# Objetos de un programa en C

Un programa se construye a base de objetos y separadores (comentarios), los objetos son símbolos especiales, identificadores, números, constantes, variables, directivas, etc.

## Comentarios

Un comentario es cualquier información o documentación que se escribe el en programa para proporcionar información de cualquier tipo. El compilador ignora los comentarios por lo cual son totalmente opcionales, aunque son sumamente útiles y recomendables.

Los comentarios en C estándar comienzan con la secuencia /* y termina con la secuencia */. Otra forma de incorporar un comentario es utilizando la secuencia //.

Veamos los distintos usos en la siguiente ilustración:

```c
int main()
{
   /* Este es un comentario que abarca
      varias líneas*/
    float /*variable real*/ d;
    int a,b;//La doble barra inserta un comentario de una línea
    int c;
    c=a+b;
    /*Este también es un comentario de una sola línea*/
return 0;
}
```

## Identificadores

Los identificadores (Id) representan los objetos de un programa, es decir los nombres usados para referirse a las constantes, variables, funciones y demás objetos definidos por el usuario.

Un identificador puede estar formado por letras, y dígitos. El lenguaje C distingue entre mayúsculas y minúsculas, así, por ejemplo: - suma, Suma, SUMA -, son tres identificadores distintos, (cuidado con esto).

El carácter subrayado (_), también lo podemos incluir como una letra.

Reglas específicas de un identificador:

- Consta de uno o más caracteres.
- El primer carácter debe ser una letra o el carácter subrayado (_), el resto pueden ser letras, dígitos y caracteres subrayados (_).
- Las letras pueden ser caracteres del alfabeto inglés, no pudiendo usar "ñ" o "Ñ".
- Las vocales no pueden llevar tilde ni diéresis.
- No puede haber dos identificadores iguales, lo que no quiere decir que no puedan repetirse a lo largo del algoritmo.
- No se puede utilizar una palabra reservada como identificador. Esto incluye nombre de funciones definidas por el usuario y de la biblioteca de C

Ejemplos de identificadores:

```
...
int suma(int,int);
void cantidad_de_letras(char);
...
float dato, resultado,_balance;
int numero, valor_1, dato2;
char nombre;
int a,b;
```

## Tipos de datos

Los diferentes objetos de información con los que un programa trabaja se conocen como datos. Todos los datos tienen asociado un "tipo", que puede ser un simple carácter, tal como 'A', un valor entero 35, un número real 3.12.

El lenguaje C no soporta un gran número de tipo de datos predefinidos, pero tiene la capacidad para crear sus propios tipos. Los tipos básicos o también denominados primitivos se pueden observar en la tabla a continuación.

| TIPO | TAMAÑO EN BITS | RANGO |
|---|---|---|
| Char | 8 | 0 a 255 |
| Int | 32 | -2147483648 a 2147483647 |
| Float | 32 | -2147483648 a 2147483647 |
| Double | 64 | 1.7E-308 a 1.7E+308 |
| Void | 0 | Sin valor |

En esta tabla mostramos los valores típicos que se muestran en la bibliografía vigente, sin embargo, estos pueden variar de un compilador a otro.

Los valores de tipo char se usan para guardar valores definidos en el juego de caracteres ASCII, es decir, se almacenan como número. Por ejemplo, el carácter 'A' se almacena como el número 65.

Las variables de tipo int se usan para guardar cantidades enteras. Las de tipo float o double se usan para guardar números reales, estos últimos dos mencionados tienen un componente entero y uno fraccionario.

El tipo void tiene tres usos, uno para declarar explícitamente una función que no devuelve valor alguno; otro para declarar explícitamente una función sin parámetros, el último para crear punteros genéricos. Luego cuando comencemos a usarlos haremos hincapié en estos usos.

## Modificadores de tipos

Excepto el tipo void, los tipos de datos básicos pueden tener distintos modificadores precediéndolos, lo que altera el significado del tipo base, los modificadores son cuatro:

- short (corto).
- long (largo).
- signed (con signo).
- unsigned (sin signo).

Los cuatro modificadores se pueden aplicar a los tipo base entero y carácter. El modificador long también se puede aplicar a double la siguiente muestra las posibles combinaciones

| TIPO | TAMAÑO EN BITS | RANGO |
|---|---|---|
| unsigned char | 8 | 0 a 255 |
| signed char | 8 | -128 a 127 |
| unsigned int | 16 | 0 a 65535 |
| signed int | 16 | -32768 a 32767 |
| short int | 16 | -32768 a 32767 |
| unsigned short int | 16 | 0 a 65535 |
| signed short int | 16 | -32768 a 32767 |
| Long int | 32 | -2147483648 a 2147483647 |
| signed long int | 32 | -2147483648 a 2147483647 |
| unsigned long int | 32 | 0 a 4294967295 |
| unsigned long | 32 | 0 a 4294967295 |
| long double | 64 ó 80 (según versión). | 1.7E-308 a 1.7E+308 ó 3.4E-4932 a 1.1E+4932 |

# Variables

Las variables son palabras que manipulan datos, para ser más precisos diremos que una variable es un espacio reservado en la memoria con nombre, que tiene la capacidad de cambiar de contenido mientras se desarrolla la ejecución de un determinado programa.

