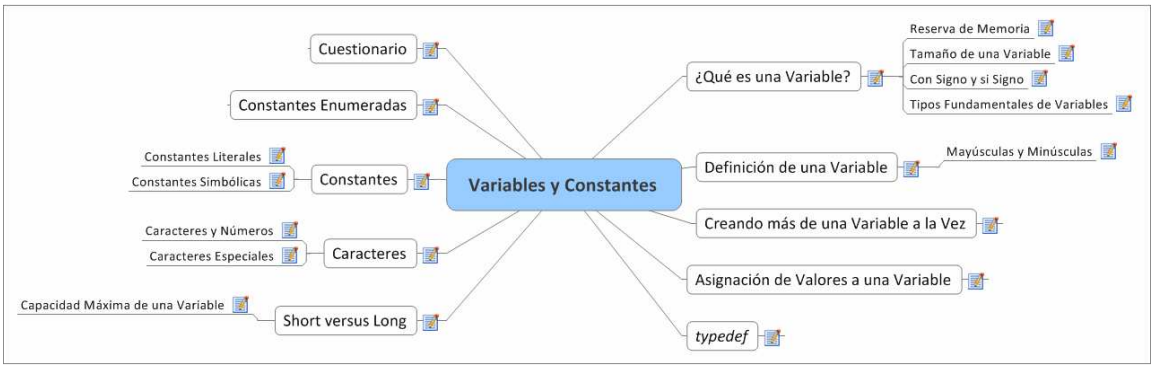
VARIABLES Y CONSTANTES



¿Qué es una Variable?

En C++, una variable es un lugar para almacenar información. Una variable es una localidad de memoria en la cual se almacena una valor que puede ser recuperados posteriormente. La memoria del computador puede ser vista como una serie de casillas. Cada casilla está numerada secuencialmente; dichos números son conocidos como dirección de memoria. Una variable reserva una o más casillas para almacenar un valor.

El nombre de una variable (por ejemplo, miVariable) es una etiqueta para una de esas casillas, de tal manera que pueda ser encontrada fácilmente, sin necesidad de conocer su dirección de memoria. Dependiendo del tamaño de miVariable, esta puede tomar una o más direcciones de memoria.

NOTA. RAM significa memoria de acceso aleatorio. Cuando un programa es ejecutado, es cargado en la memoria RAM a partir de un archivo en disco. Todas las variables son creadas en la memoria RAM. Cuando un programador se refiere a la memoria, usualmente está hablando de la memoria RAM.

Reserva de Memoria

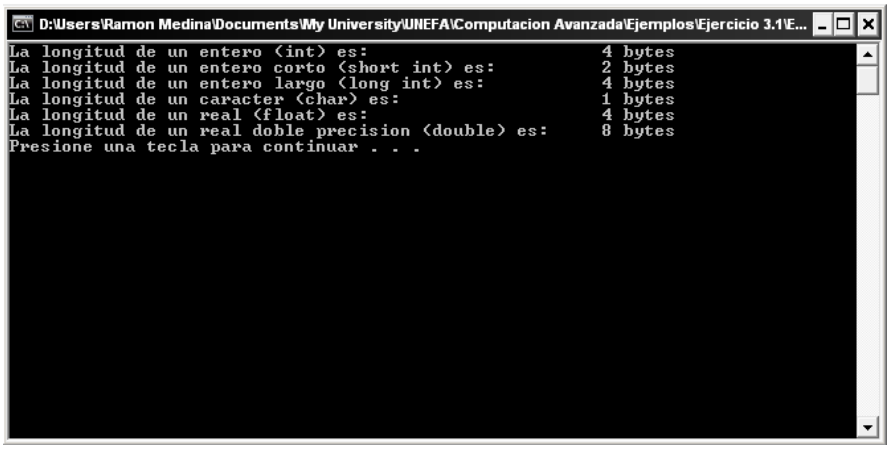
Cuando se define una variable en C++, es necesario indicar al compilador de qué tipo es: entero, carácter, etc. Esta información le dice al compilador, cuánto espacio se requiere para los valores que serán almacenados en la variable. Cada casilla tiene el tamaño de un byte. Si el tipo de variable creada es de dos bytes de longitud, se requerirán dos casillas de memoria. El tipo de variable (por ejemplo entera) le dice al compilador cuánta memoria (cuántas casillas) se deben reservar para la variable.

