# *Lenguaje de programación C*

## Introducción

El lenguaje C es un lenguaje de programación de propósito general, es uno de los más rápidos y potentes que existen. El lenguaje C ha demostrado ser un lenguaje extremadamente eficaz, hasta como para crear sistemas operativos. . Este lenguaje tiene ciertas características, unas de ellas son:

- **Es un lenguaje Compilado.** A diferencia de otros lenguajes, que son lenguajes interpretados, los cuales necesitan del código fuente para funcionar (por ejemplo Basic), C es un lenguaje compilado esto quiere decir que convierte el código fuente en un fichero objeto, éste es enlazado con las librerías necesarias dando lugar a un fichero ejecutable.
- **Es un lenguaje de Nivel medio.** Esto quiere decir que combina elementos de lenguaje de alto nivel con la funcionalidad del lenguaje ensamblador, o sea que trabaja a un nivel cercano al computador, sin embargo, nos ofrece posibilidades de construir estructuras de datos equivalentes a los que manejan los lenguajes de alto nivel.
- **Es un lenguaje Estructurado.** Esto quiere decir que permite crear procedimientos en bloques dentro de otros procedimientos.
- **Es un lenguaje Portable.** Este lenguaje permite utilizar el mismo código en diferentes equipos y sistemas informáticos, o sea que es independiente de la arquitectura de cualquier máquina.
- **Es un lenguaje Relativamente Pequeño.** Este es un lenguaje económico en cuanto a expresiones se refiere, se puede describir en poco espacio y es fácil y rápido de aprender.
- **Tiene abundancia de Operadores y Tipos de Datos.** Este lenguaje prácticamente posee un operador para cada una de las posibles operaciones en código de máquina.

Pero al igual que otros lenguajes, el C tiene sus desventajas: .

- Carece de instrucciones entrada/salida.
- Carece de instrucciones de manejo de cadena de caracteres.
- La precedencia de los operadores dificultan la comprensión de algunas expresiones.

Todo programa en C, se compone de una o más funciones. La función que se ejecuta en primera instancia es una función llamada main( ), esta función es la más importante de todo el programa y es la que nunca debe faltar. A esta función no se le puede cambiar el nombre; main( ) es el cuerpo principal de nuestro código fuente. A su vez toda función del programa debe devolver un valor al programa, este valor se indica con la palabra reservada "return". Ciertas veces no queremos que la función devuelva ningún valor para estos casos simplemente le indicamos "return 0" o escribimos la palabra reservada "void" antes del nombre de la función.

### *Comentarios*

Un buen programa debe contener buenos comentarios. Así, si lo retomamos en el futuro no tendremos que descifrar lo que hicimos en el pasado. Es una buena costumbre del programador comentar sus programas. Un buen comentario es aquel que ni ofende el conocimiento del

programador, ni tampoco deja implícito lo que se está haciendo. Para poner un comentario basta con encerrar el texto entre los signos:

*/** [*comentario*] ***/*  ó comentar el resto de la línea:  // [*comentario*]

Ejemplo:

```
int main() {
{
    /* este es un comentario que puede afectar a un
    bloque de varios renglones o a un solo renglón */

    // este comentario de un solo renglón, total o parcial.
}
```

## Estructura de un programa escrito en C

Cuando usamos un programa es muy importante manejar datos. En C podemos almacenar los datos en variables. El contenido de las variables se puede ver o cambiar en cualquier momento. Estas variables pueden ser de distintos tipos dependiendo del tipo de dato que queramos guardar en ellas. No es lo mismo guardar un nombre que un número. Hay que recordar también que la memoria del ordenador es limitada, así que cuando guardamos un dato, debemos usar sólo la memoria necesaria.

***Identificadores:***

Son nombres dados a constantes, variables y funciones. El identificador está formado por una secuencia de una o más letras, dígitos y el caracter de subrayado. El primer caracter de un identificador *siempre* debe ser una letra o el caracter de subrayado, en estos identificadores (nombres de variables) se debe tener gran cuidado, ya que el C hace diferencia entre mayúsculas y minúsculas.