# Ámbito de una variables

Es el ámbito donde la variable está disponible.

# Variables locales

Son aquellas que se declaran dentro de una función y solo pueden ser denominadas por sentencias que estén dentro del bloque en el que fueron declaradas.

Una de las cosas más importantes que hay que comprender sobre las variables locales es que solo existen mientras se está ejecutando el bloque de código en el cual fueron declaradas. Es decir, la variable local se crea al entrar en el bloque y se destruye al salir de él.

# Variables globales

A diferencia de las variables locales las variables globales se conocen a lo largo de todo el programa y se pueden usar en cualquier parte del código. Estas variables pueden ser accedidas por cualquier expresión independientemente de la función en la que se encuentre la expresión.

Cabe mencionar que todos estos usos los iremos desarrollando mejor cuando comencemos a trabajar con ejercicios y sobre todo cuando tratemos el tema de funciones.

# Declaración de variables

Todas las variables han de ser declaradas antes de poder ser utilizadas. La forma general o sintaxis de declaración es la que se muestra a continuación:

```
<Tipo de dato> <Identificador>;
```

El tipo de dato determina el tipo de datos que almacena la variable y corresponden a los tipos definidos más arriba. Identificador, es el nombre que deseamos darle a la variable.

De manera opcional podemos a su vez asignarle un valor en el mismo momento en que la estamos inicializando y la sintaxis sería:

```
<Tipo de dato> <Identificador> = <valor inicial>;
```

También se pueden declarar múltiples variables en la misma línea esto se hace separando con comas cada variable a declarar del siguiente modo:

```
<Tipo de dato> <Identificador1>,<Id2> ... <IdN>;
```

Ejemplos:

```
char nombre;/*Declaración de una variable tipo
              carácter denominada "nombre"*/
float dato, valor,res;/*Declaración de múltiples
                        variables*/
int numero=2;/*Declaración y asignación de un
               valor al mismo tiempo*/
```

## Constantes

Las constantes en C se refieren a valores fijos que no pueden ser alterados a lo largo del programa.

Las constantes enteras y de coma flotante son llamadas, constantes de tipo numérico. Las siguientes reglas se pueden aplicar a todas estas constantes de tipo numérico.

- No se pueden incluir ni comas ni espacios en blanco en las constantes.
- Si lo deseamos, la constante puede ir precedida de un signo (-).
- El valor de la constante no puede exceder de un límite máximo y un mínimo especificados. Para cada tipo de constante, estos límites varían de un compilador a otro.

Constantes de carácter, es un solo carácter, encerrado entre comillas simples, además acepta otro tipo de constante que no está dentro de los predefinidos que es el tipo cadena. Una constante de este tipo siempre está encerrada entre dobles comillas. Ej: "Buen Comienzo". Luego discutiremos en detalle el tipo de dato cadena.

# Declaración de constantes

## Constantes declaradas

Se declaran mediante la palabra reservada const es un calificador de tipo variable e indica que el valor de la variable no se puede modificar, la sintaxis es la siguiente:

```
const <tipo_de_dato><Identificador_de_constante> = <valor>;
```

Ejemplo:

```
Const int valor=200;
```

## Constantes definidas

Se declaran mediante la directiva #define de la siguiente forma:

```
#define <Identificador> <valor inicial>
```

Ejemplo:

```
#define a 3
```

## Operadores

El lenguaje C es muy rico en operadores incorporados. Un operador es un símbolo que indica al compilador que lleve a cabo ciertas manipulaciones matemáticas o lógicas y a nivel de bits. Existen operadores aritméticos, relacionales y lógicos. También tiene operadores especiales para determinadas tareas.

## Aritméticos

La tabla, a continuación muestra los operadores aritméticos básicos del lenguaje C.

| OPERADOR | ACCIÓN |
|---|---|
| + | Adición |
| - | Sustracción |
| * | Multiplicación |
| / | División |
| % | Resto de división entera |

Pueden aplicarse a casi todos los tipos de datos predefinidos en C.

