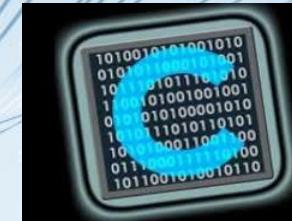


Lenguaje De Programación C



Introducción

- El lenguaje de programación en C, es un lenguaje conocido como de alto nivel.
- Es un lenguaje estructurado, lo que permite generar código claro y sencillo, ya que esta basado en la modularidad.
- C ha tenido un gran impacto en el diseño de otros muchos lenguajes. Ha sido, por ejemplo, la base para definir la sintaxis y aspectos de la semántica de lenguajes tan populares como Java y C++.

C es un lenguaje compilado: antes de ejecutar un programa escrito por nosotros, suministramos su código fuente a un compilador de C. También hay Intérpretes en C.

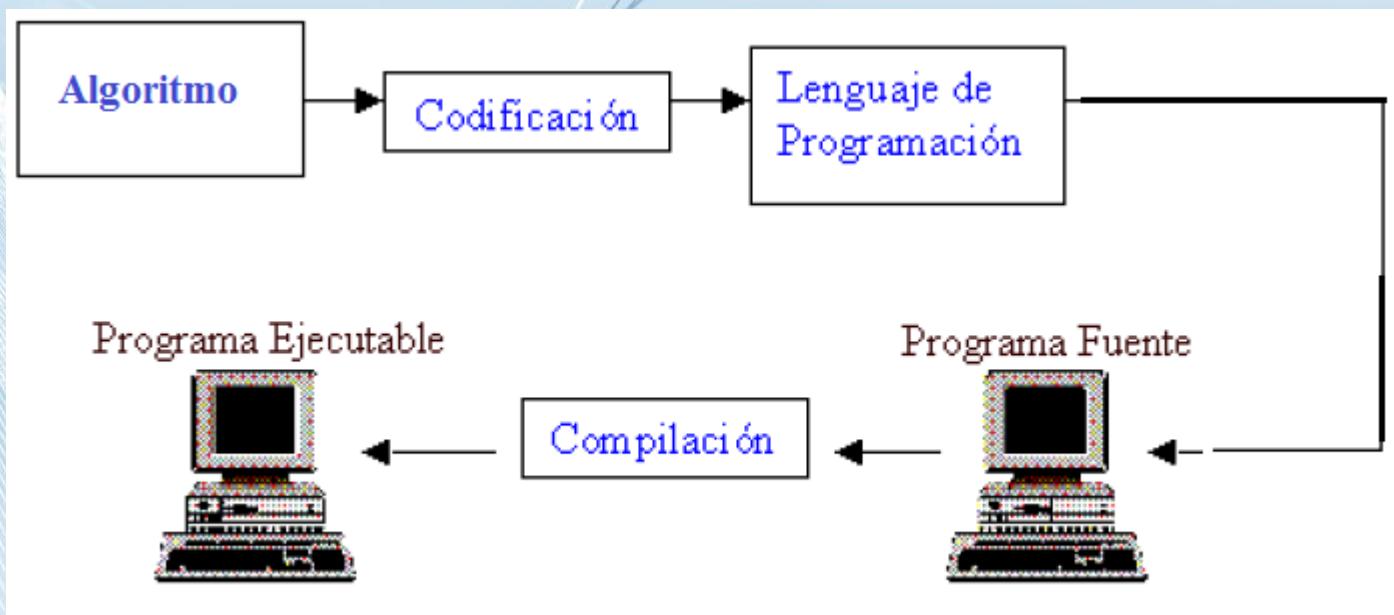
El compilador lee y analiza todo el programa. Si el programa está correctamente escrito según la definición del lenguaje, el compilador genera un nuevo fichero con su traducción a código de máquina, y si no, muestra los errores que ha detectado.

Para ejecutar el programa utilizamos el nombre del fichero generado. Si no modificamos el código fuente, no hace falta que lo compilemos nuevamente para volver a ejecutar el programa: basta con volver a ejecutar el fichero generado por el compilador

Nombre	Fecha de modifica...	Tipo	Tamaño
EjerF2	09/03/2017 8:57	CPP - Fuente C++	1 KB
EjerF2	09/03/2017 8:57	Aplicación	49 KB



- **Programa o código fuente:** Programa escrito en algún lenguaje y que no ha sido traducido a lenguaje de máquina.
- **Programa o código objeto:** Programa que ya se encuentra en lenguaje de máquina.
- **Compilador:** Traduce todo el programa y genera un código listo para funcionar



Enlace a la página para descargar el programa C

<http://zinjai.sourceforge.net/>





Palabras Reservadas

Son aquel grupo de identificadores (palabras) que no pueden ser utilizadas por el usuario para nombrar a las variables, funciones, procedimientos, objetos y demás elementos de programación que cree.