Tamaño de una Variable

Cada variable toma un espacio de memoria específico. Un entero puede, por ejemplo, tomar dos bytes en un computador y cuatro en otro; sin embargo, en el mismo computador, siempre tomará el mismo espacio de memoria. Una variable de tipo carácter (char) usualmente toma un único byte. Un entero corto (short int) toma dos bytes en la mayoría de los computador, mientras que un entero largo (long int) toma cuatro. El programa a continuación determina cuánto espacio de memoria requiere cada tipo de variable en un computador en particular.

```
1:  #include <iostream.h>
2:  #include <stdlib.h>
```

```
3:  int main()
4:  {
5:      cout << "La longitud de un entero (int) es:\t\t\t"    << sizeof(int)    << " bytes\n";
6:      cout << "La longitud de un entero corto (short int) es:\t\t" << sizeof(short)  << "
        bytes\n";
7:      cout << "La longitud de un entero largo (long int) es:\t\t"  << sizeof(long)   << "
        bytes\n";
8:      cout << "La longitud de un carácter (char) es:\t\t\t"    << sizeof(char)   << "
        bytes\n";
9:      cout << "La longitud de un real (float) es:\t\t\t"   << sizeof(float)  << " bytes\n";
10:     cout << "La longitud de un real doble precision (double) es:\t"    << sizeof(double)
        << " bytes\n";
11:     system("PAUSE");
12:     return 0;
13: }
```



NOTA. El número de bytes usado por cada tipo de variable, puede variar de un computador a otro.

La mayor parte del listado anterior ya ha sido tratada anteriormente. La única característica nueva el uso de la función sizeof() en las líneas 5 a la 10. La función sizeof() es proporcionada por el compilador e indica el tamaño del objeto que le es proporcionado como argumento. Por ejemplo, en la línea 5 el argumento de sizeof() es la palabra int. Utilizando dicha función, es posible determinar que en la computadora donde se ejecutó el programa, el tipo de dato entero ocupa 4 bytes y es igual a la del entero largo.

Con Signo y sin Signo

Los tipos de datos enteros vienen en dos variedades: con signo y sin signo. La idea aquí es que algunas veces se requieren números negativas y otras no. Los enteros (short y long) sin la palabra unsigned son considerados enteros con

signo. Los enteros con signo pueden ser positivos o negativos. Los enteros sin signo (unsigned) son siempre positivos.

Ya que los enteros con signo y sin signo usan el mismo número de bytes, el mayor número que se puede almacenar en un entero sin signo, es el doble de grande que el mayor positivo que puede ser almacenado en un entero con signo. Un entero corto sin signo (unsigned short int) puede almacenar números ente 0 y 65.535. La mitad de los números representados por un entero corto con signo son negativos; por lo tanto, un entero corto con signo (short int) puede almacenar valores entre -32.768 y 32.767.

Tipos Fundamentales de Variables

Existen otros tipos de variables en C++. Ellos pueden ser divididos en variables enteras, variables reales y caracteres. Las variables reales (de punto flotante) pueden almacenar valores expresadas como fracciones. Las variables de tipo carácter pueden caracteres alfanuméricos (ASCII).

NOTA: ASCII es un conjunto estandarizado de códigos de caracteres. El significado de ASCII es *American Standard Code for Information Interchange*. Casi todos los sistemas operativos modernos soportan la codificación ASCII, aunque muchos de ellos pueden también manejar códigos internacionales.

Los tipos de variables usados en C++ son descritos en la tabla que se muestra a continuación. Dicha tabla muestra el tipo de variable, cuánto espacio de memoria típicamente ocupa y qué tipos de datos pueden ser almacenados en ellas. Los valores que pueden ser almacenados, están determinados por el tamaño del tipo de variable.

Tipo	Tamaño	Rango de valores
unsigned short int	2	0 a 65.535
short int	2	-32.768 a 32.767
unsigned long int	4	0 a 4.294.967.295
long int	4	-2.147.483.648 a 2.147.483.647
int (16 bits)	2	-32.768 a 32.767
int (32 bits)	4	-2.147.483.648 a 2.147.483.647
unsigned int (16 bits)	2	0 a 65.535
unsigned int (32 bits)	4	0 a 4.294.967.295
char	1	0 a 255
float	4	1,2e-38 a 3,4 e38
double	8	2,2e-308 a 1,8e308

NOTA. El tamaño de las variables puede ser diferente a los mostrados en la tabla, dependiendo del compilador y del computador usado.

