Operadores y Expresiones

CAPITULO 3: OPERADORES

1. INTRODUCCION

Si analizamos la sentencia siguiente:
var1 = var2 + var3;
estamos diciéndole al programa, por medio del operador +, que compute la suma del valor de dos variables , y una vez realizado ésto asigne el resultado a otra variable var1. Esta última operación (asignación) se indica mediante otro operador, el signo =.
El lenguaje C tiene una amplia variedad de operadores, y todos ellos caen dentro de 6 categorias , a saber : aritméticos , relacionales, lógicos, incremento y decremento, manejo de bits y asignacion. Todos ellos se irán describiendo en los párrafos subsiguientes.

2. OPERADORES ARITMETICOS

Tal como era de esperarse los operadores aritméticos ,mostrados en la TABLA 4 , comprenden las cuatro operaciones basicas , suma , resta , multiplicación y división , con un agregado , el operador módulo .

TABLA 4 OPERADORES ARITMETICOS

SIMBOLO	DESCRIPCION	EJEMPLO	ORDEN DE EVALUACION
+	SUMA	a + b	3
-	RESTA	a - b	3
*	MULTIPLICACION	a * b	2
/	DIVISION	a / b	2
%	MODULO	a % b	2
-	SIGNO	-a	2

El operador módulo (%) se utiliza para calcular el resto del cociente entre dos ENTEROS , y NO puede ser aplicado a variables del tipo float ó double .
Si bien la precedencia (orden en el que son ejecutados los operadores) se analizará más adelante, en este capítulo, podemos adelantar algo sobre el orden que se realizan las operaciones aritméticas.
En la TABLA 4, última columna, se da el orden de evaluación de un operador dado. Cuanto más bajo sea dicho número mayor será su prioridad de ejecución. Si en una operación existen varios operadores, primero se evaluarán los de multiplicación , división y módulo y luego los de suma y resta . La precedencia de los tres primeros es la misma , por lo que si hay varios de ellos, se comenzará a evaluar a aquel que quede más a la izquierda . Lo mismo ocurre con la suma y la resta .
Para evitar errores en los cálculos se pueden usar paréntesis , sin limitación de anidamiento, los que fuerzan a realizar primero las operaciones incluidas en ellos . Los paréntesis no disminuyen la velocidad a la que se ejecuta el programa sino que tan sólo obligan al compilador a realizar las operaciones en un orden dado dado, por lo que es una buena costumbre utilizarlos ampliamente .
Los paréntesis tienen un orden de precedencia 0, es decir que antes que nada se evalúa lo que ellos encierran .
Se puede observar que no existen operadores de potenciación, radicación, logaritmación, etc, ya que en el lenguaje C todas estas operaciones (y muchas otras) se realizan por medio de llamadas a Funciones.
El último de los operadores aritméticos es el de SIGNO . No debe confundirselo con el de resta, ya que este es un operador unitario que opera sobre una única variable cambiando el signo de su contenido númerico. Obviamente no existe el operador +

unitario, ya que su operación sería DEJAR el signo de la variable, lo que se consigue simplemente por omisión del signo.

3. OPERADORES RELACIONALES

Todas las operaciones relacionales dan sólo dos posibles resultados : VERDADERO ó FALSO . En el lenguaje C, Falso queda representado por un valor entero nulo (cero) y Verdadero por cualquier número distinto de cero En la TABLA 5 se encuentra la descripción de los mismos .

TABLA 5 OPERADORES RELACIONALES

SIMBOLO	DESCRIPCION	EJEMPLO	ORDEN DE EVALUACION
<	menor que	(a < b)	5
>	mayor que	(a >b)	5
< =	menor o igual que	(a < = b)	5
>=	mayor o igual que	(a >>= b)	5
= =	igual que	(a = = b)	6
! =	distinto que	(a != b)	6

Uno de los errores más comunes es confundir el operador relacional IGUAL QUE (= =) con el de asignacion IGUAL A (=). La expresión a=b copia el valor de b en a, mientras que a = = b retorna un cero , si a es distinto de b ó un número distinto de cero si son iguales.