Ejemplos de palabras reservadas en C:

If – char – int –long –void -- printf

- Una variable o una constante no pueden llamarse igual que una palabra reservada
- Es preciso insistir en que C hace distinción entre mayúsculas y minúsculas. Por lo tanto, la palabra reservada for no puede escribirse como FOR, pues el compilador no la reconoce como una instrucción, sino que la interpreta como un nombre de variable.



Identificadores

Un identificador es un conjunto de caracteres alfanuméricos que sirve para identificar las entidades del programa (clases, funciones, variables, etc.)

Los identificadores pueden ser combinaciones de letras y números.

Cada lenguaje tiene sus propias reglas que definen como pueden estar construidos

Ejemplos de identificadores válidos son:

Precio_Venta

Num1

_123

D_i_5

No son válidos:

Precio Venta Lleva un espacio en blanco

1Num Empieza por un número

Precio-Venta Lleva un guión



Identificadores

En C, se debe tener en cuenta que todo identificador debe cumplir las siguientes reglas de sintaxis:

- **Consta de uno o más caracteres.**
- El primer carácter debe ser una letra o el carácter *subrayado* (_), mientras que, todos los demás pueden ser letras, dígitos o el carácter *subrayado* (_). Las *letras* pueden ser minúsculas o mayúsculas del alfabeto inglés. Así pues, no está permitido el uso de las letras 'ñ' y 'Ñ'.
- No pueden existir dos identificadores iguales, es decir, dos elementos de un programa no pueden nombrarse de la misma forma. Lo cual no quiere decir que un identificador no pueda aparecer más de una vez en un programa.
- No debe haber un espacio en blanco entre las letras o palabras que conforman el identificador.
- De un identificador sólo son significativos los 32 primeros caracteres.



Tipos de Datos

Tipo	Descripción	Cantidad de memoria requerida	Rango
<code>void</code>	Define vacío o valor <code>NULL</code> .		
<code>char</code>	Almacena un carácter. Puede almacenar un valor con signo.	1 byte	-128 a 127
<code>unsigned char</code>	Almacena un carácter o un valor sin signo.	1 byte	0 a 255
<code>int</code>	Define un valor numérico entero (sin fracción).	2 bytes	-32767 a 32768
<code>unsigned int</code>	Valor entero sin signo.	2 bytes	0 a 65,535
<code>short int</code>	Entero corto, puede ser igual al <code>int</code> o a la mitad.	2 bytes	-32767 a 32768
<code>float</code>	En punto flotante (puede ser una fracción o un entero con exponente).	4 bytes	3.4×10^{-38} a $3.4 \times 10^{+38}$
<code>double</code>	En punto flotante del doble de tamaño del <code>float</code> (más cifras significativas para la fracción o mayor para el exponente).	8 bytes	1.7×10^{-308} a $1.7 \times 10^{+308}$
<code>long</code>	Define un entero con signo, usualmente del doble de tamaño al <code>int</code> .	4 bytes	-2,147,483,648 a 2,147,483,647
<code>unsigned long</code>	Entero sin signo.	4 bytes	0 a 4,294,967,295
<code>long double</code>	Incrementa el tamaño del doblé.	10 bytes	3.4×10^{-4932} a $1.1 \times 10^{+4932}$



Constantes

Las constantes son valores que no pueden cambiar en la ejecución del programa.

Las constantes de carácter van encerradas en comillas simples. Las constantes enteras se especifican con números sin parte decimal y las de coma flotante con su parte entera separada por un punto de su parte decimal.



Constantes Simbólicas

Es un nombre que sustituye una secuencia de caracteres.

- Permite que aparezca un nombre en lugar de una constante numérica, una constante de carácter o una constante de cadena de caracteres.
- Cuando se compila un programa, cada aparición de una constante simbólica es reemplazada por su correspondiente secuencia de caracteres.
- Las constantes simbólicas se suelen definir al comienzo del programa.
- Se define una constante simbólica mediante la directiva **DEFINE**.



Constantes Simbólicas

Se define una constante simbólica escribiendo

#define nombre texto



Representa un nombre simbólico, que se suele escribir en letras mayúsculas

Representa la secuencia de caracteres asociada al nombre simbólico.

Observación:

texto no acaba con un punto y coma, ya que la definición de una constante simbólica no es una verdadera instrucción de C. Es más, si texto acabase con un punto y coma, este punto y coma se trataría como si fuese parte de la constante numérica, la constante de carácter o la constante de cadena de caracteres que se sustituye por el nombre simbólico.



Constantes Simbólicas

Cuando se utiliza `#define` para crear constantes, en realidad los valores no ocupan un espacio en la memoria como en el caso de las variables, sino que el compilador sustituye cada ocurrencia del nombre simbólico por su respectivo valor, antes de analizar sintácticamente el programa fuente.

Dicho de otro modo, cada vez que alguna instrucción utilice el identificador NP, éste será sustituido por el 1506 con que fue definido.