Definición de una Variable

Una variable es creada o definida, escribiendo su tipo, seguido de uno o más espacios, seguido a su vez del nombre de la variable de un punto y coma (;). El nombre de la variable puede prácticamente ser cualquier combinación de letras, pero no puede contener espacios en blanco. Algunos nombres válidos de variables incluyen x, J23qrsnf y MiEdad. Un buen nombre de variable indica para qué es usada la variable; el uso de nombres adecuados facilita el entendimiento de un programa. La sentencia a continuación define una variable llamada MiEdad:

```
int MiEdad;
```

Como práctica sana de programación, evite nombres como J23qrsnf, y restrinja los nombres de una sola letra (por ejemplo x o i) a variables que son usadas sólo brevemente. Trate siempre de usar nombres expresivos tales como MiEdad.

Como experimento, trate de inferir qué hacen los programas a continuación, basado en las líneas de código que se muestran:

Ejemplo 1

```
main()
{
    unsigned short x;
    unsigned short y;
    ULONG z;
    z= x*y;
}
```

Ejemplo 2

```
main()
{
    unsigned short Ancho;
    unsigned short Largo;
    unsigned short Area;
    Area = Ancho * Largo;
}
```

Claramente, el segundo programa es más fácil de entender, y la molestia de tener que escribir nombres de variables más largos, se ve recompensada por lo mucho más fácil que es de mantener el segundo programa.

Mayúsculas y Minúsculas

El C y el C++ son *case-sensitive*. Esto significa que al momento de definir nombres de elementos del programa, las letras minúsculas y mayúsculas son consideradas diferentes. Una variable llamada edad es diferente de otra llamada Edad.

Existen algunas convenciones acerca de cómo definir nombres de variables. Sin embargo, lo más importante no es que convención se siga, sino ser consistente a lo largo del programa.

Muchos programadores prefieren usar letras en minúscula para los nombres de variable. Si el nombre requiere de dos palabras (por ejemplo mi carro) existen dos convenciones populares: mi_carro o miCarro.

Algunas personas consideran que el carácter _ (*underscore*) es más fácil de leer, mientras que otros prefieren evitarlo porque es difícil de escribir.

> NOTA. Muchos programadores avanzados utilizan una convención llamada notación húngara. La idea de esta convención es preceder el nombre de cada variable con un conjunto de caracteres que identifiquen el tipo. Las variables enteras comienzan con la letra i (*integer*) y los enteros largos con l (long). Otras notaciones indican constantes, globales, apuntadores y así sucesivamente. Esto es de importancia aún mayor en C++, ya que este lenguaje soporta la creación de tipos de datos definidos por el usuario y porque el C++ es 'fuertemente tipeado'.

Algunas palabras son reservadas en C++, de tal forma que no pueden ser usadas como nombres de variables. Existen términos usados por el compilador para controlar el programa. Estos términos incluyen if, while, for y *main*. Estos términos tampoco pueden ser usados como nombres de variables.

> NOTA. Defina variables escribiendo el tipo y luego el nombre. Utilice nombres con significado. Recuerde que el C++ diferencia mayúsculas de minúsculas. No use términos del C++ como nombres de variables. Conozca el número de bytes que cada tipo de variable consume y los tipos de valores que pueden ser almacenados. No use variables sin signo (unsigned) para números negativos.

Creando más de una Variable a la Vez

Es posible declarar más de una variable en una única sentencia, escribiendo el tipo y luego los nombres de las variables separadas por comas. Por ejemplo:

```
unsigned int miEdad, miPeso; // dos variables enteras sin signo
```

```
long area, ancho, largo;          // tres enteros largos
```

En el ejemplo anterior, las variables MiEdad y MiPeso son declaradas como enteros sin signo. La segunda línea declara tres variables enteras largas con los nombres area, ancho y largo. El tipo (long) es asignado a todas las variables; no es posible mezclar tipos de variables en una única declaración.

Asignación de Valores a una Variable

Para almacenar un valor en una variable se usa el operador de asignación (=). Para almacenar el número 5 en la variable ancho se usa la siguiente sentencia:

```
unsigned short ancho;
ancho= 5;
```

Es posible combinar la declaración de una variable con la asignación de valores escribiendo lo siguiente:

```
unsigned short ancho= 5;
```

En este caso se habla de inicialización en lugar de asignación, aunque la diferencia sea muy pequeña. La diferencia esencial de una inicialización es que ocurre en el momento en que la variable es creada.