Los operadores relacionales tiene menor precedencia que los aritméticos , de forma que a < b + c se interpreta como a < (b + c), pero aunque sea superfluo recomendamos el uso de paréntesis a fin de aumentar la legilibilidad del texto.

Cuando se comparan dos variables tipo char el resultado de la operación dependerá de la comparación de los valores ASCII de los caracteres contenidos en ellas. Asi el caracter a (ASCII 97) será mayor que el A (ASCII 65) ó que el 9 (ASCII 57).

4. OPERADORES LOGICOS

Hay tres operadores que realizan las conectividades lógicas Y (AND) , O (OR) y NEGACION (NOT) y están descriptos en la TABLA 6 .

TABLA 6 OPERADORES LOGICOS

SIMBOLO	DESCRIPCION	EJEMPLO	ORDEN DE EVALUACION
&&	Y (AND)	(a>b) && (c < d)	10
||	O (OR)	(a>b) || (c < d)	11
!	NEGACION (NOT)	!(a>b)	1

Los resultados de la operaciones lógicas siempre adoptan los valores CIERTO ó FALSO. La evaluación de las operaciones lógicas se realiza de izquierda a derecha y se interrumpe cuando se ha asegurado el resultado .

El operador NEGACION invierte el sentido lógico de las operaciones , así será

!(a >> b) equivale a (a < b)
!(a == b) " " (a != b)
etc.

En algunas operaciones suele usárselo de una manera que se presta a confusión , por ejemplo : (!i) donde i es un entero. Esto dará un resultado CIERTO si i tiene un valor 0 y un resultado FALSO si i es distinto de cero .

5. OPERADORES DE INCREMENTO Y DECREMENTO

Los operadores de incremento y decremento son sólo dos y están descriptos en la TABLA 7

TABLA 7 OPERADORES DE INCREMENTO Y DECREMENTO

SIMBOLO	DESCRIPCION	EJEMPLO	ORDEN DE EVALUACION
++	incremento	++i ó i++	1
--	decremento	--i ó i--	1

Para visualizar rapidamente la función de los operadores antedichos , digamos que las sentencias :

```
a = a + 1 ;
```

```
a++ ;
```

tienen una acción idéntica , de la misma forma que

```
a = a - 1 ;
```

```
a-- ;
```

es decir incrementa y decrementa a la variable en una unidad

Si bien estos operadores se suelen emplear con variables int , pueden ser usados sin problemas con cualquier otro tipo de variable . Así si a es un float de valor 1.05 , luego de hacer a++ adoptará el valor de 2.05 y de la misma manera si b es una variable del tipo char que contiene el caracter 'C' , luego de hacer b-- su valor será 'B' .

Si bien las sentencias

```
i++ ;
++i ;
```

son absolutamente equivalentes, en la mayoria de los casos la ubicación de los operadores incremento ó decremento indica CUANDO se realiza éste .

Veamos el siguiente ejemplo :

```
int i = 1 , j , k ;
j = i++ ;
k = ++i ;
```

acá j es igualado al valor de i y POSTERIORMENTE a la asignación i es incrementado por lo que j será igual a 1 e i igual a 2 , luego de ejecutada la sentencia . En la siguiente instrucción i se incrementa ANTES de efectuarse la asignacion tomando el valor de 3 , él que luego es copiado en k .

6. OPERADORES DE ASIGNACION

En principio puede resultar algo futil gastar papel en describir al operador IGUAL A (=) , sin embargo es necesario remarcar ciertas características del mismo .

Anteriormente definimos a una asignación como la copia del resultado de una expresión (rvalue) sobre otra (lvalue) , esto implica que dicho lvalue debe tener LUGAR (es decir poseer una posición de memoria) para alojar dicho valor .

Es por lo tanto válido escribir

```
a = 17 ;
```

pero no es aceptado , en cambio

```
17 = a ;      /* incorrecto */
```

ya que la constante numérica 17 no posee una ubicación de memoria donde alojar al valor de a .

Aunque parezca un poco extraño al principio las asignaciones , al igual que las otras operaciones , dan un resultado que puede asignarse a su vez a otra expresión .

