Introducción al Lenguaje C

Palabras reservadas

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). Estas palabras sirven para indicar al computador que realice una tarea muy determinada (desde evaluar una comparación, hasta definir el tipo de una variable) y tienen un especial significado para el compilador. El C es un lenguaje muy conciso, con muchas menos palabras clave que otros lenguajes. A continuación se presenta la lista de las 32 palabras clave del ANSI C, para las que más adelante se dará detalle de su significado (algunos compiladores añaden otras palabras clave, propias de cada uno de ellos. Es importante evitarlas 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

Comentarios

El lenguaje C permite que el programador introduzca comentarios en los ficheros fuente que contienen el código de su programa. La misión de los comentarios es servir de explicación o aclaración sobre cómo está hecho el programa, de forma que pueda ser entendido por una persona diferente (o por el propio programador algún tiempo después). Los comentarios son también particularmente útiles (y peligrosos...) cuando el programa forma parte de un examen que el profesor debe corregir. El compilador 2 ignora por completo los comentarios.

Los caracteres (/*) se emplean para iniciar un comentario introducido entre el código del programa; el comentario termina con los caracteres (*/). No se puede introducir un comentario dentro de otro. Todo texto introducido entre los símbolos de comienzo (/*) y final (*/) de comentario son siempre ignorados por el compilador. Por ejemplo:

```
variable_1 = variable_2;   /*En esta línea se asigna a
                             variable_1 el valor
                             contenido en variable_2 */
```

Los comentarios pueden actuar también como separadores de otros tokens propios del lenguaje C. Una fuente frecuente de errores –no especialmente difíciles de detectar– al programar en C, es el olvidarse de cerrar un comentario que se ha abierto previamente.

El lenguaje ANSI C permite también otro tipo de comentarios, tomado del C++. Todo lo que va en cualquier línea del código detrás de la doble barra (//) y hasta el final de la línea, se considera como un comentario y es ignorado por el compilador. Para comentarios cortos, esta forma es más cómoda que la anterior, pues no hay que preocuparse de cerrar el comentario (el fin de línea actúa como cierre). Como contrapartida, si un comentario ocupa varias líneas hay que repetir la doble barra (//) en cada una de las líneas. Con este segundo procedimiento de introducir comentarios, el último ejemplo podría ponerse en la forma:

```
variable_1 = variable_2;   // En esta línea se asigna a
                           // variable_1 el valor
                           // contenido en variable_2
```

1.- Tipos de Datos en ANSI C:

1.1.- Datos Numéricos:

Existen Básicamente dos tipos de datos numéricos en ANSI C, cada uno de ellos subdividido en otros tipos de datos.

- Números Enteros: Existen variaos diferentes tipos de enteros en C y cada uno representa un subconjunto específico del conjunto de los números enteros. Estos tipos se muestran a continuación:
- Números Reales o de Punto Flotante: Un tipo real representa un subconjunto de los números reales, dicho conjunto se representa en notación de punto flotante con un número fijo de dígitos.
  ANSI C provee dos tipos reales predefinidos. Cada tipo tiene un rango de precisión específico:

ANSI C reconoce que el tamaño y rango numérico de los tipos de datos básicos, son de implementación específica y dependen usualmente de la arquitectura del computador anfitrión. En implementaciones como Borland C++, la

plataforma que se utiliza es la de la familia IBM PC y compatibles, así, la arquitectura de los microprocesadores 8088 y 80x86 de Intel gobierna sobre la elección de las representaciones internas de los distintos tipos de datos.
La siguiente tabla ofrece una lista de los distintos tipos de datos numéricos y sus dos tipos de representaciones internas (16 y 32 Bits)

Representación de 16 bits
Enteros

Tipo	Tamaño (bits)	Rango	Aplicaciones Comunes
unsigned char	8	0 a 255	Valores pequeños y todo el conjunto de caracteres
char	8	-128 a 127	Valores pequeños y subconjunto de caracteres
enum	16	-32,768 a 32,767	Conjunto de valores ordenados
unsigned int	16	0 a 65,535	Números Grandes y control de Iteraciones
short int	16	-32,768 a 32,767	Contadores, números pequeños, control de bucles
int	16	-32,768 a 32,767	Contadores, números pequeños, control de bucles
unsigned long	32	0 a 4,294,967,295	Distancias Astronómicas
long	32	-2,147,483,648 a 2,147,483,647	Números grandes, poblaciones
Reales			
float	32	3.4×10^{-38} a 3.4×10^{38}	Científico (7 dígitos de precisión)
double	64	1.7×10^{-308} a 1.7×10^{308}	Científico (15 dígitos de precisión)
long double	80	3.4×10^{-4932} a 1.1×10^{4932}	Financiero (18 dígitos de precisión)