EJEMPLO:

```
#include <stdio.h>
#include <conio.h>
#define PI 3.141592654
#define TEXTO "Esto es un prueba"
```

```
int main ()
{
    printf ("El valor de pi es %f", PI);
    printf ("\n%s", TEXTO);
    printf ("Ejemplo de uso de DEFINE");
    getch ();
    return 0;
}
```



Constante Numérica: const

La sintaxis para declarar una constante **const** es:

```
<const> <tipo de dato> <identificador> = <valor>;
```

Ejemplos:

```
const int NP=1506;  
const float PI=3.1416;  
const char CAR='a';  
const char SALUDO[ ]= "Hola a todos";
```

Las declaraciones anteriores podrían utilizarse en lugar del **#define**, la diferencia consiste en que **const** tiene un espacio de memoria reservado para cada dato; además de su sintaxis, observe que **const** sí especifica el tipo de dato de cada valor; se utiliza el operador = para asignar dicho valor y además cada sentencia (o instrucción) termina con ;.



Variables

Las variables son valores que se pueden modificar durante la ejecución de un programa.

Declaración

Una declaración asocia un tipo de datos especificado a un grupo de variable.

- Se deben declarar todas las variables antes de que aparezcan en las instrucciones ejecutables.
- Una declaración consta de un tipo de datos, seguido de uno o más nombres de variables, finalizando con un punto y coma.



Ejemplo de Declaración

```
int a, b, c;  
float raiz1, raiz2;  
char indicador, texto[80];
```

De esta forma se declaran `a`, `b` y `c` como variables enteras, `raiz1` y `raiz2` son variables en coma flotante, `indicador` una variable de tipo carácter y `texto` un array de tipo carácter de 80 elementos. Observe los corchetes que delimitan la especificación de tamaño de `texto`.

También se podrían haber escrito las declaraciones anteriores como sigue:

```
int a;  
int b;  
int c;  
float raiz1;  
float raiz2;  
char indicador;  
char texto[80];
```



Ejemplo de Declaración

Ejemplos de declaración de variables:

`int a;` *Se reserva un espacio en la memoria llamado "a", con capacidad para un entero.*

`float b,c,d;` *Se reservan 3 espacios en la memoria para guardar 3 números reales, a los cuales se hace referencia mediante "b", "c" y "d" respectivamente.*

`char j;` *Se reserva un espacio en la memoria para poder almacenar cualquier carácter y se puede hacer referencia a este espacio mediante el identificador "j".*



Comentario

Son cadenas de caracteres o texto que describen partes del programa que el programador desea explicar.

Dicho texto no forma parte del programa fuente sino una descripción del mismo.

Para poder usar comentarios en un programa C y que el compilador no lo considere como instrucciones del programa se utilizan los símbolos para abrir el texto /* y para cerrar el comentario */.

También se puede usar //

Ejemplo:

/* este es un programa */

//este es un programa

Entrada y Salida de Datos



Funciones de Entrada y Salida

- **getchar**
 - **putchar**
 - **scanf**
 - **printf**
 - **gets**
 - **puts**.
-
- Estas seis funciones permiten la transferencia de información entre la computadora y los dispositivos de entrada/ salida estándar (por ejemplo, un teclado y un monitor).
 - **getchar** y **putchar**, permiten la transferencia de caracteres individuales hacia dentro y hacia fuera de la computadora;
 - **scanf** y **printf** son más complicadas, pero permiten la transferencia de caracteres individuales, valores numéricos y cadenas de caracteres;
 - **gets** y **puts** permiten la entrada y salida de cadenas de caracteres.



Función printf

Es la función de salida de datos con formato .

La sintaxis es :

printf (texto,cadena de control, lista de argumentos);

donde texto y cadena de control son opcionales, dependiendo de lo que se desee mostrar. Cadena de control es una cadena de caracteres “%tipo” que indica el tipo de dato a desplegar (lo requiere la función *printf()*). Por otro lado, argumento o argumentos es el valor o los valores que se pretende mostrar, y pueden ser variables, constantes, expresiones aritméticas, resultados de funciones o simplemente texto que el programa debe mostrar al usuario. A continuación se muestra un ejemplo del uso y la explicación.

```
int a;  
a= 2;  
...  
printf ("el numero es :%d", a);
```



Función printf()-Ejemplos

```
int a=7;  
float b=8.2;  
char c='s';
```

•

`printf("%d", a);`

Se visualiza un 7, que es el contenido de la variable a.

`printf("%d", a+b);`

Se visualiza un 15, ya que es la suma de a + b mostrada como valor entero.

`printf("%f", a+b);`

Se visualiza un 15.2, ya que es la suma de a + b mostrada como valor real.

`printf("%c", c);`

Se visualiza la letra 's' que es el contenido de la variable c.

`printf("%c %d %f", c, a,b);`

Se visualiza s 7 8.2 que son los valores de las variables c, a y b respectivamente.

`printf("HOLA");`

Se visualiza la palabra HOLA.