De la misma forma que (a + b) es evaluada y su resultado puedo copiarlo en otra

variable : c = (a + b) ; una asignación (a = b) da como resultado el valor de b , por lo que es lícito escribir
c = (a = b) ;
Debido a que las asignaciones se evalúan de derecha a izquierda , los paréntesis son superfluos , y podrá escribirse entonces :
c = a = b = 17 ;
con lo que las tres variables resultarán iguales al valor de la contante .
El hecho de que estas operaciones se realicen de derecha a izquierda también permite realizar instrucciones del tipo :
a = a + 17 ;
significando esto que al valor que TENIA anteriormente a , se le suma la constante y LUEGO se copia el resultado en la variable .
Como este último tipo de operaciones es por demás común , existe en C un pseudocódigo , con el fin de abreviarlas .
Asi una operación arítmetica o de bit cualquiera (simbolizada por OP)
a = (a) OP (b) ;
puede escribirse en forma abreviada como :
a OP= b ;
Por ejemplo

```
a += b   ;    /* equivale :  a = a + b */
a -= b   ;    /* equivale :  a = a - b */
a *= b   ;    /* equivale :  a = a * b */
a /= b   ;    /* equivale :  a = a / b */
a %= b   ;    /* equivale :  a = a % b */
```

Nótese que el pseudooperador debe escribirse con los dos símbolos seguidos , por ejemplo += , y no será aceptado +(espacio) = .
Los operadores de asignación estan resumidos en la TABLA 8 .

TABLA 8 OPERADORES DE ASIGNACION

SIMBOLO	DESCRIPCION	EJEMPLO	ORDEN DE EVALUACION
=	igual a	a = b	13
op=	pseudocodigo	a += b	13
=?:	asig.condicional	a = (c>b)?d:e	12

Vemos de la tabla anterior que aparece otro operador denominado ASIGNACION CONDICIONAL . El significado del mismo es el siguiente :
lvalue = (operación relacional ó logica) ? (rvalue 1) : (rvalue 2) ;
de acuerdo al resultado de la operación condicional se asignará a lvalue el valor de rvalue 1 ó 2 . Si aquella es CIERTA será lvalue = rvalue 1 y si diera FALSO , lvalue = rvalue 2 .
Por ejemplo, si quisiéramos asignar a c el menor de los valores a ó b , bastará con escribir :
c = (a < b) ? a : b ;

7. OPERADORES DE MANEJO DE BITS

Estos operadores muestran una de las armas más potentes del lenguaje C , la de poder manipulear INTERNAMENTE , es decir bit a bit , las variables .
Debemos anticipar que estos operadores sólo se aplican a variables del tipo char , short , int y long y NO pueden ser usados con float ó double ,
Sabemos que las computadoras guardan los datos organizados en forma digital , en bytes , formado por números binarios de 8 bits y como se vió anteriormente cuando se analizó el tamaño de las variables , un char ocupará un byte de 8 bits , mientras que los

Tipos de datos y conversiones

CAPITULO 2: VARIABLES Y CONSTANTES

1. DEFINICION DE VARIABLES

Si yo deseara imprimir los resultados de multiplicar un número fijo por otro que adopta valores entre 0 y 9 , la forma normal de programar esto sería crear una CONSTANTE para el primer número y un par de VARIABLES para el segundo y para el resultado del producto. Una variable , en realidad , no es más que un nombre para identificar una (o varias) posiciones de memoria donde el programa guarda los distintos valores de una misma entidad . Un programa debe DEFINIR a todas las variables que utilizará , antes de comenzar a usarlas , a fin de indicarle al compilador de que tipo serán , y por lo tanto cuanta memoria debe destinar para albergar a cada una de ellas. Veamos el EJEMPLO 2:

EJEMPLO 2

```
#include <stdio.h>
main()
{
  int multiplicador;          /* defino multiplicador como un entero */
  int multiplicando;          /* defino multiplicando como un entero */
  int resultado;              /* defino resultado como un entero */
  multiplicador = 1000 ;      /* les asigno valores */
  multiplicando = 2 ;
  resultado = multiplicando * multiplicador ;
  printf("Resultado = %d\n", resultado);          /* muestro el resultado */
  return 0;
}
```

