CAPITULO 2: VARIABLES Y CONSTANTES

1. TIPOS DE DATOS.

Existen cinco tipos de datos atómicos en C; **char**: carácter, **int**: entero, **float**: real coma flotante, **double:** real de doble precisión y **void**: sin valor.

El tipo **void** se usa para declarar funciones que no devuelven ningún valor o para declarar algún puntero genérico.

Modificadores de tipos

Los modificadores de tipos se usan para alterar el tipo base: (**short, long, unsigned, signed**)

Los todos los modificadores se pueden aplicar a los tipos base **int** y **char,** (llamodos los tipos enteros), sin embargo, **long** también se puede aplicar a **double.**

El uso de **signed** con enteros es redundante debido a que la declaración implícita de los enteros asume números con signo. Su uso más importante es para modificar el tipo **char** en implementaciones en las que char no tiene signo.

La siguiente tabla muestra todas las combinaciones que se ajustan al estándar ANSI de C.

tipo	bytes	v. mínimo	v. máximo		tipo	bytes	v.mínimo	v. máximo
char	1	-127	+127		long int	4	-2.147.483.648	+2.147.483.647
int	2	-32.767	+32.767		signed long	4	-2.147.483.648	+2.147.483.647
signed char	1	-127	+127		unsigned long	4	0	4.294.967.295
signed int	2	-32.768	+32.767		float	4	3.4e-38	3.4e+38
unsigned char	1	0	+255		double	8	1.7e-308	1.7e+308
unsigned int	2	0	65.535		long double	10	3.4e-4932	3.4e+4932
short int	2	-32.768	+32.767					

2. VARIABLES

Una variable es una posición de memoria con nombre que se usa para mantener un valor que puede ser modificado por el programa. Todas las variables en C han de ser declaradas antes de ser usadas y recordar que una variable no tiene nada que ver con su tipo.

Sintaxis: tipo lista_de_variables;

```
int i,j,k;                    Declaración de 3 variables enteras y dos
float beneficio, perdida;     variables reales.
```

Donde se declaran las variables.

Existen tres sitios donde se pueden declarar variables: dentro de las funciones "**variables locales**", en la definición de parámetros de funciones "**parámetros formales**" y fuera de todas las funciones "**variables globales**".

Variables locales.

Llamadas también variables automáticas y se declaran dentro de las funciones y son referenciadas sólo por sentencias que estén dentro del bloque donde están declaradas. Recordar que un bloque está encerrado entre dos llaves. Las variables locales existen sólo cuando se ejecuta el bloque y se destruye al salir de él.

```
void func1(void)
{
int x=10;                                  1)   la variable x declarada
}                                               en fun1()no es la misma
void func2(void)                                que la declarada en
{                                               fun2()
int x =-20;
  {       /* fragmento del prog. */      2)   La variable y declarada
  int y;  /* otra variable      */             en fun2() se crea sólo
  y=2;                                          al entrar en el
  }                                             fragmento del programa y
}                                               desaparece al salir de
                                                él
```

Parámetros formales

Los parámetros formales son los argumentos de una función. Su declaración se hace entre los paréntesis de la función. Su comportamiento es igual que las variables locales de cualquier función; o sea desaparecen al terminar su ejecución.

```
float mul(float x, float y)
{
return(x*y);
}
```

Los parámetros formales de la función mul() son x e y.

Variables globales

A diferencia de las variables locales, las globales se conocen y se pueden usar en cualquier parte del programa (por lo menos de un solo archivo). Además mantienen su valor a lo largo de la ejecución del programa.

Las variables globales se crean justo después de declaración, y por lo general, se hace al principio del programa y fuera de todas las funciones.

Se puede declarar una variable local a un función con el mismo nombre de una variable global existente, pero el compilador las trata distintamente; Dentro de esa función, prevalece la variable local.

```
int x;    /* x es global    */
void fun1(....)
{
y=x;  x = 10;
}
void fun2(....)
{
int x;  /*x es local a fun2 */
x=3;
}
```

1) La variable x declarada fuera de todas las funciones

2) En la función fun1(), se accede a

3) En la función fun2(), x se refiere a la local

Especificadores de almacenamiento

Existen cuatro tipos de clase de almacenamiento: **extern, static, register, auto.** Estos especificadores indican al compilador cómo debe almacenar las variables. La forma de declaración es:

especificador tipo nombre_variable;

Variables externas

Si se quiere usar una variable global en un archivo, pero declarada en otro archivo, debemos referenciarla usando la palabra **extern**. La palabra extern enseña al compilador que las

variables que siguen ya han sido declaradas en alguna otro parte, y por lo tanto no se crea una nueva variable.

Archivo1:	Archivo2:	

`int x, y;`
`char c;`
`.......`
 	
`fun1()`
`{`
`x=10;`
`}`
	`extern int x,y;`
	`extern char c;`
	`.....`
	
	`fun11()`
	`{`
	`x=y;`
	`}`
		Las variables x, y, c declaradas en el archivo2 son exactamente las variables declaradas en el archivo1

Variable estáticas.