Representación de 32bits
Enteros

Tipo	Tamaño (bits)	Rango	Aplicaciones Comunes
unsigned char	8	0 a 255	Valores pequeños y todo el conjunto de caracteres
char	8	-128 a 127	Valores pequeños y subconjunto de caracteres
short int	16	-32,768 a 32,767	Contadores, números pequeños, control de bucles
unsigned int	32	0 a 4,294,967,295	Números Grandes y control de Iteraciones
int	32	-2,147,483,648 a 2,147,483,647	Contadores, números pequeños, control de bucles
unsigned long	32	0 a 4,294,967,295	Distancias Astronómicas
enum	32	-2,147,483,648 a 2,147,483,647	Conjunto de valores ordenados
long	32	-2,147,483,648 a 2,147,483,647	Números grandes, poblaciones
Reales			
float	32	3.4×10^{-38} a 3.4×10^{38}	Científico (7 dígitos de precisión)
double	64	1.7×10^{-308} a 1.7×10^{308}	Científico (15 dígitos de precisión)
long double	80	3.4×10^{-4932} a 1.1×10^{4932}	Financiero (18 dígitos de precisión)

1.2.- Datos de caracteres y Cadenas de Caracteres:

Una cadena de caracteres es una sucesión ordenada y finita de caracteres.

El tipo de dato *char* posee una longitud de un Byte (8 bits), y se usa para almacenar caracteres. Sólo uno de los 256 caracteres posibles puede almacenarse en un *char* a la vez. Es posible usar este tipo para definir cadenas de caracteres y se puede hacer de dos formas.

- **char** **cadena1, *cadena2, ...;*
  Donde **char** es el tipo y *cadena1, cadena2*, etc, son variables que apuntan a áreas de memoria donde se almacenan cadenas de caracteres de longitud variable.
- **char** *cadena1[n];* ó **char** *cadena1[];*
  Define una cadena de caracteres llamada *cadena1* de longitud *n* en el primer caso, y una cadena de longitud no definida en el segundo.

La diferencia básica entre las dos formas de declarar una cadena de caracteres es que la primera no asegura un espacio de memoria para la cadena, es decir, que es posible que de forma involuntaria se sobrescriba la información almacenada de esta manera.

1.3.- Datos Complejos:

A continuación se presenta una lista de todos los tipos de datos complejos en C.

1.3.1.- Arreglos: Para definir un arreglo (vectores, matrices, cubos, etc.) de algún tipo específico se usa la siguente sintaxis:

Tipo *Arreglo*[n]; /* Arreglo unidimensional (Vector) */

Tipo *Arreglo*[n][m]… ; /* Arreglo multidimensional */

Donde **Tipo** es el identificador para indicar a que tipo de datos pertenece cada elemento del arreglo. *Arreglo* es el nombre de la variable definida por el usuario y [n][m] denotan el número de elementos de cada dimensión del arreglo.
El tipo de elemento puede ser cualquiera, sin embargo, los índices para identificar las posiciones del arreglo deben ser constantes numéricas enteras.

Ejemplo:

```
int Vector[10];          /*Define un vector de 10 posiciones enteras indexadas de 0 a 9 */
double Matriz[10][5];    /*Define una matriz de 10 filas (indexadas de 0 a 9) y 5 filas  */
                         /*(indexadas  de 0 a 4), donde cada elemento es de tipo double  */
```

1.3.2.- Estructuras: Se usa para declarar registros, que son tipos definidos por el usuario que contienen una serie de componentes o campos que pueden ser de cualquier tipo. Se usan generalmente para el trabajo con archivos, control de distintas características de un mismo objeto, etc. La sintaxis para la declaración del tipo se muestra a continuación:

```
struct Registro  {  Tipo Campo1;
                    Tipo Campo2;
                      ...
                    Tipo Campon;
                 } VarTipoRegistro ;
```

Donde **struct** es la palabra reservada para la declaración de estructuras o registros, *Registro* es el nombre de registro definido por el usuario, los caracteres "{" y "}" inicio y el fin de los campos del registro, *Campo1, Campo2, etc*., constituyen la declaración de tipos de cada campo y *VarTipoRegistro*, es una variable declarada como una estructura del tipo *Registro*.

Ejemplo:

```
/* Definición de un registro */
struct Notas {  char Nombre[15], Apellido[15];
                float Nota1, Nota2, Nota3;
             } NotasAlumnos[N_ALUMNOS];
```

1.3.3.- Tipos Definidos: C proporciona una facilidad llamada **typedef** para crear nuevos nombres para tipos definidos de datos como ejemplifica la siguiente declaración:

```
typedef int Longitud;
```

Esto hace del nombre *Longitud* un sinónimo de **int**. El nuevo nombre puede emplearse en declaraciones y los llamados casts. Se debe destacar que **typedef** no crea un nuevo tipo en ningún sentido, simplemente agrega un nuevo nombre para el tipo existente.

1.3.4.- Uniones: Una unión es una variable que puede contener (en momentos diferentes) objetos de diferentes tipos y tamaños.

```
union Nombre   { Tipo1 Var1;
                 Tipo2 Var2;
                   …
                 Tipon Varn;
               } UnionTipoNombre;
```

Donde la palabra union es parte de la declaración; *Nombre* es el nombre que se le dará a la nueva unión; **Tipo1**, **Tipo2**, etc. son los tipos que pueden almacenarse en la union; *Var1*, *Var2*, etc. Son los nombres para cada tipo dados por el usuario y *UnionTipoNombre* es una variable **union** de tipo *Nombre* declarada por el usuario.