Por ejemplo, Min, min y MIN son tres identificadores diferentes.

***Constantes:***

Las constantes son un valor que una vez fijado por el compilador, no cambia durante la ejecución del programa. El valor almacenado en una constante puede ser un número o bien una cadena de caracteres ó un caracter suelto. El lenguaje C indica al programa que se trata de una constante usando la directiva **#define** (preprocesador) o con el modificador **const** (compiladores modernos)

Por ejemplo:

```
#define PI 3.141592    /* constante pi */
```

En cualquier sitio en el que se utilice **PI**, el preprocesador lo sustituye por el valor 3.141592 (aún se puede utilizar este método en C y C++).

Cuando se utiliza el preprocesador para crear constantes, su control queda fuera del ámbito del compilador. No existe ninguna comprobación de tipo y no se puede obtener la dirección de PI (de modo que no se puede pasar un puntero o una referencia a PI). PI no puede ser una variable de un tipo definido por el usuario. El significado de PI dura desde el punto en que es definida, hasta el final del fichero; el preprocesador no entiende de ámbitos.

En los compiladores modernos de C, se introduce el concepto de constantes con nombre que es lo mismo que una variable, excepto que su valor no puede cambiar. El modificador **const** le indica al compilador que el nombre representa una constante. Cualquier tipo de datos predefinido o definido por el usuario, puede ser definido como **const**. Si se define algo como **const** y luego se intenta modificar, el compilador generará un error.

Se debe especificar el tipo de un **const**, de este modo:

```
const int x = 10;
```

En C, el compilador trata a **const** del mismo modo que a una variable que tuviera asociado una etiqueta que dice ***«No me cambies»***. Cuando se define un **const** en C, el compilador pide espacio para él, de modo que si se define más de un **const** con el mismo nombre en dos ficheros distintos (o se ubica la definición en un fichero de cabeceras), el enlazador generará mensajes de error sobre del conflicto.

# Tipos de Datos

*Tipo* **int** /* número entero */

En una variable de este tipo se almacenan números enteros (sin decimales). El rango de valores que admite es -32768 a 32767. Cuando definimos una variable lo que estamos haciendo es decirle al compilador que nos reserve una zona de la memoria para almacenar datos de tipo int. Hay que decirle al compilador que queremos crear una variable y hay que indicarle de qué tipo. Si esta declaración se hace antes de main( ), esto indica que la variable será una variable global, o sea que se usará en main( ) y en el resto de funciones. Mientras que si se declara dentro de main( ), esto indica que la variable será una variable local, o sea, sólo para la función main( ).

Por ejemplo:

```
int valor;
int main( )...
/* Esta expresión declara una variable global de nombre "valor" y de tipo
entero. */
```

```
int main( )
{
   float valor ; /* Esta expresión declara una variable local de nombre */
}                /*"valor" y de tipo float (coma flotante) */
```

*Tipo* **char** /* caracter */
Las variables de tipo char sirven para almacenar caracteres. Los caracteres se almacenan en realidad como números del 0 al 255. Los 128 primeros (0 a 127) son el ASCII estándar. El resto es el ASCII extendido y depende del idioma y del ordenador. Para declarar una variable char solo escribimos: `char letra ;` y listo, ya existe una variable *"letra"* que almacenará datos de tipo caracter.

*Tipo* **float** /* número real, punto flotante de simple precisión*/
En este tipo de variable podemos almacenar números decimales, no sólo enteros como en los anteriores. El rango de posibles valores es del 3.4 E-38 al 3.4 E+38.
Para declarar un float, de forma similar, escribimos: `float prom ;` y con esto hemos declarado una variable de nombre *"prom"* que tendrá un dato de tipo float o coma flotante.

*Tipo* **double** /* número real, punto flotante de doble precisión*/
En las variables tipo double se almacenan números reales, estos tiene el rango de 1.79769 E-308 a 1.79769 E+308. Al igual que las anteriores para declararla escribimos: `double num ;` y tenemos una variable de nombre *"num"* que aceptará datos tipo double.