Así como es posible declarar más de una variable a la vez, es también posible inicializar más de una variable a la vez.

```
int MiEdad= 43, tuEdad, suEdad= 17;
```

En este caso se crean tres variables enteras y son inicializadas la primera y la tercera.

```
1:  #include <iostream.h>
2:  #include <stdlib.h>
3:  int main()
4:  {
5:        unsigned short int ancho = 5, longitud;
6:        longitud = 10;
7:        // Crea un entero corto sin signo y lo inicializa con el resultado
8:        // de multiplicar ancho por longitud
9:        unsigned short int area  = ancho*longitud;
10:
11:       cout << "Ancho:" << ancho << "\n";
```

```
12:      cout << "Longitud: "  << longitud << endl;
13:      cout << "Area: " << area << endl;
14:
15:      system("PAUSE");
16 :     return 0;
17: }
```

La línea 1 incluye la librería *iostream*, necesaria para poder usar la instrucción cout. En la línea 5 se definen las variables ancho y longitud; la variable ancho es inicializada a 5. En la línea 6 se le asigna el valor 10 a la variable longitud.

En la línea 9 se define la variable área, que es inicializada con el resultado de multiplicar ancho por longitud. En las líneas 11 a la 13, son mostrados por pantalla los contenidos de las variables ancho, longitud y área.

typedef

Escribir definiciones de variables como por ejemplo unsigned short int puede hacerse tedioso y propenso a errores. Por esta razón, el C++ permite la creación de sobrenombres (alias) para este tipo de frases, utilizando el comando typedef. Por ejemplo

```
typedef unsigned short int USHORT
```

crea el nombre USHORT que puede ser usado en cualquier lugar del programa, en lugar de unsigned short int. El listado a continuación, muestra el programa anterior modificado con el uso del comando typedef.

```
1:  #include <iostream.h>
2:  #include <stdlib.h>
```

```
3:
4:   typedef unsigned short int USHORT;
5:
6:   int main()
7:   {
8:         USHORT ancho = 5, longitud;       // definición de alias
9:         longitud = 10;
10:        // Crea un entero corto sin signo y lo inicializa con el resultado
11:        // de multiplicar ancho por longitud
12:        USHORT area  = ancho*longitud;
13:
14:        cout << "Ancho:" << ancho << "\n";
15:        cout << "Longitud: "  << longitud << endl;
16:        cout << "Area: " << area << endl;
17:
18:        system("PAUSE");
19:        return 0;
20: }
```

En la línea 4 se define USHORT como entero corto sin signo (unsigned short int). Esta definición puede ser luego usada en cualquier lugar del programa.

Short versus Long

Un fuente común de confusión para los programadores de C y C++ es cuándo declarar una variable long y cuándo declararla short. Sin embargo, la regla es muy simple: si existe la posibilidad de que la información que se vaya a almacenar no quepa, use el modelo más grande.

Sobre la base de que las variables de tipo unsigned short son de dos bytes de longitud, el mayor valor que pueden almacenar es 65.535. Para los enteros con signo, el máximo valor posible es la mitad del citado anteriormente. Aunque los enteros largos sin signo (unsigned long int) pueden almacenar número muy grandes (4.294.967.295), es sin embargo finito. Si se necesita almacenar un número aún más grande, es necesario usar float o double, aunque se pierda precisión. Las variables flotantes en simple y doble precisión, pueden almacenar números extremadamente grandes, pero sólo los primero 7 o 19 dígitos (respectivamente) son significativos en la mayoría de las computadoras. Esto significa, que a partir del siguiente dígito, el número es 'redondeado'.

Capacidad Máxima de una Variable

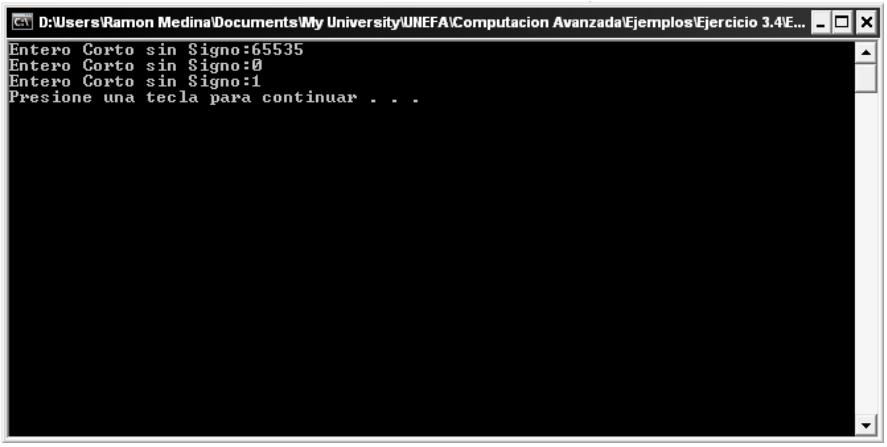
El hecho de que un entero largo sin signo tenga una capacidad de almacenamiento finita rara vez es un problema. Sin embargo, conviene preguntarse ¿qué pasa si esto ocurre?. Cuando un entero sin signo rebasa su