Ejemplo:

```
union UPrueba { int Val_Entero;
                float Val_Real;
                char *Val_Cadena;
```

} *MyUnion*;

Para referirse a cada tipo se debe usar el nombre de la variable declarada de tipo unión seguida por un punto y el nombre del identificador de tipo (*Val_Entero, Val_Real, *Val_Cadena*) que se desea mostrar. El programador es el encargado de controlar el último tipo almacenado en la unión para así usar siempre el correcto identificador de tipo.

1.3.5.- Enumeraciones: En C existen una clase especial de constantes, llamadas constantes enumeración. Estas constantes se utilizan para definir los posibles valores de ciertos identificadores o variables que sólo deben poder tomar unos pocos valores. Y la sintaxis es como sigue:

enum *Nombre* {*Valor1*, *Valor2*, …, *Valorn*} *EnumeracionTipoNombre*;

Donde **enum** es parte de la declaración; *Nombre* es el identificador del nuevo tipo, las palabras entre llaves (*Valor1*, *Valor2*, etc.) son los posibles valores que puede tomar una variable definida de este tipo y *EnumeracionTipoNombre* es una variable definida como enumeración de tipo *Nombre*.

Ejemplos:

enum Bool {Falso = 0, Verdadero = 1}; Bandera; /* La variable Bandera es del tipo definido por el usuario Bool y solo puede tomar uno de dos posibles valores Falso (0), o Verdadero (1) */

enum Dia {Lunes, Martes, Miercoles, Jueves, Viernes, Sabado, Domingo} Dia1, Dia2;
// Dia1 y Dia2 solo pueden contener un valor de siete posibles Lunes…Domingo (0…6)

1.3.6.- Apuntadores: Se utiliza para la definición de estructuras dinámicas, ya que es un tipo de dato que puede almacenar una posición de memoria para leer o escribir información en ella. Más adelante se proporcionará una mejor descripción del tipo, cuando se trate el tema de las estructuras dinámicas.

2.- Constantes y Variables:

Una constante es un tipo de dato que mantiene un valor estático a lo largo del programa.

Una variable por otro lado es un tipo de dato, cuyo valor puede cambiar durante la ejecución del programa. Para que una variable pueda ser usada en el programa, ésta debe declararse primero, asignándole un tipo de dato específico, acorde al valor que se espera almacenar en ella. Las variables en C poseen nombres definidos por el usuario y la regla de construcción es la siguiente: Deben estar construidas como una cadena de por lo menos un caracter, sólo se permiten los caracteres alfabéticos (‘a’..’z’, ‘A’..’Z’), numéricos (‘0’..’9’) y el caracter especial para subrayado (_). La cadena debe comenzar en un caracter alfabético seguido por cualquier combinación de caracteres válidos. No hay restricciones en cuanto a longitud, sin embargo, sólo se tomarán en cuenta los primeros 63. Es importante señalar que C hace distinción entre mayúsculas y minúsculas, así, por ejemplo, el programador no puede referirse a una variable como *variable1*, si la declaró como *Variable1*.

3. Estructura de un Programa en Lenguaje C.

Un programa en C no esta sujeto a una estructura fija, sin embargo a continuación se presenta una muy común:

3.1.- Inclusión de librerías:

Lenguaje C tiene un número de palabras reservadas muy limitado en comparación con otros lenguajes, es por eso que un gran numero de funciones, rutinas, tipos predefinidos, macros, constantes etc. Se encuentran declarados en las llamadas librerías de cabecera. Así el programador puede usar alguna de las librerías estándar ofrecida por el proveedor del lenguaje o alguna elaborada por el mismo o un tercero. Las librerías simpre se incluyen al principio del programa y se hace de la siguiente forma.

#include<*NombreDeLibreia*> ó bién **#include**”*NombreDeLibreia*”

Donde el símbolo # le dice al compilador que ese bloque debe ser tratado de forma especial, **incluye** indica que se debe incluir una librería, denotada por *NombreDeLibreia*, la diferencia entre escribir el nombre de la librería entre comillas o los símbolos de desigualdad estriba en el algoritmo de búsqueda que se empleará para encontrar la librería. Usualmente se encierran entre comillas las librerías desarrolladas por el usuario y entre los símbolos (< y >) las librerías estándar.

Algunas librerías de interés son:

- conio.h: Alberga rutinas para operaciones de entrada y salida de datos.
- dos.h: Define algunas constantes y rutinas propias de MS_DOS.
- Math.h: Contiene constantes y funciones matemáticas comunes.
- stdio.h: Similar a conio.h
- stdlib.h: Guarda algunas rutinas de conversión comunes, así como algunas rutinas de ordenación y búsqueda.
- String.h: Ofrece rutinas para la manipulación de cadenas de caracteres y memoria.