Cuando se aplica / (división) a un entero o a un carácter, debemos tener en cuenta estos casos:

- Si dicha división se produce para dos operandos enteros el resultado será un número entero que será truncado, o sea, obtendremos sólo la parte entera del resultado.
- Si la división es entre dos números en coma flotante el resultado será otro número en coma flotante.
- Si la división es entre un número entero y otro en coma flotante, el resultado será un número en coma flotante.
- Si la división tiene algún operando negativo, o los dos, entonces la división entera (entre dos números enteros) estará truncada hacia cero, esto quiere decir que el resultado será siempre menor en valor absoluto que el verdadero cociente.

En C el operador de división en módulo (%), actúa igual que en otros lenguajes. Recuerde que este operador proporciona el resto de una división entera. Por ello % no puede aplicarse a un tipo de coma flotante.

## Relacionales y lógicos

La palabra relacional se refiere a la relación entre unos valores y otros. En el término operador lógico, la palabra lógico se refiere a las formas en que esas relaciones pueden conectarse entre sí siguiendo las reglas de la lógica formal. Los discutiremos a la vez, ya que a menudo actúan juntos.

Cabe destacar en este momento que C no tiene valores booleanos. Por lo tanto, verdadero es cualquier valor distinto de o, y falso es o.

A continuación, se muestran los operadores relacionales y lógicos respectivamente:

| OPERADOR | ACCIÓN |
|---|---|
| > | Mayor que |
| >= | Mayor o igual que |
| < | Menor que |
| <= | Menor o igual que |
| == | Igual |
| != | Distinto |

| OPERADOR | ACCIÓN |
|---|---|
| && | AND lógico |
| \|\| | OR lógico |
| ! | NOT lógico |

A la hora de trabajar con estos últimos operadores hay que tener en cuenta sus correspondientes tablas de verdad o tablas de valores de verdad, que se muestran a continuación:

| A | B | A && B |
|---|---|---|
| 1 | 1 | **1** |
| 0 | 1 | **0** |
| 1 | 0 | **0** |
| 0 | 0 | **0** |

| A | B | A \|\| B |
|---|---|---|
| 1 | 1 | **1** |
| 0 | 1 | **1** |
| 1 | 0 | **1** |
| 0 | 0 | **0** |

## Operadores especiales o de asignación

| OPERADOR | ACCIÓN | IMPLEMENTACIÓN |
|---|---|---|
| = | **Asignación** | A=1 |
| ++ | **Incremento** | A++ o A=A+1 |
| -- | **Decremento** | A-- o A=A-1 |
| += | **Incremento en 1 o más** | A+=2 o A=A+2 |
| -= | **Decremento en 1 o más** | A-=2 o A=A-2 |
| *= | **Multiplicación** | A*=2 o A=A*2 |
| /= | **División** | A/=2 o A=A/2 |
| %= | **Resto de la división** | A%=2 o A=A%2 |

El lenguaje C tiene dos operadores muy útiles que no existen generalmente en otros lenguajes de computadora. Son el operador de incremento y el de decremento que están reflejados en el cuadro anterior.

++ Operador incremento, hace que su operando se incremente en uno

-- Operador decremento, hace que el operando se decremente en uno.

Depende del tratamiento, es como se comportan.

Ejemplo:

```
...
++a;//Incrementa la variable, luego la utiliza
a++;//Utiliza la variable, luego la incrementa
--a;//Decrementa la variable, luego la utiliza
a--;//Utiliza la variable, luego la decrementa
...
```

## Precedencia de los operadores

A continuación, se indica la precedencia de los operadores del lenguaje C. Comenzando desde los de mayor precedencia hasta los de menor precedencia.

| Precedencia |
|---|
| () [] -> . |
| ++ -- ! sizeof (tipo)<br>+(unario) -(unario) *(indir.) &(dirección) |
| * / % |
| + - |
| < <= > >= |
| == != |
| && |
| \|\| |
| ?: |
| = += -= *= /= |
| , (operador coma) |

## Sentencias

La sentencias o expresiones describen las acciones algorítmicas que pueden ser ejecutadas, es decir que tras realizar una determinada acción devuelven un resultado.

Las sentencias son una combinación de todos los elementos mencionados anteriormente, siguiendo una lógica específica. Vea los siguientes ejemplos, teniendo en cuenta que dependen del valor que tome la variable.

```
res=a+b;
valor=(a%b)/c;
estado=(a<b)&&(c>2);
```

Se pueden clasificar en dos tipos:

## Sentencias simples

En la práctica, son las más abundantes. Generalmente son asignaciones, invocaciones de funciones o porque no entrada y salida de datos.

## Sentencias compuestas

Las sentencias compuestas, también denominadas bloques, se utilizan en aquellas situaciones en que la sintaxis espera otra sentencia, pero se necesitan usar varias, generalmente están encerradas entre llaves. Profundizaremos en esto más adelante.