Función printf() - Cadena de Control

Cadena de tipo	Descripción
%d	<i>El dato es un entero decimal (int).</i>
%i	<i>El dato es un entero.</i>
%o	<i>El dato es un entero octal.</i>
%x	<i>El dato es un entero hexadecimal.</i>
%u	<i>El dato es un entero sin signo en decimal (unsigned int).</i>
%c	<i>El dato es un carácter (char).</i>
%e	<i>El dato es un real expresado en base y exponente (float).</i>
%f	<i>El dato es un real escrito con punto decimal con signo (float).</i>
%g	<i>El dato es un real (float).</i>
%s	<i>El dato es una cadena de caracteres que finaliza con el carácter nulo \0.</i>
%lf	<i>El dato es real de tipo long double.</i>



Secuencia de Escape

Las secuencias de escape son cadenas de caracteres que tienen distintos significados dependiendo de la cadena que se utilice.

La forma más sencilla de escribir una secuencia de escape es con el carácter *barra invertida* (\), seguido de un carácter especial. Por tanto, cuando en la cadena de control de la función printf se escriba una secuencia de escape, o bien se mostrará un carácter gráfico por pantalla, o bien se realizará una acción.



Secuencia de Escape

Secuencia de escape	Descripción
\a	Alarma
\b	Retroceso
\f	Avance de página
\n	Retorno de carro y avance de línea
\r	Retorno de carro
\t	Tabulación
\v	Tabulación vertical
\	Diagonal invertida
\?	Signo de interrogación
\"	Comillas dobles
\000	Octal
\xhh	Hexadecimal
\0	Carácter nulo



Secuencia de Escape- Ejemplo

```
int a=7;  
float b=8.2;  
char c='s';
```

```
printf("%d \n \t %f \n \t\t %c", a,b,c);
```

7

8.2

s

Estructura de un Programa en C

printf ()



Formateadores: Permite dar formato específico a la salida.

Formateador	Salida
%d ó %i	entero en base 10 con signo (int)
%u	entero en base 10 sin signo (int)
%o	entero en base 8 sin signo (int)
%x	entero en base 16, letras en minúscula (int)
%X	entero en base 16, letras en mayúscula (int)
%f	Coma flotante decimal de precisión simple (float)
%lf	Coma flotante decimal de precisión doble (double)
%e	La notación científica (mantisa / exponente), minúsculas (decimal precisión simple ó doble)
%E	La notación científica (mantisa / exponente), mayúsculas (decimal precisión simple ó doble)
%c	caracter (char)
%s	cadena de caracteres (string)

Ejemplo del uso de Formateadores:

%.7i	largo mínimo de 7 dígitos, justificado a la derecha, rellena con ceros
%8.2f	tamaño total de 8 dígitos, con dos decimales



Función scanf()

Es la función de entrada de datos con formato .

La sintaxis es :

scanf (cadena de control, &variable);

El operador & es necesario en scanf() para simular las *llamadas por referencia*, y hace que la función trabaje internamente con la dirección de la variable.

```
int edad;  
float est;  
printf("Teclea tu edad");  
scanf("%d",&edad);
```



Ejemplo de Funciones de Entrada y Salida

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

```
int main ()
```

```
{
```

```
    float x;
```

```
    int k;
```

```
    printf("Ingrese un valor flotante:");
```

```
    scanf("%f",&x);/* entrada de número en coma flotante */
```

```
    printf("Ingrese un valor entero:");
```

```
    scanf("%d\n",&k);/* entrada de enteros */
```

```
    printf("Mostrar los valores ingresados:");
```

```
    printf("Valor flotante: %f“ y valor entero:%d”, x,k);
```

```
}
```



Los parámetros formales de una función son variables locales que se crean al comenzar la función y se destruyen cuando termina. El tipo de dato de cada parámetro formal debe ser el mismo que el tipo de los argumentos se utilicen al llamar a la función. Este error no se detecta en la compilación y para remediarlo, se deben usar los prototipos de funciones.

Los parámetros de una función pueden ser :

- valores (llamada por valor)
- direcciones (llamada por referencia)

Llamada por valor

Cuando un argumento se pasa por valor, la función llamada recibe una copia del argumento, pudiendo modificarla libremente sin alterar el valor original.



Cualquier modificación sobre estos parámetros no afecta a las variables que se utilizan para llamar a la función, puesto que el parámetro en estos casos es una copia de la variable.

Solo se pueden pasar por valor los tipos atómicos, es decir no son arrays ni estructuras.

```
int suma(int, int);

main()
{
    float x = 1.0, y = 2.0, z;
    .....
    z = suma(x, y);
    .....
}
float suma(float x, float y)
{
    return(x + y);
}
```

Paso de argumentos por valor.

1. Dentro de la función suma(), las variables x e y son copias de las variables x e y de main().



Llamada por referencia

Cuando un argumento es una dirección, el parámetro recibe la dirección de la variable que se ha pasado como argumento al invocar a la función.