Ejemplo

```
#include<conio.h>
#include<stdlib.h>
#include"MiLib.h"
```

3.2.- Otras Directivas de Compilación:

#ifdef, #define, #ifndef, #else, #endif, #undef, #if, entre otras, al igual que #include, le indicant al compilador que se deben tartar de forma especial. Ejemplo.

```
#define MAXLINEAS 25
```

le indica al compilador que debe sustituir el identificador MAXLINEAS por el número 25, cada vez que aparezca en el programa, excepto cuando es una constante literal de cadena, es decir, que se encuentre entre comillas.

3.3.- Declaraciónes:

Aquí suelen colocarse todas las declaraciones de constantes, tipos y variables locales.

Una declaración de variables, asocia los identificadores definidos por el usuario con los tipos predefinidos C o bien con los tipo definidos por el usuario. Mediante una declaración de variable, se reserva una locación de memoria para almacenar datos del tipo definido.

Sintaxis:

Tipo *Variable1, Variable2, …* ;
...

Donde: **Tipo** es la palabra reservada que denota el tipo al cual pertenece la lista de variables, *Variable1, etc.*, son los nombres de las variables definidas por el usuario,

Ejemplo:

```
// Declaración de Variables
long double X, Y, Z;
int I, J, K;
enum Sex { Masculino, Femenino};
float Arreglo[10];
struct Ficha {  char Nombre[15], Cedula[12];
                Sex Sexo;
             } Fichero[15];
```

Nota: toda variable declarada a este nivel tiene carácter global.

3.4.- Procedimientos:

Un procedimiento es una parte del programa que especifica una acción, a menudo se basa en un conjunto de parámetros. Aunque en C no existen los procedimientos, si hay funciones que según su definición pueden comportarse como tales.

Sintaxis:

void *Nombre* (**Tipo** *Prametro1,***Tipo** *Parametro2,…,* **Tipo** *Parámetron*)
{ *Declaración_de_locales;*
Instrucción;
Instrucción;
}

Donde **void** denota el tipo que se devolverá (sin valor), *Nombre* es el identificador para el procedimiento definido por el usuario, *Parametro1, etc.*, denotan los parámetros del procedimiento (si existen), y **Tipo**, el tipo al cual pertenecen los parámetros. Es de hacer notar que aunque el procedimiento no involucre parámetros, se deben escribir los parentesis.

Ejemplo:

```
/* Declaración del Procedimiento */
void EscribirEn(int X, int Y,  char S[])
   {  gotoxy(X,Y);              // Pociciona el cursor en la columna X, fila Y de la pantalla
      cprintf("%s",S);          // Escribe la cadena contenida en la variable S
   }
```

3.5.- Funciones:

Es una parte del programa que calcula y retorna un valor. Su estructura es muy parecida a la de los procedimientos, sin embargo, es preciso que en su declaración se indique el tipo de valor que será devuelto. Una función al igual que los procedimientos puede o no poseer parámetros.

Sintaxis:

TipoDevuelto *Nombre* (**Tipo** *Prametro1,***Tipo** *Parametro2,…,* **Tipo** *Parámetron*)

```
{ Declaración_de_locales;
  Instrucción;
  Instrucción;
  return Valor;
}
```

Donde **TipoDevuelto** es el tipo de dato que retornará la función, *Nombre* es el identificador con el cual se definirá la función por parte del usuario, *Parametros,* es la declaración de los parámetros (si los hay) y **Tipo** denota el tipo de valor de cada parámetro.

Cualquiera de los tipos existentes en C o de los definidos por el usuario son validos para declara funciones.

Ejemplo:

```
char *UpCaseStr(char S[])                  /* Declaración de la Función */
   {  int i;
      char *UpCase="";
      for (i = 0; i<strlen(S); i++)        // incrementa i de uno en uno desde 0 hasta la longitud de S
         if (S[i] >= 'a' && S[i] <= 'z' )  // si el character en la posicion i es una letra minúscula
            UpCase[i] = S[i] + 'A' - 'a';  // asigna la mayúscula del character a una cadena auxiliar
         else                              // de lo contrario
            UpCase[i] = S[i];              // asigna el carácter sin modificar a la cadena auxiliar
      UpCase[i]='\0';                      // añade el carácter de fin de cadena a la cadena auxiliar
      return UpCase;                       // devuelve el contenido de la cadena auxiliar
   }
```

3.6.- Función Principal main():

Está constituido por las instrucciones que efectivamente serán ejecutadas durante la corrida del programa. Estas instrucciones deben terminar siempre con un símbolo punto y coma (;) y las hay de distintos tipos, desde una simple asignación, hasta la llamada a una función. Todas las instrucciones que conforman el programa principal deben estar encerradas entre llaves {} para marcar el inicio y el final respectivamente.

