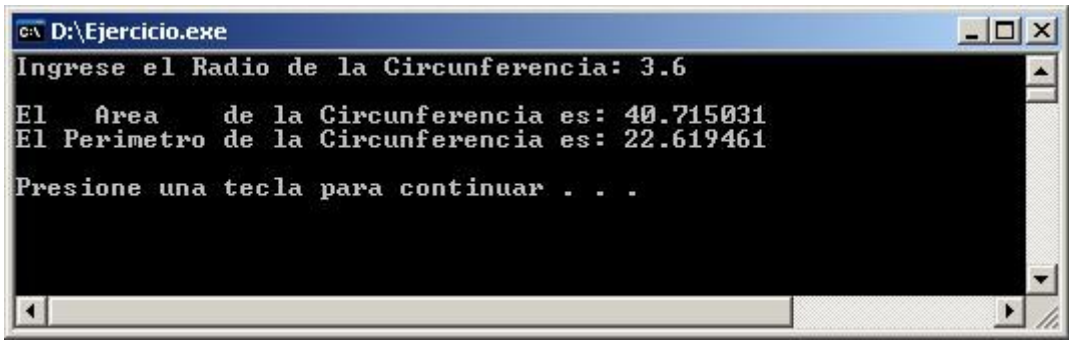



D:\Ejercicio.exe
Este programa calcula y muestra la posicion de un objeto
que se mueveen forma recta y uniforme:
Ingrese la Posicion Inicial Xo= 10
Ingrese la Velocidad Inicial Vo= 2
Ingrese la Aceleracion a= 2
Ingrese el Tiempo t= 5
La nueva posicion es X=45
Presione una tecla para continuar . . .

EJEMPLO 4: Diseñe un programa que permita calcular el Área y el Perímetro de una circunferencia, ingresando como dato el Radio de la misma.

```
#include <stdio.h>

int main()
{
float pi=3.141592, radio, area, per;

printf ("Ingrese el Radio de la Circunferencia: ");

scanf ("%f", &radio);

area = pi*radio*radio;

per = 2*pi*radio;

printf ("\nEl   Area    de la Circ. es: %f", area);

printf ("\nEl Perimetro de la Circ. es: %f\n\n", per);

system("PAUSE");
return 0;
}
```





Estructura Selectiva Simple y Doble

Estructura Selectiva Simple "if"

Es una estructura de control de flujo condicional, cuya sintaxis en C es la siguiente:

```
if (Condición)
    Acción;
```

En el caso de un cuerpo compuesto seria:

```
if (Condición)
    {
    Acción1;
    Acción2;
    ...
    }
```

La acción o acciones se ejecutan sólo si la condición que se evalúa es distinta de 0, o sea, si es verdadera.

Estructura Selectiva Doble "if-else"

Es una estructura de control de flujo condicional, cuya sintaxis en C es la siguiente:

```
if (Condición)
    Acción_i;
else
    Acción_e;
```

En el caso de un cuerpo compuesto seria:

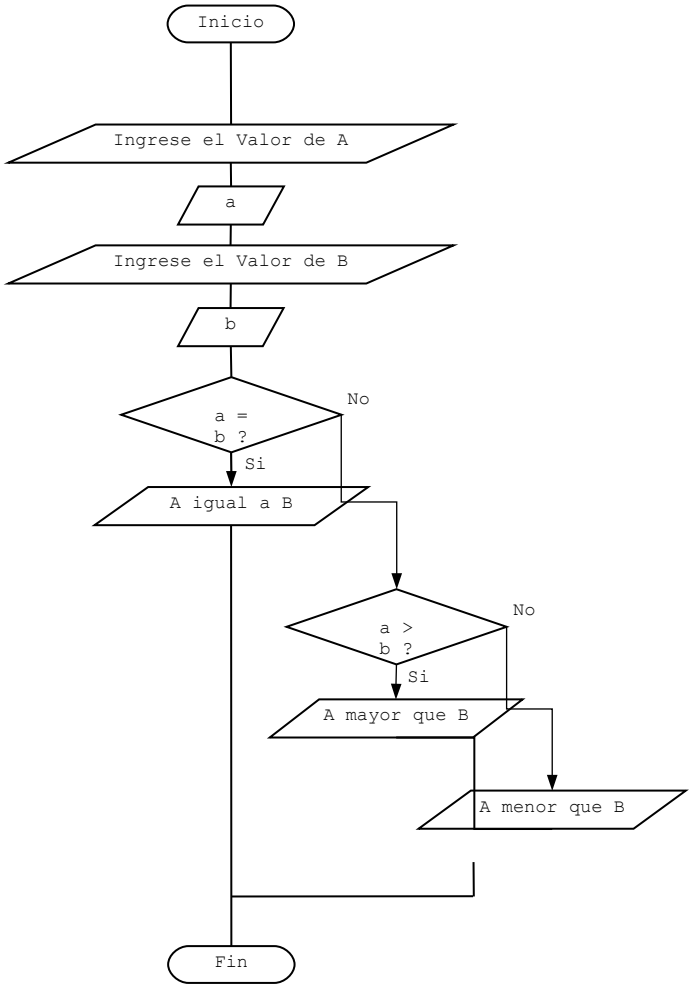
```
if (Condición)
    {
    Acción_i1;
    Acción_i2;
    ...
    }
else
    {
    Acción_e1;
    Acción_e2;
    ...
    }
```

La acción_i o acciones_i se ejecutan sólo si la condición que se evalúa es distinta de 0, o sea, verdadera.
La acción_e o acciones_e se ejecutan sólo si la condición que se evalúa es igual a 0, o sea, falsa.

En este caso, la solución del problema conduce a que, según se cumpla cierta condición o no, se ejecute una u otra de las acciones diferentes.

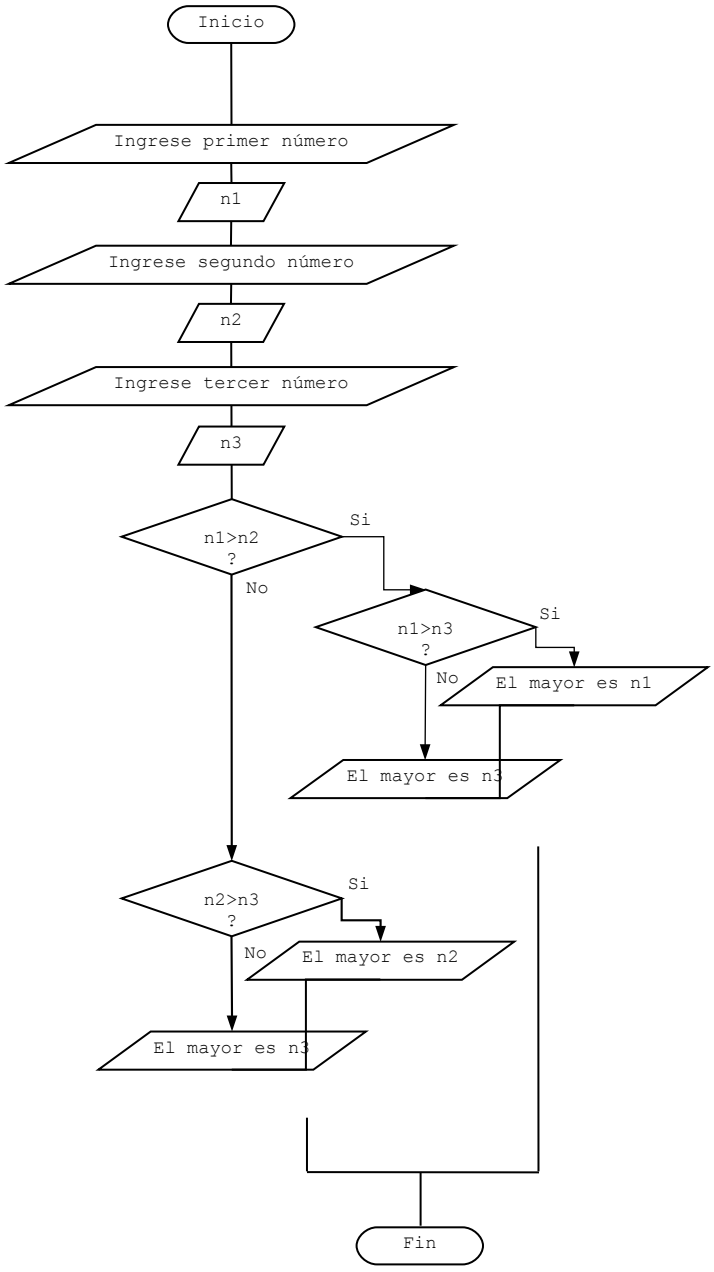
EJEMPLO 1: Ingresar dos números, por ejemplo **'a'** y **'b'**, e indicar si **a<b**, si **a=b** o si **a>b**.

```
#include <stdio.h>

main()
{
int a, b;

printf("Ingrese el valor de A: ");

scanf("%d", &a);

printf("Ingrese el valor de B: ");

scanf("%d", &b);


if (a==b)

        printf ("A es igual a B");

else

        if (a>b)

                printf ("A es mayor que B");

        else

                printf ("A es menor que B");

printf ("\n\n");

system("PAUSE");
return 0;
}
```