En las primeras líneas de texto dentro de main() defino mis variables como números enteros , es decir del tipo "int" seguido de un identificador (nombre) de la misma . Este identificador puede tener la cantidad de caracteres que se desee , sin embargo de acuerdo al Compilador que se use , este tomará como significantes sólo los primeros n de ellos ; siendo por lo general n igual a 32 . Es conveniente darle a los identificadores de las variables , nombres que tengan un significado que luego permita una fácil lectura del programa. Los identificadores deben comenzar con una letra ó con el símbolo de subrayado "_" , pudiendo continuar con cualquier otro caractér alfanumérico ó el símbolo "_" . El único símbolo no alfanumérico aceptado en un nombre es el "_" . El lenguaje C es sensible al tipo de letra usado ; así tomará como variables distintas a una llamada "variable" , de otra escrita como "VARIABLE". Es una convención entre los programadores de C escribir los nombres de las variables y las funciones con minúsculas, reservando las mayúsculas para las constantes.

El compilador dará como error de "Definición incorrecta" a la definición de variables con nombres del tipo de :

4pesos $variable primer-variable !variable etc.etc

NOTA: Los compiladores reservan determinados términos ó palabras claves (Keywords) para el uso sintáctico del lenguaje, tales como: asm, auto, break, case, char, do, for, etc. Si bien estas palabras están definidas para el ANSI C, los distintos compiladores extienden esta definición a OTROS términos, por lo que es aconsejable leer la tabla completa de palabras reservadas del compilador que se vaya a usar, para no utilizarlas en nombres de variables.

Vemos en las dos lineas subsiguientes a la definición de las variables, que puedo ya asignarles valores (1000 y 2) y luego efectuar el cálculo de la variable "resultado". Si prestamos ahora atención a la función printf(), ésta nos mostrará la forma de visualizar el valor de una variable. Insertada en el texto a mostrar, aparece una secuencia de control de impresión "%d" que indica, que en el lugar que ella ocupa, deberá ponerse el contenido de la variable (que aparece luego de cerradas las comillas que marcan la finalización del texto , y separada del mismo por una coma) expresado como un un número entero decimal. Así, si compilamos y corremos el programa, obtendremos una salida:

SALIDA DEL EJEMPLO 2

Resultado = 2000

---

2. INICIALIZACION DE VARIABLES

Las variables del mismo tipo pueden definirse mediante una definición múltiple separándolas mediante " , " a saber :
int multiplicador, multiplicando, resultado;
Esta sentencia es equivalente a las tres definiciones separadas en el ejemplo anterior.
Las variables pueden también ser inicializadas en el momento de definirse .
int multiplicador = 1000, multiplicando = 2, resultado;
De esta manera el EJEMPLO 2 podría escribirse:
EJEMPLO 2 BIS

```
#include <stdio.h>

main()

{

  int multiplicador=1000 , multiplicando=2 ;

  printf("Resultado = %d\n", multiplicando * multiplicador);

  return 0;

}
```

Obsérvese que en la primer sentencia se definen e inicializan simultaneamente ambas variables.La variable "resultado" la hemos hecho desaparecer ya que es innecesaria. Si analizamos la función printf() vemos que se ha reemplazado "resultado" por la operación entre las otras dos variables. Esta es una de las particularidades del lenguaje C : en los parámetros pasados a las funciones pueden ponerse operaciones (incluso llamadas a otras funciones) , las que se realizan ANTES de ejecutarse la función , pasando finalmente a esta el valor resultante de las mismas. El EJEMPLO 2 funciona exactamente igual que antes pero su código ahora es mucho más compacto y claro.