Ejemplo:

```
#include<lib.h>                 //Inclusión de Librerías
…                               //Otras directivas de compilación
DeclaracionesGlobales           //Declaración de variables globales

void main(void)                 //Función principal (las palabras void, indican que la función no recibe ni
  {                             //devuelve ningún valor.
    clrscr();                   //Instrucción: Limpiar la pantalla.
    printf("Hola mundo");       //Instrucción: Escribir "Hola mundo"
    getch();                    //Instrucción: Esperar a que se presione una tecla
  }                             //Fin del Programa
```

4.- Expresiones

Son combinaciones de constantes, variables, símbolos, operadores, paréntesis y nombres de funciones o procedimientos.

4.1.- Expresiones Aritméticas:

Son Análogas a las fórmulas matemáticas, las variables y constantes son numéricas así como los valores devueltos por las funciones; todas las operaciones son aritméticas.

Operadores aritméticos

Operador	Significado	Tipo de Operandos	Tipo de Resultado
+	Suma	Entero o Real	Entero o Real
-	Resta	Entero o Real	Entero o Real
*	Multiplicación	Entero o Real	Entero o Real
/	División	Entero o Real	Entero o Real
%	Residuo de la División Entera	Entero	Entero

Ejemplos de Expresiones aritméticas:

```
16 /  x                  //x = 2
(32%2+16-234)/0.5
2*PI/2                   //PI = 3,14…..}
```

Reglas de Prioridad para los operadores.

Las expresiones con más de dos operandos, requieren de reglas matemáticas que aseguren su correcta interpretación. Por Ejemplo: 3 * 4 + 5 Puede ser interpretada como: (3 * 4) + 5 = 17 ó bien 3 * (4 +5) = 27

Para evitar confusión, C Sigue las siguientes normas de prioridad.

- Operaciones entre paréntesis, se evalúan primero los más internos.
- Operadores (*, /)
- Operadores (%)
- Operadores (+, -)
- En caso de operadores de igual prioridad se evalúa la expresión de izquierda a derecha.

Ejemplo: 2 * 2 + 3 * 3 + 6 / 2 % 2 = (2*2) + (3*3) + ((6 / 2) % 2)

Operadores Incrementales.

Los *operadores incrementales* (++) y (--) son operadores unarios que incrementan o disminuyen *en una unidad* el valor de la variable a la que afectan. Estos operadores pueden ir inmediatamente delante o detrás de la variable. Si preceden a la variable, ésta es incrementada antes de que el valor de dicha variable sea utilizado en la expresión en la que aparece. Si es la variable la que precede al operador, la variable es incrementada después de ser utilizada en la expresión. A continuación se presenta un ejemplo de estos operadores:

```
i = 2;
j = 2;
m = i++; // después de ejecutarse esta sentencia m=2 e i=3
n = ++j; // después de ejecutarse esta sentencia n=3 y j=3
```

Estos operadores son muy utilizados. Es importante entender muy bien por qué los resultados **m** y **n** del ejemplo anterior son diferentes.

4.2.- Expresiones Lógicas:

Su valor es siempre Verdadero o Falso (1 ó 0 respectivamente) y los operadores son (&& (conector y), || (conector ó) y ! (negación)), además de los operadores relacionales son: (==, <, >, >=, <=, !=) , igual a, menor que, mayor que, mayor o igual a, menor o igual a y distinto a, respectivamente. Es de hacer notar que estos operadores no son aplicables a cualquier tipo de dato en C.

4.3.- Algunas Funciones Matemáticas:

C no posee funciones matemáticas predefinidas, sin embargo, la librería “math.h” ofrece muchas de las más comunes. Algunas de ellas son:

Función	Descripción	Argumento	Resultado
abs(x)	Valor Absoluto se (x)	Entero o Real	Entero o Real
sin(x)	Seno de (x)	Entero o Real	Real

cos(x)	Coseno de (x)	Entero o Real	Real
asin(x)	Arco Seno de (x)	Real	Real
acos(x)	Arco Coseno de (x)	Real	Real
atan(x)	Arco tangente de (x)	Real	Real
exp(x)	Exponencial de (x)	Real o Entero	Real
log(x)	Logaritmo natural de (x)	Real o Entero	Real
log10(x)	Logaritmo base 10 de (x)	Real o Entero	Real
ceil(x)	Aproxima x al entero inmediato superior	Real	Entero
pow(x,n)	x elevado a la n	Real o Entero	Real o Entero
sqrt(x)	Raíz Cuadrada de (x)	Real o Entero	Real

Estas funciones operan con valores de doble precisión (double), para obtener mayor precisión deben usarse las funciones análogas nombradas de la misma forma pero terminadas en ele (l), ejemplo absl(x), que operan con tipos long double.

4.4.- El operador de Asignación:

La forma de darle valores a las variables en C es a través de la asignación. El operador de asignación es el símbolo de la igualdad (=). Una operación de asignación también es conocida como instrucción de asignación y es una de las más importantes dentro de todo lenguaje.

Sintaxis para la instrucción de asignación:

Variable = *Expresión*;

Donde *Variable* es el nombre de la variable en la cual se espera almacenar el valor obtenido al evaluar *Expresión*. Es muy importante que el tipo al cual pertenece la variable sea el mismo que el del valor de retorno de la expresión.