EJEMPLO 2: Ingresar tres números, e imprimir el mayor de ellos.

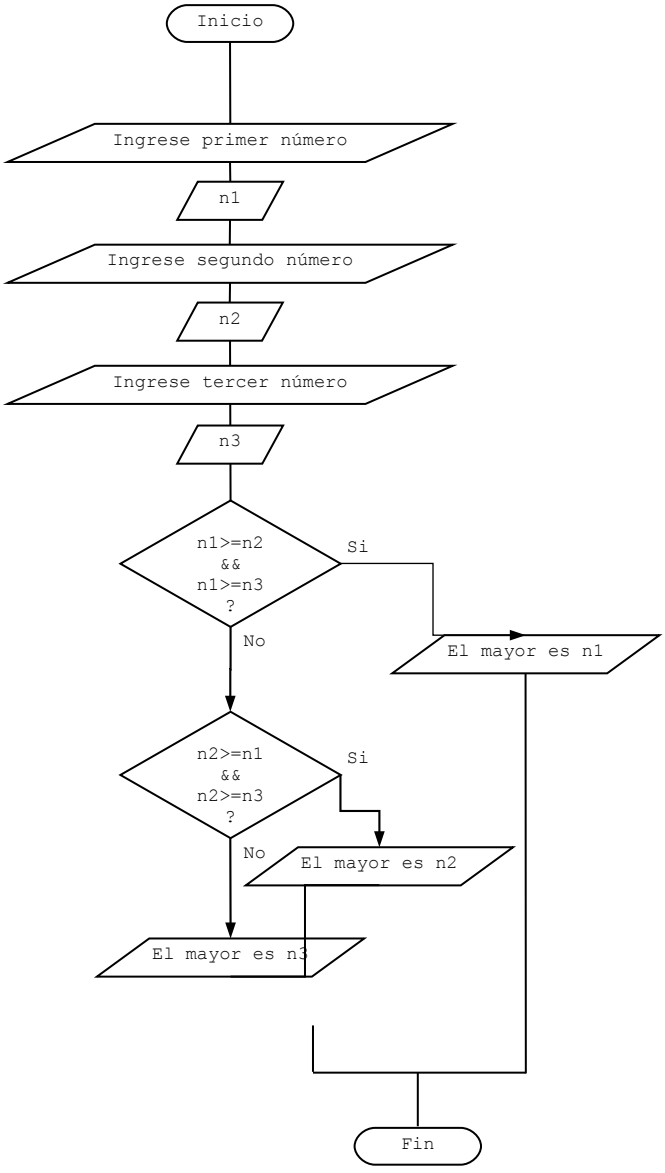
```
#include <stdio.h>

main()
{
int n1 ,n2, n3;

printf("\nIngrese el primer número: ");

scanf("%d", &n1);

printf("\nIngrese el segundo número: ");

scanf("%d", &n2);

printf("\nIngrese el tercer número: ");

scanf("%d", &n3);


if (n1>n2)

   {
   if (n1>n3)

      printf("\nEl mayor es: %d", n1);

   else
      printf("\nEl mayor es: %d", n3);
   }
else
   {
   if(n2>n3)

      printf("\nEl mayor es: %d", n2);

   else
      printf("\nEl mayor es: %d", n3);
   }

printf ("\n\n");
```



```
system("PAUSE");
return 0;
}
```