valor máximo, comienza desde cero. El ejemplo a continuación muestra este fenómeno.

```
1:  #include <iostream.h>
2:  #include <stdlib.h>
3:
4:  int main()
5:  {
6:      unsigned short int enteroCorto;
7:      enteroCorto = 65535;
8:      cout << "Entero Corto sin Signo:" << enteroCorto << endl;
9:      enteroCorto++;
10:     cout << "Entero Corto sin Signo:" << enteroCorto << endl;
11:     enteroCorto++;
12:     cout << "Entero Corto sin Signo:" << enteroCorto << endl;
13:
14:     system ("PAUSE");
15:     return 0;
16: }
```

En la línea 6 se declara una variable entera corta sin signo con el nombre enteroCorto, que en la computadora donde fue ejecutado el programa, pueden contener números entre 0 y 65.535. En la línea 7 se le asigna el mayor valor posible a la variable enteroCorto, y es mostrada por pantalla en la línea 8.

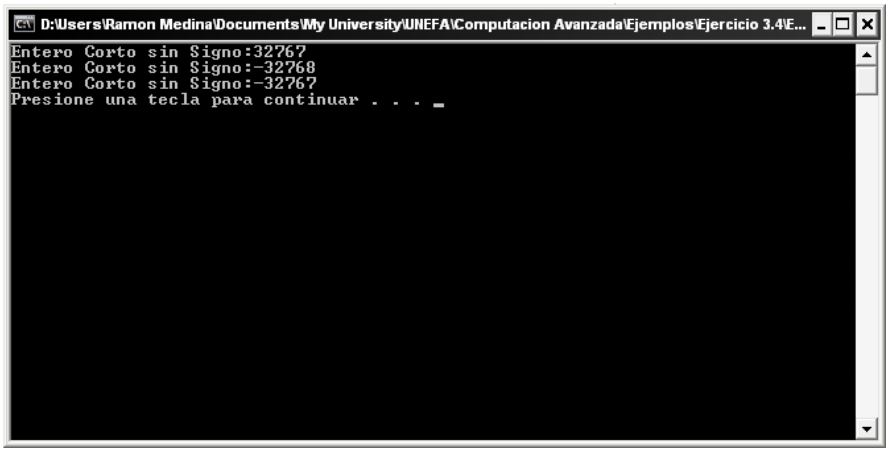
En la línea 9, enteroCorto es incrementado en 1 (se le suma 1 al contenido previo). El operador de incremento es ++. Desde un punto de vista matemático, la variable enteroCorto luego de haber sido incrementada, debería contener el valor 65.536. Sin embargo, como el valor máximo que puede contener es 65.535, al sumarle 1, regresa a 0, que es mostrado en la línea 10. En la línea 11 se incrementa nuevamente y su contenido, en este caso 1, es mostrado en la línea 12.



En el caso de los enteros con signo, la situación es ligeramente distinta ya que el espacio de almacenamiento se usa para representar números positivos y negativos (la mitad del espacio para cada tipo). Cuando se rebasa el mayor número positivo, la variable adopta el valor más negativo que puede almacenar.

```
1:  #include <iostream.h>
2:  #include <stdlib.h>
3:
4:  int main()
5:  {
6:      short int enteroCorto;
7:      enteroCorto = 32767;
8:      cout << "Entero Corto sin Signo:" << enteroCorto << endl;
9:      enteroCorto++;
10:     cout << "Entero Corto sin Signo:" << enteroCorto << endl;
11:     enteroCorto++;
12:     cout << "Entero Corto sin Signo:" << enteroCorto << endl;
13:
14:     system ("PAUSE");
15:     return 0;
16: }
```

En la línea 6 se declara la variable enteroCorto como un entero corto con signo (no indicar unsigned de manera explícita significa que la variable es de tipo signed). El programa procede de manera similar al anterior, pero con resultados diferentes debido al tipo de variable.



Caracteres

Las variables de tipo carácter (char) son típicamente de un byte de longitud, espacio suficiente para almacenar 256 valores distintos. Un char puede ser interpretado como un número pequeño o como un carácter ASCII. ASCII significa *American Standard Code for Information Interchange*. El conjunto de caracteres ASCII y su equivalente ISO (*International Standards Organization*) son dos maneras de codificar letras, números y signos de puntuación.