Por lo tanto, el parámetro deberá declararse como un puntero y de esta forma se puede modificar el contenido de las variables.

Si una función tiene que devolver más de un valor, lo hará utilizando sus parámetros y necesariamente los argumentos deben pasarse por referencia.



Cuando un argumento se pasa por referencia, más que el valor mismo del argumento, lo que se pasa es la dirección de memoria donde ese valor está almacenado. Por lo tanto, la función llamada puede modificar el valor original directamente.

```
main()
{
int x = 10, y = 20;

alternar(&x, &y);
printf("x = %d, y = %d", x,y);
}

void alternar(int *px, int *py)
{
int temp;

temp = *px;
*px = *py;
*py = temp;
}
```

Paso de argumentos por referencia.

1. Los valores de x e y han sido intercambiados.



Ejemplo de Funciones de Entrada y Salida

```
#include <stdio.h>
int main ()
{
    char c;
    float x, y;
    int i;

    printf("Ingrese un carácter:");
    c = getchar(); /* entrada de un carácter */
    printf("Ingrese un valor flotante:");
    scanf("%f\n",&x);/* entrada de número en coma flotante */
    printf("Ingrese un valor entero:");
    scanf("%d\n",&i);/* entrada de enteros */
    printf("Mostrar los valores ingresados:");
    putchar(c);/* salida de un carácter */
    printf("%3d %7.4f", i, x); /* salida de números*/
}
```

Estructura de un Programa en C

El lenguaje C proporciona fundamentalmente dos formas de manejo de funciones: las internas y las externas.

Las funciones internas son las ya implementadas e incorporadas en el lenguaje de programación. Para poder hacer uso de ellas, es necesario incluir el archivo de cabecera estándar correspondiente (también llamada biblioteca) al que corresponda cada función. La forma de hacer esto es escribiendo la directiva `#include` generalmente al principio de un programa.



Librería Estándar de C

La librería estándar de C es una recopilación de ficheros cabecera y librerías con rutinas que implementan operaciones comunes, como las de entrada salida o el tratamiento de cadenas.

- El nombre y las características de cada función se encuentran en un fichero denominado archivo de cabecera (con extensión ".h"), pero la implementación real de las funciones están separadas en un archivo de la librería.
- Estas librerías se deben colocar en un programa de lenguaje programación en C , en la instrucción o directiva conocida como **include**.



Ejemplos de Librería Estándar en C

<u><ctype.h></u>	Contiene funciones para clasificar caracteres según sus tipos o para convertir entre mayúsculas y minúsculas independientemente del conjunto de caracteres (típicamente ASCII o alguna de sus extensiones).
<u><math.h></u>	Contiene las funciones matemáticas comunes.
<u><stdio.h></u>	Proporciona el núcleo de las capacidades de entrada/salida del lenguaje C (incluye la venerable función <u>printf</u>).
<u><string.h></u>	Para manipulación de cadenas de caracteres.
<u><time.h></u>	Para tratamiento y conversión entre formatos de fecha y hora.

Estructura de un Programa en C

Es posible utilizar en un programa múltiples funciones, pero siempre debe haber una función principal, de la cual depende el control del programa completo.

En el caso de C se utiliza una función llamada *main()*. Ésta constituye el programa principal y desde ahí se puede hacer uso tanto de las funciones internas como de las externas.

La estructura o sintaxis de un programa creado en C se muestra a continuación en un primer programa elemental.

Estructura de un Programa en C

El siguiente programa imprime en pantalla la frase " Hoy es lunes"

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

int main()

{
    /* Aquí va el cuerpo del programa */

    printf("Mi primer programa \n");