## Operaciones de Entrada y salida

Los datos se pueden almacenar en memoria de tres formas diferentes: asociados con constantes, asignados a una variable con una sentencia de asignación o una sentencia de lectura.

La sentencia de lectura es la más indicada si se desea manipular diferentes datos cada vez que se ejecuta el programa. La lectura de datos permite asignar valores desde dispositivos de entrada hasta archivos, (por ejemplo, un teclado o una unidad de disco y se denomina operación de entrada o lectura.

A medida que se realizan cálculos en un programa, se necesitan visualizar los resultados. Esta operación se conoce como operación de salida o escritura.

## Salida por pantalla

La función de "impresión" de información en pantalla utilizada habitualmente es **printf()**. Está disponible dentro de la biblioteca de C y se debe incluir a través de la librería <stdio.h>.

La sintaxis para la función **printf()** es la siguiente:

```
printf("Texto y especificador de formato",Id_de_variable);
```

Ejemplos:

```
printf("Bienvenidos");//Muestra un mensaje de texto
//Muestra un mensaje de texto y un valor
printf("Este es el valor que tiene %d",a);
//Muestra dos valores con sus formatos correspondientes
printf("El valor de a es: %d y el valor de b es: %d",a,b);
```

## Especificadores de formato

Para mostrar valores ya sea números enteros o flotantes, caracteres o cadenas de caracteres se necesitan especificar los formatos. La siguiente tabla muestra algunos de los especificadores de formato básicos.

| TIPO DE DATO | ESPECIFICADOR DE FORMATO | TIPO ESPECÍFICO |
|---|---|---|
| Int | %d | Entero |
| Float | %f | Real |
| Char | %c | Carácter |
| Char [n] | %s | Cadena de caracteres* |

(*)Las cadena de caracteres tiene un tratamiento especial que iremos desarrollando más adelante.

Por ejemplo, si tengo la siguiente declaración de variable con sus respectivas asignaciones int a=5, float b=5.3, Char c='a', la sentencia de salida que se muestra a continuación:

```
printf("Un entero %d\n Un float %f\n Un caracter %c",a,b,c);
```

devuelve el siguiente resultado:

Un entero 5

Un float 5.300000

Un carácter a

Note que se interpreta de izquierda a derecha, cada especificador de formato coincide en la salida con su variable correspondiente.

## Indicadores de formato

Cabe destacar que las especificaciones de formato aceptan una serie de indicadores que pueden alterar su representación en la salida. Algunos ejemplos de estos se muestran a continuación.

La sintaxis para utilizar estos identificadores es:

```
printf("%<Indicador_de_formato><formato>",Id_de_variable)
```

Ejemplo con cadena de caracteres: los comentarios simulan la salida por pantalla, los guiones bajos reemplazan espacios y están de forma figurativa.

```
...
char linea[13]="Programacion";
printf("%s\n",linea)            //Programacion
printf("%15s\n",linea)          //___Programacion
printf("%15.2s\n",linea);       //_____________Pr
printf("%.5s\n",linea);         //Progr__________
printf("%-15s en C\n",linea);   //Programacion___en C
...
```

Ejemplos con números:

```
...
printf ("%6d\n",123);   //___123
printf ("%-6d\n",123);  //123___
printf ("%06d\n",123);  //000123
printf ("%.2f",12.3400); //12.34
...
```

Puede haber más indicadores  los iremos mencionando a medida que los empecemos a utilizar.

## Entrada de datos

La función que se utiliza para ingresar datos es scanf() que también se encuentra dentro de la librería <stdio.h>. Esta función recibe datos que ingresa el usuario y los guarda en una variable creada anteriormente.

Cabe destacar que la declaración de una variable supone la reserva de una zona de memoria es por eso que vamos a necesitar manipular dichas posiciones utilizando algún carácter especial. Profundizaremos en esto cuando veamos otro tipo de variable (punteros).

La función scanf() necesita una dirección de memoria para saber dónde debe guardar un resultado. Como no estamos en una sentencia de asignación, es necesario que obtengamos explícitamente la dirección de memoria anteponiendo el operador &.

También son necesarios los especificadores de formato vistos anteriormente con printf, que indican el tipo de dato que se va a ingresar.

La sintaxis para la función scanf() es la siguiente:

```
...
scanf("Especificador de formato",&Identificador_de_variable);
```

Ejemplos:

```
scanf("%d",&a);//Ingresar un valor de tipo entero
scanf ("%c",&c);//Ingresar un valor de tipo carácter
scanf("%f",&b);//Ingresar un valor de tipo flotante
scanf("%d%d",&a,&b);//Se permite leer múltiples variables
```