Ejemplo:

```
A = 10;                 /*Se le asigna el valor 10 a la variable A*/
```

En C, así como en la mayoría de los lenguajes de programación, el operador de asignación es del tipo destructivo, pues al ser usado sobre una variable, ésta pierde el valor que poseía, para adoptar el nuevo.

Ejemplo:

```
A = 5;
A = 10;          /*A pierde su anterior valor 5 y toma el valor 10*/
```

En toda instrucción de asignación, primero se evalúa el lado derecho del operador antes de ser asignado a la variable.

Ejemplo:

```
A = 15 * 3 + 2          /*se evalúa (15*3)+2 = 47, y finalmente se almacena este valor dentro de la variable A*/
```

En Expresiones del tipo (A = A + 1;), donde la variable objeto de la asignación aparece a ambos lados del operador de asignación, la expresión de la derecha se evalúa tomando para ello el valor anterior de la variable A, y luego se almacena el resultado en A.

Ejemplo:

```
A = 15;
A = A / 2;          /*Se evalúa (15 / 2) = 7.5, y se almacena en A*/
```

5.- Entrada y Salida de Información

A diferencia de otros lenguajes, *C no dispone de sentencias de entrada/salid*a. En su lugar se utilizan funciones contenidas en la librería estándar y que forman parte integrante del lenguaje.

Las funciones de entrada/salida (Input/Output) son un conjunto de funciones, incluidas con el compilador, que permiten a un programa recibir y enviar datos al exterior. Para su utilización es necesario incluir, al comienzo del programa, el archivo **stdio.h** en el que están definidos sus prototipos:

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

donde ***stdio*** proviene de ***standard-input-output***.

5.1 Función printf()

La función printf() imprime en la unidad de salida (el monitor, por defecto), el texto, y las constantes y variables que se indiquen. La forma general de esta función se puede estudiar viendo su prototipo:

```
int printf("cadena_de_control", tipo arg1, tipo arg2, ...)
```

Explicación: La función printf() imprime el texto contenido en cadena_de_control junto con el valor de los otros argumentos, de acuerdo con los formatos incluidos en cadena_de_control. Los puntos suspensivos (...) indican que puede haber un número variable de argumentos. Cada formato comienza con el carácter (%) y termina con un carácter de conversión.
Considérese el ejemplo siguiente,

```
int i;
double tiempo;
float masa;
printf("Resultado nº: %d. En el instante %lf la masa vale %f\n", i, tiempo, masa);
```

en el que se imprimen 3 variables (i, tiempo y masa) con los formatos (%d, %lf y %f), correspondientes a los tipos (int, double y float), respectivamente. La cadena de control se imprime con el valor de cada variable intercalado en el lugar del formato correspondiente.
Lo importante es considerar que debe haber correspondencia uno a uno (el 1º con el 1º, el 2º con el 2º, etc.) entre los formatos que aparecen en la cadena_de_control y los otros argumentos (constantes, variables o expresiones). Entre el carácter % y el carácter de conversión puede haber, por el siguiente orden, uno o varios de los elementos que a continuación se indican:
- Un número entero positivo, que indica la anchura mínima del campo en caracteres.
- Un signo (-), que indica alineamiento por la izda (el defecto es por la dcha).
- Un punto (.), que separa la anchura de la precisión.
- Un número entero positivo, la precisión, que es el nº máximo de caracteres a imprimir en un string, el nº de decimales de un float o double, o las cifras mínimas de un int o long.
-Un cualificador: una (h) para short o una (l) para long y double

Los caracteres de conversión más usuales se muestran a continuación

Carácter	Tipo de argumento	Carácter	Tipo de argumento
d, i	int decimal	o	octal unsigned
u	int unsigned	x, X	hex. unsigned
c	Char	s	cadena de char
f	float notación decimal	e, g	float not. científ. o breve
p	puntero (void *)		

Ejemplos

```
printf("Con cien cañones por banda,\nviento en popa a toda vela,\n");

printf("%s\n%s\n%s\n%s\n",
"Con cien cañones por banda,",
"viento en popa a toda vela,",
"no cruza el mar sino vuela,",
"un velero bergantín.");
```

5.2 Función scanf()
La función scanf() es análoga en muchos aspectos a printf(), y se utiliza para leer datos de la entrada estándar (que por defecto es el teclado). La forma general de esta función es la siguiente:

```
int scanf("%x1%x2...", &arg1, &arg2, ...);
```

donde x1, x2, ... son los caracteres de conversión, mostrados más adelante en este documento, que representan los formatos con los que se espera encontrar los datos. La función scanf() devuelve como valor de retorno el número de conversiones de formato realizadas con éxito. La cadena de control de scanf() puede contener caracteres además de formatos. Dichos caracteres se utilizan para tratar de detectar la presencia de caracteres idénticos en la entrada por teclado. Si lo que se desea es leer variables numéricas, esta posibilidad tiene escaso interés. A veces hay que comenzar la cadena de control con un espacio en blanco para que la conversión de formatos se realice correctamente.
En la función scanf() los argumentos que siguen a la cadena_de_control deben ser pasados por referencia, ya que la función los lee y tiene que trasmitirlos al programa que la ha llamado. Para ello, dichos argumentos deben estar

constituidos por las direcciones de las variables en las que hay que depositar los datos, y no por las propias variables. Una excepción son las cadenas de caracteres, cuyo nombre es ya de por sí una dirección (un puntero), y por tanto no debe ir precedido por el operador (&) en la llamada.