*Modificador* **unsigned** /* sin signo*/

Este modificador (que significa: sin signo) modifica el rango de valores que puede contener una variable. Sólo admite valores positivos. Este modificador se usa así: `unsigned int total ;` esto declara un variable de nombre "total", de tipo entero positivo.

**Resumen De Tipos De Datos**

| Tipo de Dato | Dato almacenado | Tamaño (en bytes) | Rango | Observaciones |
|---|---|---|---|---|
| **unsigned char** | Caracteres | 1 | 0 a 255 | Valores correspondientes a números ordinales de los 256 caracteres ASCII |
| **char** | Caracteres | 1 | -128 a 127 | char letra = 'A'; ó letra = 65; |
| **short int** | Entero | 2 | -32768 a 32767 | Proporciona un entero en el rango dado |
| **unsigned int** | Entero | 2 | 0 a 65535 | Un entero positivo |
| **int** | Enero | 2 | -32768 a 32767 | Equivalente al tipo short en máquinas de 16 bits |
| **long int** | Entero | 4 | -2147483648 a 2147483647 | tamaño(int) <= tamaño(long) |
| **unsigned long** | Entero | 4 | 0 a 4294967295 | entero largo sin signo |
| **float** | Números Reales | 4 | -3.4 E-38 a 3.4 E+38 | no tiene más de 5 dígitos significativos |
| **long double** | Números Reales (coma flotante con doble precisión) | 10 | 3.4 E-4932 a 1.18 E+4932 | Puede tener hasta 19 dígitos significativos |
| **double** | Números Reales (coma flotante con doble precisión) | 8 | 1.79769 E-308 a 1.79769 E+308 | Tiene hasta 16 dígitos significativos |
| **definido por el usuario** | Usando estructuras | xx | depende del tipo de dato | XXXXXXXXXXXXXX |

***Observación***: los tamaños en bytes que expresa esta tabla son los que nos brinda portabilidad, existen compiladores modernos que han ampliado estos rangos

## Introducir e Imprimir Datos

Es usual que en un programa el usuario introduzca datos por el teclado. Para ello contamos con varias posibilidades: Usar las funciones de la biblioteca estándar es la más usual de éstas, aunque no es la mejor es la más fácil de aprender y es la que nosotros utilizaremos.

***scanf ( )*** /* leer desde el teclado*/
Al utilizar la palabra scanf(), nosotros podemos hacer que el usuario introduzca datos por el teclado. El uso de scanf() es de la siguiente manera: `scanf("%i", &num);` esta última expresión indica al programa que se debe solicitar un número entero (%i, es una especificación de formato y esto lo puedes ver en la sección de operadores y manipuladores), luego se indica el nombre de la variable en la que se va a guardar el dato, éste (el nombre) precedido por el signo '&', que indica al programa la dirección de ésta variable.

***printf ( )*** /* mostrar por pantalla*/
El uso de printf(), es similar al de scanf(), lo que scanf() hace para leer una variable, printf() lo hace para imprimirla. De esta manera podemos imprimir en pantalla, ya sea un mensaje o el valor contenido en una variable. La sintaxis de printf() es así: `printf("%d", num);` esta sentencia lo que hace es imprimir el valor de la variable num en el lugar donde aparece el "%d". El printf() nos sirve también para mandar a imprimir en pantalla un mensaje así:
`printf ("Bienvenidos al mundo de C") ;`

esto imprimirá en pantalla:

```
Bienvenidos al mundo de C
```

Nota: Para hacer uso de éstas y otras funciones, es necesario incluir el archivo de cabecera **stdio.h**, esto se hace de la siguiente manera: `#include<stdio.h>`, así queda incluido éste archivo y se podrá proceder con el programa. Ahora haremos uso de lo que hemos aprendido para hacer un programa:

```c
# include <stdio.h>
int main()
{
    int x;
    printf("Bienvenidos al mundo de C\n");
    printf("Introduzca un valor entero: ");
    scanf("%d", &x);
    printf("\nEl valor que introdujo fue %d", x);
    return 0;
}
```

Ahora explicaremos lo que hace:

- Primero se declara una variable de tipo entero y de nombre x.
- Después se manda a imprimir en pantalla el mensaje "Bienvenidos al mundo de C", el "\n" que sigue es para dar un salto de línea en el programa, si no lo pusiéramos el siguiente mensaje

aparecería pegado al anterior.