3. TIPOS DE VARIABLES

VARIABLES DEL TIPO ENTERO

En el ejemplo anterior definimos a las variables como enteros (int).
De acuerdo a la cantidad de bytes que reserve el compilador para este tipo de variable,

queda determinado el "alcance" ó máximo valor que puede adoptar la misma.
Debido a que el tipo int ocupa dos bytes su alcance queda restringido al rango entre -32.768 y +32.767 (incluyendo 0).
En caso de necesitar un rango más amplio, puede definirse la variable como "long int nombre_de_variable" ó en forma más abreviada "long nombre_de_variable"
Declarada de esta manera, nombre_de_variable puede alcanzar valores entre -2.347.483.648 y +2.347.483.647.
A la inversa, si se quisiera un alcance menor al de int, podría definirse "short int " ó simplemente "short", aunque por lo general, los compiladores modernos asignan a este tipo el mismo alcance que "int".
Debido a que la norma ANSI C no establece taxativamente la cantidad de bytes que ocupa cada tipo de variable, sino tan sólo que un "long" no ocupe menos memoria que un "int" y este no ocupe menos que un "short",los alcances de los mismos pueden variar de compilador en compilador , por lo que sugerimos que confirme los valores dados en este parágrafo (correspondientes al compilador de Borland C++) con los otorgados por su compilador favorito.
Para variables de muy pequeño valor puede usarse el tipo "char" cuyo alcance está restringido a -128, +127 y por lo general ocupa un único byte.
Todos los tipos citados hasta ahora pueden alojar valores positivos ó negativos y, aunque es redundante, esto puede explicitarse agregando el calificador "signed" delante; por ejemplo:

signed int

signed long

signed long int

signed short

signed short int

signed char

Si en cambio, tenemos una variable que sólo puede adoptar valores positivos (como por ejemplo la edad de una persona) podemos aumentar el alcance de cualquiera de los tipos , restringiéndolos a que sólo representen valores sin signo por medio del calificador "unsigned" . En la TABLA 1 se resume los alcances de distintos tipos de variables enteras

TABLA 1 VARIABLES DEL TIPO NUMERO ENTERO

TIPO	BYTES	VALOR MINIMO	VALOR MAXIMO
signed char	1	-128	127
unsigned char	1	0	255
signed short	2	-32.768	+32.767
unsigned short	2	0	+65.535
signed int	2	-32.768	+32.767
unsigned int	2	0	+65.535
signed long	4	-2.147.483.648	+2.147.483.647
unsigned long	4	0	+4.294.967.295

NOTA: Si se omite el calificador delante del tipo de la variable entera, éste se adopta por omisión (default) como "signed".

VARIABLES DE NUMERO REAL O PUNTO FLOTANTE

Un número real ó de punto flotante es aquel que además de una parte entera, posee fracciones de la unidad. En nuestra convención numérica solemos escribirlos de la siguiente manera : 2,3456, lamentablemente los compiladores usan la convención del PUNTO decimal (en vez de la coma) . Así el numero Pi se escribirá : 3.14159 Otro formato de escritura, normalmente aceptado, es la notación científica. Por ejemplo podrá escribirse 2.345E+02, equivalente a 2.345 * 100 ó 234.5

De acuerdo a su alcance hay tres tipos de variables de punto flotante , las mismas están descriptas en la TABLA 2

TABLA 2 TIPOS DE VARIABLES DE PUNTO FLOTANTE

TIPO	BYTES	VALOR MINIMO	VALOR MAXIMO
float	4	3.4E-38	3.4E+38
double	8	1.7E-308	1.7E+308
long double	10	3.4E-4932	3.4E+4932

Las variables de punto flotante son SIEMPRE con signo, y en el caso que el exponente sea positivo puede obviarse el signo del mismo.

5. CONVERSION AUTOMATICA DE TIPOS

Cuando dos ó mas tipos de variables distintas se encuentran DENTRO de una misma operación ó expresión matemática , ocurre una conversión automática del tipo de las variables. En todo momento de realizarse una operación se aplica la siguiente secuencia de reglas de conversión (previamente a la realización de dicha operación):

- 1) Las variables del tipo char ó short se convierten en int
- 2) Las variables del tipo float se convierten en double
- 3) Si alguno de los operandos es de mayor precisión que los demás , estos se convierten al tipo de aquel y el resultado es del mismo tipo.
- 4) Si no se aplica la regla anterior y un operando es del tipo unsigned el otro se convierte en unsigned y el resultado es de este tipo.