Los caracteres de conversión más usuales para la función scanf() se muestran a continuación

Carácter	caracteres leídos	argumento
c	cualquier carácter	char *
d, i	entero decimal con signo	int *
u	entero decimal sin signo	unsigned int
o	entero octal	unsigned int
x, X	entero hexadecimal	unsigned int
e, E, f, g, G	número de punto flotante	float
s	cadena de caracteres sin ' '	char
h, l	para short, long y double	
L	modificador para long double	

Por ejemplo, para leer los valores de dos variables int y double y de una cadena de caracteres, se utilizarían la sentencia:

```
int n;
double distancia;
char nombre[20];
scanf("%d%lf%s", &n, &distancia, nombre);
```

en la que se establece una correspondencia entre n y %d, entre distancia y %lf, y entre nombre y %s. Obsérvese que nombre no va precedido por el operador (&). La lectura de cadenas de caracteres se detiene en cuanto se encuentra un espacio en blanco, por lo que para leer una línea completa con varias palabras hay que utilizar otras técnicas diferentes.

En los formatos de la cadena de control de scanf() pueden introducirse corchetes [...], que se utilizan como sigue. La sentencia,

```
scanf("%[AB \n\t]", s);      // se leen solo los caracteres indicados
```

lee caracteres hasta que encuentra uno diferente de ('A','B',' ','\n','\t'). En otras palabras, se leen sólo los caracteres que aparecen en el corchete. Cuando se encuentra un carácter distinto de éstos se detiene la lectura y se devuelve el control al programa que llamó a scanf(). Si los corchetes contienen un carácter (^), se leen todos los caracteres distintos de los caracteres que se encuentran dentro de los corchetes a continuación del (^). Por ejemplo, la sentencia,

```
scanf(" %[^\n]", s);
```

lee todos los caracteres que encuentra hasta que llega al carácter nueva línea '\n'. Esta sentencia puede utilizarse por tanto para leer líneas completas, con blancos incluidos. Recuérdese que con el formato %s la lectura se detiene al llegar al primer delimitador (carácter blanco, tabulador o nueva línea).

6.- Instrucciones en C

Una instrucción es una orden que se da al computador para que lleve a cabo algún trabajo. En un lenguaje de programación como C, las instrucciones se ejecutan una tras otra siguiendo el orden lógico de arriba hacia abajo, comenzando a partir de la llave de apertura "{" que se encuentra inmediatamente después de la función principal "main" que debe estar al inicio del programa principal, hasta la llave que cierra "}" situada al final. Sin embargo, es posible romper con este orden lógico mediante el uso de una instrucción de salto. Es preciso mencionar que cada instrucción en C debe culminarse con la escritura del símbolo (;).

En C existen varios tipos de instrucciones que se pueden usar para escribir programas. Entre ellas se encuentran las ya mencionadas instrucciones de asignación y de entrada y salida de datos. Además de éstas se encuentran las siguientes.

6.1.- Instrucciones de repetición o bucles: C cuenta con tres tipos de instrucciones para la repetición de una secuencia de instrucciones. Ellas son:

while: Repite un conjunto de instrucciones un número indeterminado de veces mientras una condición se cumpla.
Sintaxis:

```
while (Condición)
  {
    Instrucción;
```

```
    Instrucción;
        :
}
```

Donde **while**, es parte de la instrucción. *Instrucción* representa el conjunto de instrucciones que se desean repetir, y *Condición* es cualquier expresión lógica, que puede involucrar en su interior cualquiera de los otros tipos de expresiones. La condición de parada se evalúa antes de ejecutar cualquier instrucción dentro del while en cada iteración, así, si la expresión es falsa el flujo del programa salta hasta la instrucción que se encuentra inmediatamente después de la última instrucción del while.

Ejemplo:

```
/*Determina la longitud de una cadena de caracteres (string)*/
int Longitud(char Cadena[])
  { int Cont = 0;
    while  (Cadena[Cont]¡='\0')     /*Compara cada caracter de la cadena para buscar el final*/
      Cont++;                       /*Cont=Cont+1*/
    return Cont;
  }
```

do while: Al igual que la instrucción anterior, repite un conjunto de instrucciones un número indeterminado de veces mientras una condición se cumpla, con la diferencia que al usar "do while", se garantiza que el bloque de instrucciones contenidos en la instrucción, se ejecuten al menos una vez, pues la condición se evalúa al final y no al principio como ocurre con "while" . Su sintaxis es la siguiente:

```
do
  {
    Instrucción;
    Instrucción;
        :
  } while (Condición);
```