> NOTA. Las computadoras no saben nada acerca de letras, signos de puntuaciones y oraciones. Todo lo que entienden es número. De hecho, en realidad de lo que saben es si hay niveles de tensión presentes o no. Si lo hay, esto es interpretado como un 1; si no, es interpretado como un 0. Agrupando unos y ceros, el computador puede generar patrones que pueden ser interpretados como números, los cuales a su vez pueden ser asignados a letras y signos de puntuación.

En el código ASCII, la letra 'a' minúscula tiene asignado el valor 97. Todas las letras, mayúsculas y minúsculas, así como los números y signos de puntuación, tienen asignados valores entre 1 y 128. Los otros 128 números están disponibles para representar otros caracteres y símbolos.

Caracteres y Números

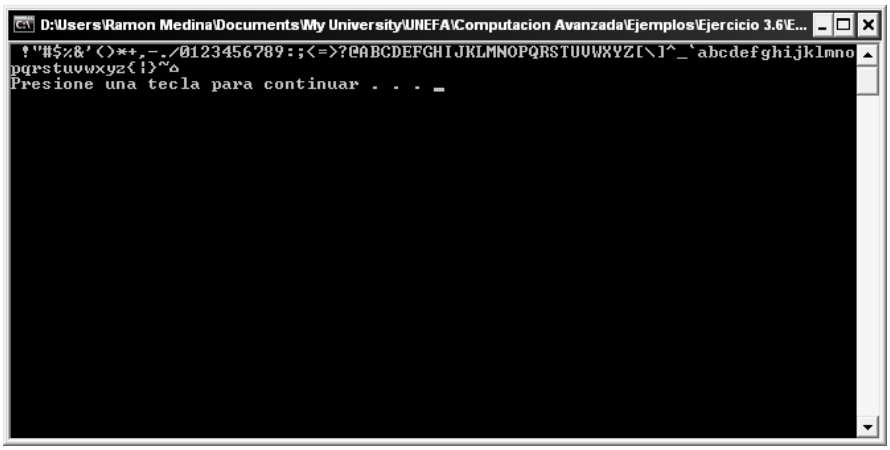
Cuando se almacena un carácter en una variable, por ejemplo la letra 'a' en una variable de tipo char, lo que realmente se deposita es un número entre 0 y 255. El compilador sabe, sin embargo, cómo traducir entre un carácter y su correspondiente código ASCII.

La relación entre una letra y su código es arbitraria; no existe ninguna razón por la que la letra 'a' deba tener el código 97. Mientras que el teclado, el compilador y la pantalla lo interpreten de la misma manera, no existe posibilidad de problema al respecto. Es importante sin embargo caer en cuenta de que existe una diferencia importante entre el carácter '5' y el número 5. El carácter '5' corresponde al código ASCII 53, mientras que el número 5 es simplemente el número 5.

El ejemplo a continuación imprime los caracteres cuyos códigos ASCII están entre 32 y 128.

```
1:  #include <iostream.h>
2:  #include <stdlib.h>
3:
4:  int main()
```

```
5:  {
6:      int i;
7:
8:      for (i=32; i<128; i++)
9:      cout << (char) i;
10:     cout << endl;
11:     system ("PAUSE");
12:     return 0;
13: }
```



Caracteres Especiales

El compilador de C/C++ reconoce algunos caracteres especiales. Los caracteres especiales pueden ser empleados al mostrar caracteres por pantalla, imprimirlos o almacenarlos en un archivo. La tabla a continuación muestra los más comunes.

Carácter	Significado
\n	Salto de línea
\t	Tabulación
\b	Retroceso
\"	Comilla doble
\'	Comilla simple
\?	Signo de interrogación
\\	Backslash

NOTA. Un carácter de escape modifica el significado del carácter que le sigue. Por ejemplo un carácter n significa simplemente la letra n; sin embargo, cuando la n es precedida por un carácter de escape (\), significa salto de línea.

Constantes

Como las variables, las constantes son localidades para almacenamiento de información. A diferencia de las variables, y como su nombre lo implica, el contenido de las constantes no cambia. Las constantes deben ser inicializadas en el momento en que son creadas y no será posible asignar valores posteriormente.

En C/C++ existen dos tipos de constantes: literales y simbólicas.

Constantes Literales

Una constante literal es un valor escrito directamente en un programa, cuando es requerido. Por ejemplo:

```
int miEdad= 43;
```

miEdad es una variable de tipo entera. 39 es una constante literal. No se puede asignar un valor a 39; su valor no puede cambiar.