El ejemplo 2 también puede ser realizado de la siguiente manera (IF de doble condición):

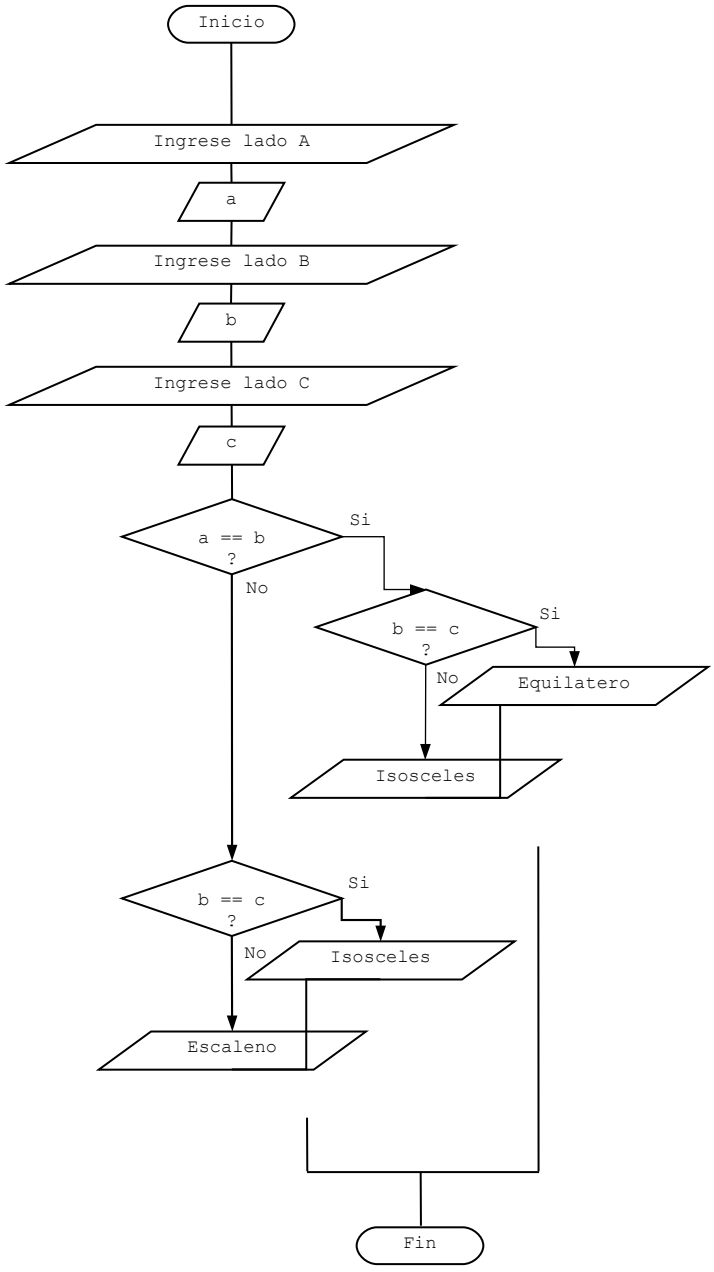
```
#include <stdio.h>

main()
{
int n1 ,n2, n3;

printf("\nIngrese el primer número: ");

scanf("%d", &n1);

printf("\nIngrese el segundo número: ");

scanf("%d", &n2);

printf("\nIngrese el tercer número: ");

scanf("%d", &n3);


   if (n1>=n2 && n1>=n3)


      printf("\nEl mayor es: %d", n1);

   else
      {
      if (n2>=n1 && n2>=n3)


         printf("\nEl mayor es: %d", n2);
      else

         printf("\nEl mayor es: %d", n3);
      }

printf ("\n\n");
```