- Luego se imprime "Introduzca un valor entero", para que el usuario pueda introducir un valor por medio del teclado y se manda a guardar este valor a la variable antes declarada.
- Después se imprime un mensaje y el valor de la variable.
- La expresión "return 0;" es una instrucción.
- Como verán, toda función debe retornar un valor, como no queremos que nuestra función main( ) devuelve nada, entonces le decimos que retorne 0 (cero).

La salida del programa anterior sería:

```
Bienvenidos al mundo de C Introduzca un valor entero: 25
/* por ejemplo 25 */
El valor que introdujo fue 25
```

**Indicadores de formato y secuencias de escape**

| | |
|---|---|
| ***%d*** | Número entero con signo, en notación decimal |
| ***%i*** | Número entero con signo, en notación decimal |
| ***%u*** | Número entero sin signo, en notación decimal |
| ***%o*** | Número entero sin signo, en notación octal (base 8) |
| ***%x*** | Número entero sin signo, en hexadecimal (base 16) |
| ***%X*** | Número entero sin signo, en hexadecimal, mayúsculas |
| ***%f*** | Número real, float (coma flotante, con decimales) |
| ***%e*** | Número real en notación científica |
| ***%g*** | Usa el más corto entre %e y %f |
| ***%c*** | Un único carácter |
| ***%s*** | Cadena de caracteres |
| ***%%*** | Signo de tanto por ciento: % |
| ***%p*** | Puntero (dirección de memoria) |
| ***%ld*** | Número entero largo con signo, long int (long) |
| ***%lf*** | Número real, double (coma flotante, con decimales) |

**Secuencias de escape**

| Código | Significado |
|---|---|
| “ \n “ | nueva línea |
| “ \r " | retorno de carro |
| " \f " | nueva página |
| " \t " | tabulador horizontal |
| " \b " | retroceso (backspace) |
| “ \’ “ | comilla simple |
| " \" " | Comillas |
| " \\ " | Barra |

**Otras funciones para capturar datos**

**getch( ) - getche( ) – getc( ) – getchar( ) – gets( )**

Si lo que queremos es que el usuario introduzca un caracter por el teclado usamos las funciones getchar( ), getc( ), getch( ) y getche( ). Estas esperan a que el usuario introduzca un carácter por el teclado. La diferencia entre getc( ), getche( ), getchar( ) y getch( ) es que las tres primeras imprimen en pantalla la tecla que hemos pulsado y la última no.

Observaremos además que getc( ) y getchar( ) necesitan de la tecla “[ENTER]” para confirmar la captura. Por ejemplo:

```c
# include< stdio.h>
# include< conio.h>
/* los compiladores más modernos traen portabilidad para "conio.h", un
archivo cabecera o HEADER que antiguamente era exclusivo de Borland
International.*/
int main( )
{
    char letra;
    printf ( "Introduzca una letra: " );
    fflush(stdin); // leer nota al pie de página [1]
    letra = getche( );
    printf ( "\nLa letra que introdujo es: %c", letra );
    return 0;
}
```

[1] La función del fflush, realiza la limpieza del buffer de entrada (**stdin**) standard input. Usualmente quedan almacenados algunos datos en el buffer sobre todo saltos de línea y espacio y se debería usar antes de cada scanf con el fin de garantizar que el buffer esté limpio

La salida del programa anterior sería:

```
Introduzca una letra: N /* por ejemplo N */
La letra que introdujo es: N
```