    return 0;
}
```



Estructura de un Programa en C

#include <stdio.h>

- **#INCLUDE ES LO QUE SE LLAMA UNA DIRECTIVA.** Sirve para indicar al compilador que incluya otro archivo. Cuando en compilador se encuentra con esta directiva la sustituye por el archivo indicado. En este caso es el archivo stdio.h que es donde está definida la función printf.
- La directiva, le dice a C que en el proceso de compilación incluya un archivo denominado **stdio.h**. Este fichero se suministra como parte del compilador de Turbo C y contiene la información necesaria para el correcto funcionamiento de la E/S de datos.
- La sentencia **#include** no es una instrucción C. **El símbolo # la identifica como una directiva.**
- **Los archivo *.h se denominan archivos de cabecera.** Todos los programas C requieren la inclusión de uno o varios archivos de este tipo, por lo que normalmente es necesario utilizar varias líneas **#include**.



Estructura de un Programa en C

main()

- Todo programa en C consta de uno o más módulos llamados funciones.
- **Una de las funciones se llama main.** El programa siempre comenzará por la ejecución de la función maín, la cual puede acceder a las demás funciones.
- Las definiciones de las funciones adicionales se deben realizar aparte, bien precediendo o siguiendo a maín
- Los **paréntesis** identifican a main() como una función. Generalmente, dentro de ellos se incluye información que se envía a la función. En este caso no hay traspaso de información por lo que no hay nada escrito en su interior. Aún así son obligatorios.
- **int** significa que la función retorna un valor.



Estructura de un Programa en C

Cuerpo de una función

- El **cuerpo de una función** (conjunto de sentencias que la componen) va enmarcado entre llaves { }. Ese es el significado de las llaves que aparecen en el ejemplo.
- Las llaves e indican el comienzo de una función, en este caso la función main y su final.
- ;
- El ";" se coloca al final de cada sentencia, es la forma que se usa en **C** para separar una instrucción de otra. Se pueden poner varias en la misma línea siempre que se separen por el punto y coma.



Estructura de un Programa en C

Comentario

/* Aquí va el cuerpo del programa */

- El compilador reconoce como comentario cualquier grupo de caracteres situados entre /* y */, aunque estén en diferentes líneas.
- Un comentario, no se ejecuta. Sirve para describir el programa. Conviene acostumbrarse a comentar los programas. Un comentario puede ocupar más de una línea
- Se pueden definir comentarios de una sola línea mediante //.

// Este comentario ocupa una sola línea

- En el caso de comentarios de una sola línea no hay indicador de fin de comentario.

Estructura de un Programa en C



printf(" Mi primer programa \n");

- Permite escribir datos en el dispositivo de salida estándar utilizando la función de biblioteca printf.
- Permite mostrar una cadena con formato y muestra la misma por la pantalla.
- La función printf del ejemplo muestra el mensaje “Mi primer programa” aparece el símbolo '\n'; este hace que después de mostrar el mensaje se pase a la línea siguiente, corresponde a un carácter Ascii no imprimible.
- La cadena con formato provee una descripción de la salida con el uso de un atributo marcador de posición específico que describe el valor esperado de un campo de entrada usando caracteres de escape “%” para especificar la posición relativa y el tipo de salida que la función debe producir.

```
printf("Color %s, numero1 %d, numero2 %05d, hex %x, real %5.2f.\n", "rojo", 12345, 89, 255, 3.14);
```

imprimirá la siguiente linea (incluyendo el carácter de nueva línea \n):

```
Color rojo, numero1 12345, numero2 00089, hex ff, real 3.14.
```



Sentencia de Asignación

- **Se utiliza para asignar o almacenar valores a variables o constantes.**
- **Es una operación que sitúa un valor determinado en una posición de memoria.**
- **El tipo de expresión debe ser del mismo tipo que el de la variable, en caso contrario en la fase de compilación se produciría un error de tipos.**



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

int y; /* Global. Declaración de la variable */

int main ()
{
    int x; /* Esta x es local a main (). Declaración de la variable local */
    y = 100; /* sentencia de asignación*/
    x = 1; /* sentencia de asignación*/

    printf ("x=%d, y=%d", x, y); /* Visualiza x=1, y=100 */

    getch();

    return 0;
}
```



Expresiones

Representa una unidad de datos simple, tal como un número o un carácter

- La expresión puede consistir en una entidad simple, como una constante, una variable, o una referencia a una función. También puede consistir en alguna combinación de tales entidades interconectadas por uno o más operadores.
- El uso de expresiones involucrando operadores.
- Las expresiones también pueden representar condiciones lógicas que son verdaderas o falsas. En C las condiciones verdadero y falso se representan por los valores 1 y 0, respectivamente. Por tanto, las expresiones lógicas representan en realidad cantidades numéricas.



Operadores

Un operador es un símbolo que indica alguna operación sobre uno o varios objetos del lenguaje, a los que se denomina operandos.

Los operadores, junto con los operandos, forman expresiones.

Una expresión se convierte en una sentencia cuando va seguida de un punto y coma.

Cuando un grupo de sentencias se encierran entre llaves { }, forman un bloque, sintácticamente equivalente a una sentencia.

Atendiendo al tipo de operación que realizan, se clasifican en :

- Aritméticos
- Relacionales
- Lógicos



Operadores Aritméticos

	OPERADOR	DESCRIPCIÓN
UNARIOS	-	<i>Cambio de signo</i>
	--	<i>Decremento en uno</i>
	++	<i>Incremento en uno</i>
BINARIOS	-	<i>Resta</i>
	+	<i>Suma</i>
	*	<i>Producto</i>
	/	<i>División</i>
	%	<i>Resto de división entera</i>



Operadores de Asignación

Las asignaciones se realizan mediante el operador =.

El uso de este operador tiene ciertos aspectos que lo distinguen del de otros lenguajes. En primer lugar, se puede emplear cualquier número de veces en una expresión, como se muestra a continuación:

$a = b = c = 3;$

que asigna el valor 3 a las variables a, b y c. Esto es así porque la operación de asignación,



Operadores de Asignación

Expresión	Equivale a	Resultado
$a+=2;$	$a=a+2;$	12
$a-=2;$	$a=a-2;$	8
$A*=2;$	$a=a*2;$	20
$a/=2;$	$a=a/2;$	5
$A\%=2;$	$a=a\%2;$	0



Ejemplo de Operadores

Suponga que i es una variable entera que tiene asignado el valor 5.