Constantes Simbólicas

Una constante simbólica es aquella que es representada por un nombre, de manera parecida a como se hace con una variable. Sin embargo, como se dijo anteriormente, la constante puede ser inicializada pero no asignada.

Si un programa tiene una variable llamada estudiantes y otra llamada clases, es posible calcular cuántos estudiantes hay en total, si se conoce el número de estudiantes por clase (por ejemplo 30).

```
estudiantes= clases * 30          // * significa multiplicación
```

En este caso, 30 es una constante literal. El código puede ser más fácil de leer y mantener, si la constante literal se reemplaza por una constante simbólica. Por ejemplo:

```
estudiantes= clases * estudiantesPorClase
```

Si se hace necesario cambiar el número de estudiantes por clase, basta con modificar la definición de la constante estudiantesPorClase, sin tener que hacerlo de manera individual en cada parte del programa donde se haga referencia a dicho valor.

En C existe una manera de definir constantes. En C++ existen dos (una de ellas compartida con C). La forma tradicional y actualmente considerada obsoleta, es usando la directiva #define. Para definir una constante de esta manera se procede como se muestra a continuación:

```
#define estudiantesPorClase 30
```

Conviene notar que estudiantesPorClase no es de ningún tipo en particular (int, char, etc.). La directiva #define simplemente genera un reemplazo de texto. Cada vez que el preprocesador ve la palabra estudiantesPorClase, la sustituye por 30. Debido a que el preprocesador se ejecuta antes del compilador, este último nunca ve la constante sino que lo que procesa en todos los casos es el número 30.

Otra manera de definir constantes simbólicas es utilizando el comando const. Por ejemplo:

```
const unsigned short int estudiantesPorClase= 15;
```

Este ejemplo declara una constante simbólica llamada estudiantesPorClase, que en este caso es de tipo entero corto sin signo. Este método tiene varias ventajas al facilitar la manutención del código y prevenir la presencia de errores. La mayor diferencia es que esta constante tiene tipo, por lo que el compilador la interpretará de acuerdo dicho tipo.

NOTA. Las constantes no pueden ser modificadas durante la ejecución del programa. Si requiere hacerlo, debe modificar la definición de las constante y recompilar el código.

PRÁCTICAS SANAS DE PROGRAMACIÓN. No use el término int. Use short y long para dejar en claro que tamaño de variable requiere. Tenga en cuenta que rebasar la capacidad de una variable puede producir resultados inesperados. Defina variables con nombres significativos que hagan referencia a su uso. No use comandos del compilador como nombres de variables.

Constantes Enumeradas

Las constantes enumeradas permiten la creación de nuevos tipos y de declarar variables asociadas a dichos tipos, cuyos valores pueden estar restringidos a un conjunto dado. Por ejemplo, usted puede declarar COLOR como una

enumeración, y definir cinco valores posibles para ROJO, AZUL, VERDE, BLANCO y NEGRO.

La forma de definir constantes enumeradas consiste en escribir el comando enum seguido del nombre del tipo, una llave abierta ({), la lista de valores posibles (separados por comas), una llave cerrada (}) y un punto y coma. Por ejemplo:

```
enum COLOR {ROJO, AZUL, VERDE, BLANCO, NEGRO};
```

Esta sentencia lleva a cabo dos tareas:

Hace que COLOR sea el nombre de una enumeración, y por tanto de un nuevo tipo de datos

Hace que ROJO sea una constante simbólica con el número 0 asociado, AZUL otra constante simbólica con el número 1 asociado, VERDE con el 2, y así sucesivamente.

Cada constante enumerada tiene asociada un valor entero. Si no se indica lo contrario, la primera constante estará asociada al valor 0. Cada una de las constantes puede ser inicializada con un valor en particular. Por ejemplo:

```
enum COLOR {ROJO=100, AZUL, VERDE=500, BLANCO, NEGRO=700};
```

Las constante ROJO estará asociada al valor 100, AZUL al valor 101, VERDE a 500, BLANCO a 501 y NEGRO a 700.