Ahora hagamos el mismo programa con getch( ): y con getc( ):

```
# include< stdio.h>
# include< conio.h>
int main( )
{
  char letra;
  printf ( "Introduzca una letra: " );
  fflush(stdin);
  letra = getch( );
  printf ( "\nLa letra que introdujo
es: %c", letra );
  return 0;
}
```

```
# include< stdio.h>
# include< conio.h>
int main( )
{
  char letra;
  printf ( "Introduzca una letra: " );
  letra = getc(stdin);
  printf ( "\nLa letra que introdujo
es: %c", letra );
  return 0;
}
```

La salida de los programas anteriores sería:

**Con getch( ):**

```
Introduzca una letra:
/* por ejemplo N */
La letra que introdujo es: N
```

**Con getc( ):**

```
Introduzca una letra: F
/* por ejemplo N */
La letra que introdujo es: F
```

Se debe tener en cuenta, se haga o no en estos ejemplos, que los archivos de cabecera (“stdio.h”/“conio.h”) deben estar entre los signos **< >**, de lo contrario se produciría un error en el programa.

**Captura de Cadenas de Caracteres**

Las cadenas de caracteres o strings las trataremos más adelante, a la hora de ver colecciones de variables, pero podemos ir mencionando la posibilidad de captura de este tipo compuesto de datos a partir del uso de funciones como scanf ( ) ó gets ( ). Para empezar diremos que una variable de tipo string que utilizaremos para guardar un nombre, en C la declararemos como: char nombre [15]; (15 será la cantidad máxima de caracteres que soportará, y su uso con las funciones de captura desde el teclado será el siguiente:

```c
#include<stdio.h>
void main ( )
{
    char nombre [15];
    printf("Ingrese su nombre: ");
    fflush(stdin);
    scanf("%s", nombre);
}
```

```c
#include<stdio.h>
void main ( )
{
    char nombre [15];
    printf("Ingrese su nombre: ");
    fflush(stdin);
    gets (nombre);
}
```

## Operadores (Manipuladores)

Los operadores son símbolos que indican cómo son manipulados los datos. Estos operadores se pueden clasificar en: aritméticos, lógicos, relacionales, de asignación, de manejo de bits y otros.

***Operadores de Asignación:*** Estos operadores son los que nos sirven para darle (asignarle) un valor a una variable. Este valor lo podemos teclear manualmente o bien puede ser el valor de otra variable. Por ejemplo:

```c
x = 5 ; /*Esto le carga a x un valor directamente*/
x = a ; /*Esto le asigna a x el valor de otra variable*/
```

***Operadores Aritméticos:*** Los operadores aritméticos son aquellos que sirven para realizar operaciones tales como suma, resta, división y multiplicación. Con ellos podemos calcular el resto o módulo de una división entera, o bien podemos incrementar o disminuir el valor de las variables. Por ejemplo:

```c
# include <stdio.h>
int main( ) {
  int x = 7 , y, int z = 2 ;
  /* es posible declarar e iniciar variables al mismo tiempo */
  printf ("El valor de x es: %d\n", x);
  y = x + z ; /* asigna el valor de una suma */
  x ++ ; /* incrementa el valor de x en uno */
  printf ("Los valores de x, y, z son: %d, %d, %d", x, y, z); return 0;
}
```

La salida a este programa sería:

```
El valor de x es: 7
Los valores de x, y, z son: 8, 9, 2
```

***Operadores Lógicos:*** Estos operadores son los que nos permiten hacer comparaciones entre dos o más variables. Estos operadores son los que nos permitirán decidir en una sentencia condicional que veremos más adelante. Por ejemplo:

```
(x < 7) && (t = = 4); /* Esta línea devuelve 1 (verdadero), si x es
menor que 7 "y" t es igual a 4 */
```