- La expresión $++i$, que es equivalente a escribir $i = i + 1$, hace que el valor de i sea 6.
- Análogamente la expresión $- -i$, que es equivalente a $i = i - 1$, hace que el valor (partiendo del original) de i pase a ser 4.
- Los operadores incremento y decremento se pueden utilizar, cada uno de ellos, de dos formas distintas, dependiendo de si el operador se escribe delante o detrás del operando.
- Si el operador precede al operando (por ejemplo $++i$), el valor del operando se modificará antes de que se utilice con otro propósito.
- Sin embargo, si el operador sigue al operando (por ejemplo $i++$), entonces el valor del operando se modificará después de ser utilizado.



Operadores Relacionales

Se usan para expresar condiciones y describir una relación entre dos valores.

resultado de una expresión relacional sólo puede ser *verdadero* o *falso*, lo que en C se identifica con los valores distinto de cero y cero, respectivamente.

```
if (a == b) printf ("Son iguales");
```

	OPERADOR	DESCRIPCIÓN
BINARIOS	>	<i>Mayor que</i>
	>=	<i>Mayor o igual que</i>
	<	<i>Menor que</i>
	<=	<i>Menor o igual que</i>
	==	<i>Igual que</i>
	!=	<i>Diferente que</i>



Ejemplo de Operadores Relacionales

Suponga que i es una variable entera cuyo valor es 7, f es una variable en coma flotante cuyo valor es 5.5 y c es una variable de carácter que representa el carácter 'w'. A continuación se muestran varias expresiones lógicas que hacen uso de estas variables. En cada expresión aparecen dos tipos diferentes de operandos.

<u>Expresión</u>	<u>Interpretación</u>	<u>Valor</u>
$f > 5$	verdadero	1
$(i + f) \leq 10$	falso	0
$c == 119$	verdadero	1
$c != 'p'$	verdadero	1
$c \geq 10 * (i + f)$	falso	0



Operadores Lógicos

Actúan sobre expresiones booleanas, es decir, sobre valores *verdadero* o *falso* generados por expresiones como las explicadas en el caso anterior.

	OPERADOR	DESCRIPCIÓN
UNARIOS	!	<i>not</i>
BINARIOS	&&	<i>and</i>
		<i>or</i>

El resultado de una operación lógica viene dado por su tabla de verdad



Ejemplo de Operadores Lógicos

Suponga que *i* es una variable entera cuyo valor es 7, *f* una variable en coma flotante cuyo valor es 5.5 y *c* una variable de carácter que representa el carácter 'w'. A continuación se muestran varias expresiones lógicas complejas en las que aparecen estas variables.

<u>Expresión</u>	<u>Interpretación</u>	<u>Valor</u>
$(i \geq 6) \ \&\& \ (c == 'w')$	verdadero	1
$(i \geq 6) \ \ (c == 119)$	verdadero	1
$(f < 11) \ \&\& \ (i > 100)$	falso	0
$(c != 'p') \ \ ((i + f) \leq 10)$	verdadero	1

La primera expresión es verdadera porque los dos operandos son verdaderos. En la segunda expresión, los dos operandos también son verdaderos, por tanto toda la expresión es verdadera. La tercera expresión es falsa porque el segundo operando es falso. Y, finalmente, la cuarta expresión es verdadera porque el primer operando es verdadero.



Ejemplo de Operadores con printf

Un programa en C incluye una variable entera i, cuyo valor inicial es 1.

Suponga que el programa incluye las tres siguientes instrucciones printf:

```
printf (" i = %d \n",i);
```

```
printf ("i =%d \n " , ++i);
```

```
printf ("i=%d \n" ,i) ;
```

Estas instrucciones printf generarán las tres líneas siguientes

i = 1

i = 2

i = 2

La primera instrucción hace que se visualice el valor original de i. La segunda instrucción incrementa i y presenta después su valor. La última instrucción visualiza el valor final de i.



Estructura de un Programa en C

Ejemplo:

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

```
int main ()
{
    /* escribir varios números en coma flotante */
    float i = 2.0, j = 3.0;
    printf("%f %f %f %f", i, j, i+j, sqrt(i+j));
    return 0;
}
```

- Observe que los dos primeros argumentos dentro de la función printf son variables simples, el tercer argumento es una expresión aritmética y el último argumento una referencia a una función que tiene una expresión numérica como argumento.
- La ejecución del programa produce la siguiente salida:

2.000000 3.000000 5.000000 2.236068



Estructura de un Programa en C

return 0;

- Finaliza la ejecución de una función y devuelve el control a la función de llamada (o al sistema operativo si se transfiere el control de la función main). La ejecución se reanuda en la función de llamada, en el punto que sigue inmediatamente a la llamada.
- Las funciones de tipo void, no pueden especificar una expresión en la instrucción return. Las funciones de todos los demás tipos deben especificar una expresión en la instrucción return.