Las reglas 1 a 3 no presentan problemas, sólo nos dicen que previamente a realizar alguna operación las variables son promovidas a su instancia superior. Esto no implica que se haya cambiado la cantidad de memoria que las aloja en forma permanente

Otro tipo de regla se aplica para la conversión en las asignaciones.

Si definimos los términos de una asignación como,"lvalue" a la variable a la izquierda del signo igual y "rvalue" a la expresión a la derecha del mismo, es decir:

"lvalue" = "rvalue" ;

Posteriormente al cálculo del resultado de "rvalue" (de acuerdo con las reglas antes descriptas), el tipo de este se iguala al del "lvalue". El resultado no se verá afectado si el tipo de "lvalue" es igual ó superior al del "rvalue", en caso contrario se efectuará un truncamiento ó redondeo, segun sea el caso.

Por ejemplo, el pasaje de float a int provoca el truncamiento de la parte fraccionaria, en cambio de double a float se hace por redondeo.

5. ENCLAVAMIENTO DE CONVERSIONES (casting)

Las conversiones automáticas pueden ser controladas a gusto por el programador,

imponiendo el tipo de variable al resultado de una operación. Supongamos por ejemplo tener:

```
double  d , e , f = 2.33 ;

int i = 6 ;

e = f * i ;

d = (int) ( f * i ) ;
```

En la primer sentencia calculamos el valor del producto (f * i) , que según lo visto anteriormente nos dará un double de valor 13.98 , el que se ha asignado a e. Si en la variable d quisiéramos reservar sólo el valor entero de dicha operación bastará con anteponer, encerrado entre paréntesis, el tipo deseado. Así en d se almacenará el número 13.00.
También es factible aplicar la fijación de tipo a una variable, por ejemplo obtendremos el mismo resultado, si hacemos:

```
d = (int) f * i ;
```

En este caso hemos convertido a f en un entero (truncando sus decimales)

6. VARIABLES DE TIPO CARACTER

El lenguaje C guarda los caracteres como números de 8 bits de acuerdo a la norma ASCII extendida , que asigna a cada caracter un número comprendido entre 0 y 255 (un byte de 8 bits) Es común entonces que las variables que vayan a alojar caracteres sean definidas como:

```
char c ;
```

Sin embargo, también funciona de manera correcta definirla como

```
int c ;
```

Esta última opción desperdicia un poco más de memoria que la anterior ,pero en algunos casos particulares presenta ciertas ventajas . Pongamos por caso una función que lee un archivo de texto ubicado en un disco. Dicho archivo puede tener cualquier caracter ASCII de valor comprendido entre 0 y 255. Para que la función pueda avisarme que el archivo ha finalizado deberá enviar un número NO comprendido entre 0 y 255 (por lo general se usa el -1 , denominado EOF, fin de archivo ó End Of File), en este caso dicho número no puede ser mantenido en una variable del tipo char, ya que esta sólo puede guardar entre 0 y 255 si se la define unsigned ó no podría mantener los caracteres comprendidos entre 128 y 255 si se la define signed (ver TABLA 1). El problema se obvia facilmente definiéndola como int.
Las variables del tipo caractér también pueden ser inicializadas en su definición, por ejemplo es válido escribir:

```
char c = 97 ;
```

para que c contenga el valor ASCII de la letra "a", sin embargo esto resulta algo engorroso , ya que obliga a recordar dichos códigos . Existe una manera más directa de asignar un caractér a una variable ; la siguiente inicialización es idéntica a la anterior :

```
char c = 'a' ;
```

Es decir que si delimitamos un caracter con comilla simple , el compilador entenderá que debe suplantarlo por su correspondiente código numérico .
Lamentablemente 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 caracter . Un caso típico sería el de "nueva linea" ó ENTER .