Las variables modificadas por la palabra **static** mantienen su valor dentro de su ámbito; (bloque, función o archivo).

Cuando se aplica **static** para una variable local, su valor se retiene entre llamadas de funciones y se crea un almacenamiento permanente igual que las variable globales (en segmento de datos) aunque sigue desconocida fuera de su ámbito.

Cuando se aplica **static** para una variable global, se indica al compilador que cree una variable conocida únicamente en el archivo. Ninguna rutina de otros archivos puede tener acceso a ella. Esto nos permite una tremenda ventaja de poder ocultar datos en un programa.

```
serie1(void)
{
static int n;
n=n+23;
}

serie2(void)
{
static int n=10;
n=n+23;
}
```

1) Cada llamada a la función serie1() produce un nuevo valor n, basándose en el numero anterior.

2) En la segunda función serie2(), el valor de n se inicializa a 10, pero no cada vez que se llame a la función serie2(),(como es el caso de las variables locales).

Variable registro

El especificador **register**, pide al compilador que almacene la variable correspondiente en los registros internos de la CPU. Con esta modificación, se pretende ganar rapidez de acceso a dichas variable.

Se aplica únicamente para las variable de alcance local y nunca para una variable global y se suele usar para almacenar los contadores de los bucles (**for, while,** etc).

Inicialización de variables

Todas las variables se pueden inicializar en el momento de su declaración. Las variables globales, se inicializan a cero sino se especifica otro valor y las variables locales tendrán valores desconocidos antes de que se lleve a cabo su primera asignación.

```
int global_1
float global_2=100;
.....
main()
{
float local_1=0, local_2;
char c='a';
...
}
```

1) La variable global_1 se inicializa automáticamente a cero y global_2 se ha inicializado a 100.

2) La variable local_1 se ha inicializado a 0, mientras que la segunda contiene cualquier valor. (Garbage)

3. CONSTANTES

A fin de tener control sobre el tipo de las constantes, se aplican la siguientes reglas :

Una variable expresada como entera (sin parte decimal) es tomada como tal salvo que se la siga de las letras F ó L (mayúsculas ó minúsculas) :

1 : tomada como **int.**
1F : tomada como **float.**
1L : tomada como **long int.**

Una variable con parte decimal es tomada siempre como **double**, salvo que se la siga de la letra F ó L .

1.2 : tomada como **double**.
1.2F : tomada como **float.**
1.2L : tomada como **long double.**

Si en los casos anteriores agregamos la letra U ó u la constante queda calificada como **unsigned.**

1u : tomada como **unsigend int.**
1UL : tomada como **unsigned long int.**

Una variable numérica que comienza con "**0**" es tomada como **Octal**

012 equivale a 10 unidades decimales

Una variable numérica que comienza con "**0x**" ó "**0X**" es tomada como **Hexadecimal.**

0x16 equivale a 22 unidades decimales

Constantes simbólicas

Se puede asignar un símbolo a cada constante, y reemplazarla a lo largo del programa por el mismo, de forma que este sea más legible y además, en caso de querer modificar el valor, bastará con cambiarlo en la asignación.

El compilador, en el momento de crear el ejecutable, reemplazará el símbolo por el valor asignado.
Para dar un símbolo a una constante bastará, en cualquier lugar del programa (previo a su uso) poner la directiva: (**#define**)

Las constantes de tipo **char** se representan entre comillas simples y las constantes de cadena de texto "strings" se delimitan entre comillas dobles.

```
#define  UNO     1          1) Define una constante entera.
#define  UNOS    0xffff     2) Define una constante en hexadecimal.
#define  PI      3.14F      3) Define una constante float.
#define  A       'A'        4) Define una constante carácter.
#define  MENSAJE "Hola"     5) Define una constante cadena de carácter
```

Secuencias de escape

Existen una serie de caracteres que no son imprimibles, en otras palabras que cuando editemos nuestro programa fuente (archivo de texto) nos resultará difícil de asignarlas a una variable ya que el editor las toma como un COMANDO y no como un carácter. Un caso típico sería el de "nueva línea" ó ENTER.
El carácter de nueva línea se representa como **\n**

Tabla de secuencias de escape.

CODIGO	SIGNIFICADO	CODIGO	SIGNIFICADO
\n	Nueva línea	\'	Comilla simple
\r	Retorno de carro	\"	Comillas dobles
\f	Nueva página	\\	Barra
\t	Tabulador horizontal	\?'	Interrogación
\v	Tabulador vertical	\a	Sonido(Alerta)
\b	Retroceso (backspace)	\0	Nulo, (se usa para terminar cualquier cadena string)

```
printf("\t Pulsar la tecla \" Return \".  \n"

El mensaje que saldría en pantalla es el siguiente:
C:\     Pulsar la tecla "Return".
C:\ _
```