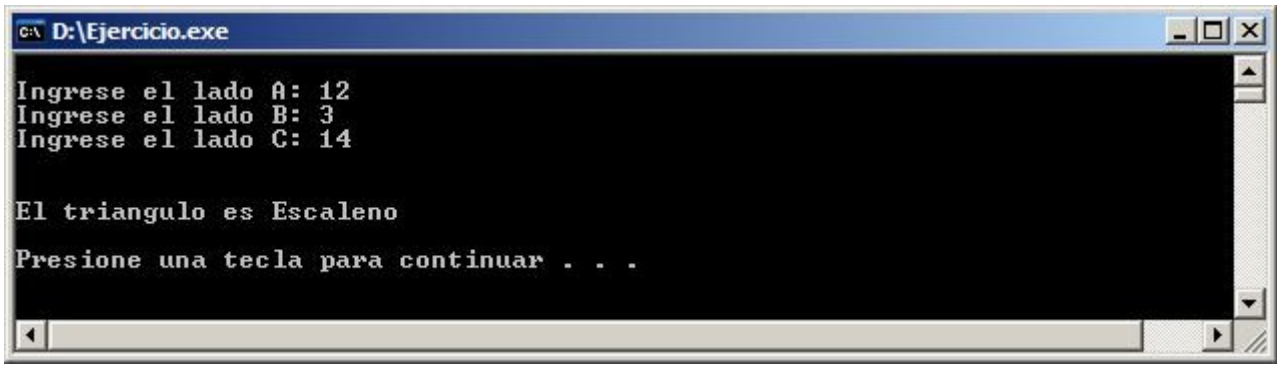
```
system("PAUSE");
return 0;
}
```

EJEMPLO 3: Ingresar el valor de cada lado de un triangulo, e indicar si el triangulo es equilátero, isósceles o escaleno.

```
#include <stdio.h>

main()
{
int a, b, c;

printf("\nIngrese el lado A: ");

scanf("%d", &a);

printf("\nIngrese el lado B: ");

scanf("%d", &b);

printf("\nIngrese el lado C: ");

scanf("%d", &c);


if(a==b)
   {

   if(b==c)

      printf("\n\nEl triangulo es Equilatero");

   else
      printf("\n\nEl triangulo es Isosceles");
   }
else
   {
   if(b==c)

      printf("\n\nEl triangulo es Isosceles");
```



```
   else
      printf("\n\nEl triangulo es Escaleno");
   }

printf ("\n\n");

system("PAUSE");
return 0;
}
```



Trabajo Práctico Número 3

Actividad 1:
Correr en C todos los ejemplos de estructura secuencial y selectiva en el Laboratorio de informática modificar las salidas e implementar mejoras si es posible.
Actividad 2: Codificar en C todos los diagramas de flujo realizados en clase práctica y teórica que solo usen estructuras secuenciales.
Actividad 3: Diseñar un programa que determine el mayor y el menor de cuatro números utilizando una variable mayor y menor para almacenarlos.
Diagrama de flujo y pseudocódigo
Actividad 4: Codificar en C el ejercicio anterior en el Laboratorio de informática
Actividad 5: Diseñar un programa que muestre un menú donde se puedan seleccionar estas distintas opciones y si ingresa otra opción que no sea alguna de estas mostrar un cartel de error:
1-sumar
2-multiplicar
3-restar
4-dividir
Ejecutar la operación de acuerdo a la opción elegida entre dos números ingresados por teclado utilizando funciones de entrada y salida de manera adecuada
Diagrama de flujo y pseudocódigo

Actividad 6:
Codificar en C el ejercicio anterior en el Laboratorio de informática.