Con el fin de tener acceso a los mismos es que aparecen ciertas secuencias de escape convencionales . Las mismas estan listadas en la TABLA 3 y su uso es idéntico al de los caracteres normales , asi para resolver el caso de una asignación de "nueva linea " se escribirá:

```
char c = '\n' ;          /* secuencia de escape */
```

TABLA 3 SECUENCIAS DE ESCAPE

CODIGO	SIGNIFICADO	VALOR ASCII (decimal)	VALOR ASCII (hexadecimal)
'\n'	nueva línea	10	0x0A
'\r'	retorno de carro	13	0x0D
'\f'	nueva página	2	x0C
'\t'	tabulador horizontal	9	0x09
'\b'	retroceso (backspace)	8	0x08
'\''	comilla simple	39	0x27
'\"'	comillas	4	0x22
'\\ '	barra	92	0x5C
'\? '	interrogación	63	0x3F
'\nnn'	cualquier caracter (donde nnn es el código ASCII expresado en octal)		
'\xnn'	cualquier caracter (donde nn es el código ASCII expresado en hexadecimal)		

7. TAMAÑO DE LAS VARIABLES (sizeof)

En muchos programas es necesario conocer el tamaño (cantidad de bytes) que ocupa una variable, por ejemplo en el caso de querer reservar memoria para un conjunto de ellas. Lamentablemente, como vimos anteriormente este tamaño es dependiente del compilador que se use, lo que producirá, si definimos rigidamente (con un número dado de bytes) el espacio requerido para almacenarlas, un problema serio si luego se quiere compilar el programa con un compilador distinto del original

Para salvar este problema y mantener la portabilidad, es conveniente que cada vez que haya que referirse al TAMAÑO en bytes de las variables, se lo haga mediante un operador llamado "sizeof" que calcula sus requerimientos de almacenaje

Está también permitido el uso de sizeof con un tipo de variable, es decir:

```
sizeof(int)

sizeof(char)

sizeof(long double) , etc.
```

8. DEFINICION DE NUEVOS TIPOS (typedef)

A veces resulta conveniente crear otros tipos de variables , ó redefinir con otro nombre las existentes , esto se puede realizar mediante la palabra clave "typedef" , por ejemplo:

```
typedef unsigned long double  enorme  ;
```

A partir de este momento ,las definiciones siguientes tienen idéntico significado:

```
unsigned long double nombre_de_variable ;

enorme nombre_de_variable ;
```

9. CONSTANTES

Aquellos valores que , una vez compilado el programa no pueden ser cambiados , como por ejemplo los valores literales que hemos usado hasta ahora en las inicializaciones de las variables (1000 , 2 , 'a' , '\n' , etc), suelen denominarse CONSTANTES .
Como dichas constantes son guardadas en memoria de la manera que al compilador le resulta más eficiente suelen aparecer ciertos efectos secundarios , a veces desconcertantes , ya que las mismas son afectadas por las reglas de RECONVERSION AUTOMATICA DE TIPO vista previamente.
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) ejemplos :
  1 : tomada como ENTERA
  1F : tomada como FLOAT
  1L : tomada como LONG DOUBLE
- Una variable con parte decimal es tomada siempre como DOUBLE, salvo que se la siga de la letra F ó L
  1.0 : tomada como DOUBLE
  1.0F : tomada como FLOAT
  1.0L : tomada como LONG FLOAT
- Si en cualquiera de los casos anteriores agregamos la letra U ó u la constante queda calificada como UNSIGNED (consiguiendo mayor alcance) :
  1u : tomada como UNSIGNED INT
  1.0UL : tomada como UNSIGNED LONG DOUBLE
- Una variable numérica que comienza con "0" es tomado como OCTAL asi : 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 y 0x1A a 26 unidades decimales.

10. CONSTANTES SIMBOLICAS

Por lo general es una mala práctica de programación colocar en un programa constantes en forma literal (sobre todo si se usan varias veces en el mismo) ya que el texto se hace dificil de comprender y aún más de corregir, si se debe cambiar el valor de dichas constantes.
Se puede en cambio 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, por ejemplo:

```
#define VALOR_CONSTANTE 342

#define PI 3.1416
```