***Operador Sizeof( ):*** Existe un operador muy utilizado que es el sizeof( ). Este operador nos devuelve el tamaño en bytes de una variable. De esta manera no tenemos que preocuparnos en recordar o calcular cuánto ocupa. Además el tamaño de una variable cambia de un compilador a otro, es la mejor forma de asegurarse. Se usa poniendo el nombre de la variable después de sizeof y separado de un espacio. Por ejemplo:

```
int main( ) {
  float variable;
  printf ( "Tamaño de la variable: %d\n”, sizeof variable);
  printf ( "Tamaño de un entero: %d", sizeof (int));
}
```

Se debe tener gran cuidado al agrupar estos operadores, pues unas acciones se llevan a cabo antes que otras, esto depende del nivel de precedencia que tenga el operador. Para esto se presenta una tabla, en la que se muestra en orden descendente la prioridad de los mismos.

**Operadores Lógicos**

| Operador | Operación |
|---|---|
| && | AND, en español Y. Da como resultado el valor lógico 1, si ambos operadores son distintos de cero |
| \|\| | OR, en español O. El resultado es cero, si ambos operandos son cero |
| ! | NOT, en español NO. El resultado es cero si el operando tiene un valor distinto de cero |

**Operadores Aritméticos**

| Operador | Operación |
|---|---|
| + | Suma dos numeros, pueden ser enteros o reales |
| - | Resta dos números, pueden ser enteros o reales |
| * | Multiplica dos números, pueden ser enteros o reales |
| / | División, el resultado depende de los tipos de datos |
| % | Módulo o resto de una división entera, los operandos tienen que ser enteros |
| ++ | Incrementa en uno la variable |
| -- | Decrementa en uno la variable |

**Operadores de Relación**

| Operador | Operación |
|---|---|
| < | Primer operando menor que el segundo |
| > | Primer operando mayor que el segundo |
| <= | Primer operando menor o igual que el segundo |
| >= | Primer operando mayor o igual que el segundo |
| == | Primer operando igual que el segundo |
| != | Primer operando distinto del segundo |

**Operadores de Asignación**

| Operador | Operación |
|---|---|
| = | Asignación simple |
| *= | Multiplicación más asignación |
| /= | División más asignación |
| %= | Módulo más asignación |
| += | Suma más asignación |
| -= | Resta más asignación |

**Niveles de Precedencia**

| Operador |
|---|
| ( ) [ ] |
| ! ++ -- |
| * / % |
| + - |
| == != |
| ! |
| && |
| \|\| |
| = *= /= %= += -= |

***Operadores avanzados***

Los operadores de **incremento, decremento** y **asignación compuesta** permiten modificar el contenido de una variable de forma eficiente y abreviada.

**Operadores Significado** A++, ++A Incrementa en 1 el valor de A (A=A+1) A--, --A Disminuye en 1 el valor de A (A=A-1)

A+=x A=A+x A-=x A=A-x A*=x A=A*x A/=x A=A/x

***Operadores "pre" y "post" y valor devuelto***

Si el operador **++** o **--** se coloca a la izquierda, se llama **preincremento** o **predecremento**, respectivamente. Si se coloca a la derecha, se llama **postincremento** o **postdecremento**. Cuando se escriben estas expresiones dentro de expresiones más complejas, el valor que se devuelve es:

- Operaciones "pre": El valor *nuevo* de la variable afectada
- Operaciones "post": el valor *anterior* de la variable afectada

Ejemplo:

```
x = 1;
A = ++x;     //preincremento: A valdrá 2, x valdrá 2

x = 1;
A = x++;     // postincremento: A valdrá 1, x valdrá 2
```

Las asignaciones compuestas devuelven el nuevo valor de la variable:

```
x = 2 ;
A = (x *= 3) + 1 ; //x valdrá 6, A valdrá 7
```

**Bibliografia**

Eckel, B. (s.f.). Thinking in C++, Volumen 1. Recuperado 28 febrero, 2020, de http://arco.inf-cr.uclm.es/%7Edavid.villa/pensar_en_C++/vol1/ch03s06s05.html

Joyanes Aguilar y Zahonero Martínez (2005), “Programación en C” Segunda Edición en español, Editorial Mc Graw Hill, España.