Ejemplo:

```
/*almacena el valor entero equivalente de una cadena de caracteres S compuesta por dígitos,
  en la variable n*/
n = 0;
i = 0;
do { n = 10*n + (S[i]- '0') ;
     i++;
   } while(S[i]>='0' && S[i]<='9');
```

for: La instrucción for o para como se traduce al español, es una de las instrucciones más poderosas que presente lenguaje alguno aunque su funcionamiento es similar a la instrucción while. A continuación se presenta su sintaxis.
Sintaxis:

```
for( Ininicializaciones; Expresión de Control; Actualizaciones)
  {  Instruction;
        :
  }
```

Donde **for** forma parte de la instrucción, las secciones entre paréntesis separadas por los símbolos (;) denotan las distintas partes de la sentencia:
- *Ininicializaciones:* En este bloque pueden colocarse instrucciones que se realizaran sólo al principio de la instrucción y suele usarse para inicializar variables.
- *Expresión de Control:* Es una expresión lógica que se evalúa al inicio de cada ciclo y de resultar falsa se continua en la instrucción ubicada después de la llave que denota la finalización del bucle "}"; pero de cer cierta se procede a ejecutar el cuerpo del **for** que son las *Instrucciones* ubicadas entre las llaves.
- *Actualizaciones:* Son instrucciones que se ejecutan al final de cada iteración, justo antes de evaluar la *Expresión de Control* y determinar si se realizará un nuevo ciclo.

Otra forma de explicar el funcionamiento de la instrucción **for** es mediante una instrucción while

```
inicializacion;
while (expresion_de_control)
  {  Instruccione;
     actualizacion;
  }
```

Téngase en cuenta que los cuerpos ubicados entre los paréntesis de la sentencia **for** pueden estar formados cada uno por una, varias o ninguna instrucción o expresión, separadas entre si mediante comas (,).

Ejemplos:

```
// Calcula la suma de todos los enteros consecutivos desde el número uno hasta el diez
for( Acumulador = 0, i= 1; i<=10; i++)
   Acumulador = Acumulador + i;        //i toma valores crecientes desde uno hasta diez

//Calcula el factorial de un número almacenado en la variable num
for (Factorial=num, k = num-1; k>1; k--)
   Factorial = Factorial*k;
```

Nota: en los dos ejemplos se omitió el uso de las llaves pues el cuerpo de la instrucción sólo involucró una sentencia

6.2.- Instrucciones Selectivas: En C existen dos tipos de instrucciones selectivas o de decisión, que pueden incluir saltos en el flujo del programa dependiendo del cumplimiento o no cumplimiento, de una condición.

Las instrucciones selectivas en C son:

if … else: Con este tipo de instrucción, se especifican las condiciones bajo las cuales una o un grupo de instrucciones pueden ser ejecutadas.

Sintaxis:

```
if(Condición)                    o bien        if(Condición)
   {  Instrucción;                               {  Instrucción;
            :                                             :
   }                                             }
                                                 else
                                                   {  Instrucción;
                                                            :
                                                   }
```

Donde: **if** y **else** son parte de la instrucción. Las llaves {} enmarcan el bloque de *Instrucciones* que se deben ejecutar si se cumple la *Condición* especificada y sólo son necesarios si se espera que se ejecute mas de una instrucción dentro del if. Else (de lo contrario) denota el bloque de instrucciones que deben ejecutarse de no cumplirse la *Condición*.

Ejemplo:

```
if (I < Min || I > Max)
   I = 0;
if (num % 2==0)
   printf("%d es un número par \n",num);
else
   {  fprint("%d es un número impar \nintrodusca un nuevo valor", num);
      scanf("%d",num);
   }
```

switch: La instrucción switch, consta de una expresión que es evaluada y según su resultado o valor, se ejecuta uno u otro bloque de instrucciones, la instrucción a ser evaluada debe ser entera.

Sintaxis:

```
switch (Expresión)
  {  case expression_constante1  : Instrucciones; [break;]
     case expression_constante2  : Instrucciones; [break;]
       …
     default : Instrucciones;
  }
```

Donde **switch, case, default, s**on palabras reservadas que forman parte de la instrucción. La sentencia **switch** funciona de la siguiente forma: *Expresión* es evaluada, luego se compara con cada *expresión_constante* ubicada al lado de un **case** y en caso de resultar iguales se procede a ejecutar el bloque de *instrucciones* correspondientes. Para asegurar que una vez que se encuentra una expresión constante concurrente, las demás no sean evaluadas, se suele escribir la palabra reservada **break** al final del bloque de instrucciones.

Si ninguna de las expresiones constantes satisface la condición de igualdad, se ejecutan las instrucciones correspondientes a la palabra reservada **default.**

Ejemplo:

```
printf("Presione 1, 2 ó 3");
Caracter = getch();
switch (Caracter)
   {  case '1':    printf("Ha presionado la tecla 1"); break;
      case '2':    printf("Ha presionado la tecla 2"); break;
      case '3':    printf("Ha presionado la tecla 3"); break;
      default:     printf("Tecla no válida");
   }
```