Funciones de Biblioteca



El lenguaje C se acompaña de un cierto número de funciones de biblioteca que realizan varias operaciones y cálculos de uso frecuente.

Función	Tipo	Propósito
<code>abs(i)</code>	<code>int</code>	Devolver el valor absoluto de <code>i</code> .
<code>ceil(d)</code>	<code>double</code>	Redondear por exceso al entero más próximo (el entero más pequeño que sea mayor o igual a <code>d</code>).
<code>cos(d)</code>	<code>double</code>	Devolver el coseno de <code>d</code> .
<code>cosh(d)</code>	<code>double</code>	Devolver el coseno hiperbólico de <code>d</code> .
<code>exp(d)</code>	<code>double</code>	elevar <code>e</code> a la potencia <code>d</code> (<code>e=2.7182818...</code> es la base del sistema logarítmico natural (Neperiano)).
<code>fabs(d)</code>	<code>double</code>	Devolver el valor absoluto de <code>d</code> .
<code>floor(d)</code>	<code>double</code>	Redondear por defecto al entero más próximo (el entero más grande que no sea mayor que <code>d</code>).
<code>fmod(d1,d2)</code>	<code>double</code>	Devolver el resto de <code>d1/d2</code> (parte no entera del cociente), con el mismo signo que <code>d1</code> .
<code>getchar()</code>	<code>int</code>	Introducir un carácter desde el dispositivo de entrada estándar.
<code>log(d)</code>	<code>double</code>	Devolver el logaritmo natural de <code>d</code> .
<code>pow(d1,d2)</code>	<code>double</code>	Devolver <code>d1</code> elevado a la potencia <code>d2</code> .
<code>printf(...)</code>	<code>int</code>	Mandar datos al dispositivo de salida estándar (los argumentos son complicados; ver Capítulo 4).
<code>putchar(c)</code>	<code>int</code>	Mandar un carácter al dispositivo de salida estándar.
<code>rand()</code>	<code>int</code>	Devolver un entero positivo aleatorio.
<code>sin(d)</code>	<code>double</code>	Devolver el seno de <code>d</code> .
<code>sqrt(d)</code>	<code>double</code>	Devolver la raíz cuadrada de <code>d</code> .
<code>srand(u)</code>	<code>void</code>	Inicializar el generador de números aleatorios.
<code>scanf(...)</code>	<code>int</code>	Introducir datos del dispositivo de entrada estándar (los argumentos son complicados; ver Capítulo 4).
<code>tan(d)</code>	<code>double</code>	Devolver la tangente de <code>d</code> .
<code>toascii(c)</code>	<code>int</code>	Convertir el valor del argumento a ASCII.
<code>tolower(c)</code>	<code>int</code>	Convertir una letra a minúscula.
<code>toupper(c)</code>	<code>int</code>	Convertir una letra a mayúscula.

Ejemplo Funciones de Biblioteca



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

int main()
{
    /* solución de una ecuación cuadrática */
    double a,b,c, raiz, x1, x2;

    /* leer valores de a, b y c */
    a=3;
    b=11;
    c=-4;

    raiz = sqrt(b * b -4*a*c);
    x1=(-b + raiz)/(2* a);
    x2=(-b - raiz)/(2* a);

    /* escribir valores de a, b, c, x1 y x2*/
    printf ("a=%f\n",a);
    printf ("b=%f\n",b);
    printf ("c=%f\n",c);
    printf ("x1=%f\n",x2);
    printf ("x2=%f\n",x1);
    return 0;
}
```

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

```
int main ()
```

```
{
```

```
    int base, altura,perimetro;
    float area;
    printf( "Entra la base:" );
    scanf( "%d", &base );
    printf( "Entra la altura:" );
    scanf( "%d", &altura );
    area=(base*altura)/2;
    perimetro=(2*base)+(2*altura);
    printf( "El area es %f.\n", area );
    printf( "El perimetro es %d.\n", perimetro );
    return(0);
}
```

The logo for Zinjal, featuring a stylized blue and red 'Z' shape with two red plus signs to its right, all set against a yellow circular gradient background.

(Zinjal is not just another IDE)

Lenguaje de Programación C

```
#include <stdio.h>
#include <conio.h>
int main ()
{
    long int n1, n2;
    printf ("\nTeclee 2 numeros enteros: ");
    printf ("\nIngrese el primer entero: ");
    scanf ("%5ld , &n1);
    printf ("\nIngrese el segundo entero: ");
    scanf ("%5ld ", &n2);

    /*5ld indica un entero largo de 5 posiciones*/
    printf ("\nLos números tecleados son %ld y %ld", n1, n2);
    return(0);
}
```

(Zinjal is not just another IDE)