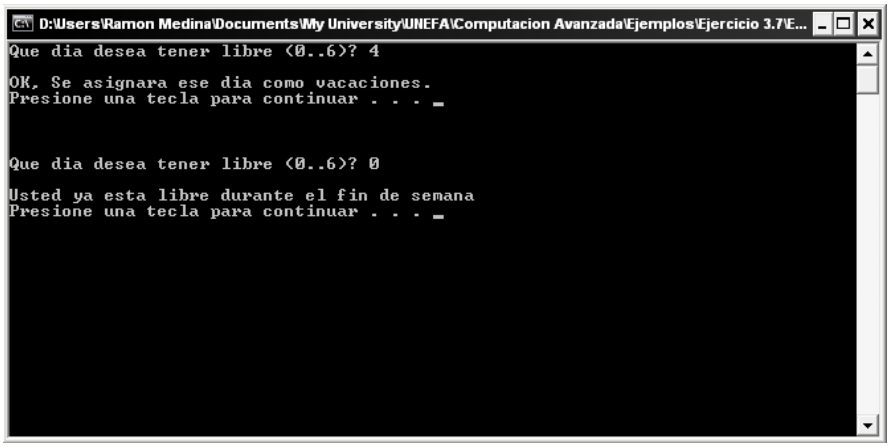
Es posible declarar variables de tipo COLOR, a las que se le puede asignar sólo valores contenidos en la enumeración, en este caso ROJO, AZUL, VERDE, BLANCO y NEGRO o 100, 101, 500, 501 y 700. En realidad, es posible almacenar a dicha variable cualquier entero aunque no se de los contenidos en la enumeración. Sin embargo, la mayoría de los compiladores recomendarán no hacerlo. Es importante saber que las variables enumeradas son de tipo entero sin signo y que por lo tanto las constantes enumeradas deben ser de tipo entero. Sin embargo, resulta de mucha utilidad el uso de enumeraciones cuando se trabaja con colores, días de la semana y cualquier otro conjunto de valores relacionados.

```
1:  #include <iostream.h>
2:  #include <stdlib.h>
3:
4:  int main ()
5:  {
6:      enum DIAS {Domingo,Lunes,Martes,Miercoles,Jueves,Viernes,Sabado};
7:
8:      DIAS diaLibre;
```

```
9:      int  x;
10:
11:     cout << "Que dia desea tener libre (0..6)? ";
12:     cin >> x;
13:     diaLibre= DIAS(x);
14:
15:     if ((diaLibre==Domingo)|| (diaLibre==Sabado))
16:          cout << "\nUsted ya esta libre durante el fin de semana\n";
17:     else cout << "\nOK, Se asignara ese dia como vacaciones.\n";
18:
19:     system ("PAUSE");
20:     return 0;
21: }
```

En la línea 6 se define la constante enumerada DIAS, con siete valores posibles que inician en 0. En la línea 11 se le hace una pregunta al usuario. El valor ingresado (entre 0 y 6) es comparado en la línea 15 con los posibles de la enumeración, tomándose las acciones correspondientes.

Cuando el usuario es interrogado, no puede escribir la palabra "Domingo" como respuesta, ya que el programa no conoce como traducir los caracteres en dicha palabra, al valor enumerado Domingo definido en DIAS.



Cuestionario

1. ¿Si al rebasar la capacidad de almacenamiento de un entero corto, el contenido de la variable se reinicia, por qué no usar siempre enteros largos?

2. ¿Qué sucede si se le asigna un número con punto decimal a una variable entera?. Por ejemplo,

```
int unNumero= 5.4;
```

3. ¿Cuál es la ventaja de usar constantes simbólicas en lugar de las literales?

4. ¿Qué sucede si se asigna un número negativo a una variable sin signo? Por ejemplo,

```
unsigned int unNumeroPositivo= -1;
```

5. ¿Cuál es la diferencia entre una variable entera y una de punto flotante?

6. Cuál es la diferencia entre unsigned short int y long int?

7. ¿Cuál es la ventaja de usar const versus #define al definir constantes?

8. ¿Qué hace que el nombre de una variable sea bueno o malo?

9. Dada la siguiente enumeración, ¿qué valor le corresponde a AZUL? enum COLOR {BLANCO, NEGRO=100, ROJO, AZUL, VERDE=300};

10. ¿Cuáles de las siguientes son buenos nombres de variables?:

```
edad
!ex
R79J
ingresoTotal
__invalido
```

11. ¿Cuál sería el tipo de variable correcto para almacenar la siguiente información?

```
Su edad
El área del patio de su casa
La cantidad de estrellas en la galaxia
El promedio de lluvia durante el mes de Enero
```

12. Defina variables con buenos nombres, para los casos de la pregunta anterior

13. Declare una constante simbólica para el valor de PI (3,14159)

14. Declare una variable flotante e inicialícela usando la constante PI definida anteriormente