Constantes y variables C++

Constantes y variables C++

Son *lugares de memoria que almacenan un valor.*

Constantes : valor no varía durante la ejecución del programa.

```
const <tipo> <identificador>[=<valor>][,<iden>[=<exp>][...]];
```

Variables : valor si puede variar durante la ejecución del programa.

Tanto las variables como las constantes están constituidas por un **nombre** y un **valor.** El nombre lo llamaremos **identificador.**

Toda variable utilizada en un programa debe ser declarada previamente.

```
<tipo> <identificador>[=<exp>][,<iden>[=<exp>][...]];
```

Dependiendo donde sean declaradas pueden ser:

- **Variables globales** (declaradas antes de main())
- **Variables locales (**declaradas dentro de las funciones o de los bloques.)
- **Parámetros** (declarados en parámetros de las funciones)

```
//locales01.cpp
#include <iostream>
using namespace std;

int x =12;   //variable global. Definida fuera de cualquier bloque. Ámbito en todo el programa.
void globales();

int main()
{
        int x=5; //variable local. Ámbito dentro de main()
        {
                int x=10; //variable local. Ámbito dentro de este bloque {}
                cout<<x<<endl; //Muestra valor 10
        }
        cout<<x<<endl; //Muestra valor 5
        globales(); //Muestra valor 12
        return 0;
}

void globales()
{
        cout<<x<<endl;
}
```

OPERADORES
Y
EXPRESIONES

Expresiones

Un programa se basa en la realización de una serie de cálculos con los que se producen los resultados deseados. Dichos resultados se almacenan en variables y a su vez son utilizados para nuevos cálculos, hasta obtener el resultado final. Los cálculos se producen mediante la evaluación de expresiones en las que se mezclan operadores con operandos (constantes literales, constantes simbólicas o variables).

En caso de que en una misma expresión se utilice más de un operador habrá que conocer la precedencia de cada operador. (Lo mejor usar paréntesis para cambiar el orden de precedencia).

Ejemplo de diferentes expresiones:

```
a + b/2 * 5
(a + b)/2 * 5
 a + b/(2 * 5)
(a + b)/(2 * 5)
a == 5
```

Operador de asignación.

<nombre_de_variable> = <expresión> ;

El operador **=** asigna a la variable de la izquierda el resultado de la expresión

Otros operadores de asignación

Operador	Expresión	Equivalencia
=	a=b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

OPERADORES ARITMÉTICOS

- valor	Menos unario
valor + valor	Suma
valor - valor	Resta
valor * valor	Producto (multiplicación)
valor / valor	División (entera o real según el tipo de operandos)
valor % valor	Módulo (resto de la división) (sólo tipos enteros)

Conversiones Automáticas (Implícitas) de Tipos

Promoción: en cualquier operación en la que aparezcan dos tipos diferentes se eleva el rango del que lo tiene menor para igualarlo al del mayor. El rango de los tipos de mayor a menor es el siguiente:

double, float, long, int, short, char

Almacenamiento: En una sentencia de asignación, el resultado final de los cálculos se reconvierte al tipo de la variable al que está siendo asignado.

OPERADORES RELACIONALES

El resultado es de tipo **bool** (true o false)

expresión1 < expresión2	Menor que
expresión1 <= expresión2	Menor o igual que
expresión1 > expresión2	Mayor que
expresión1 >= expresión2	Mayor o igual que
expresión1 == expresión2	Igual que
expresión1 != expresión2	Distintos

Es conveniente usar paréntesis para evaluar estas expresiones debido al orden de precedencia de los operadores

Operador condicional

condición ? valor1 : valor2

Si condición es true entonces el resultado es valor1, en otro caso es valor2

OPERADORES LÓGICOS

Aplicables a operandos de tipo booleano, y dan como resultado un valor **bool**eano

! <expr>	NOT Negación lógica
<expr1> && <expr2>	AND lógico
<expr1> || <expr2>	OR lógico

Tablas de verdad de los operadores lógicos

<expr1>	<expr2>	<expr1> **&&** <expr2>	<expr1> **||** <expr2>
F	F	F	F
F	T	F	T
T	F	F	T
T	T	T	T

<expr>	**!** <expr>
F	T
T	F

OPERADORES DE BIT

Sólo aplicables a operandos de tipos enteros. El resultado es del mismo tipo que los operandos:

~ valor	Negación de bits (complemento)
valor << despl	Desplazamiento de bits a la izq.
valor >> despl	Desplazamiento de bits a la dch.
valor & valor	AND de bits
valor | valor	OR de bits
valor ^ valor	XOR de bits

```
//Operacionesbits.cpp
#include<iostream>
using namespace std;
int main ()

{
   int i=5;  //5 en binario 0.....101
   cout<< ~i<<endl;
   /*Su complemento es 111.....010 . Por tener el primer bit 1 será negativo
   Para calcular su nuevo valor se calcula complemento a la base
   Cambiando ceros por unos y unos por cero y añadiendo 1
   Por tanto el valor absoluto del número negativo sería 010+1 es decir 6
   */
   i=10;  //Binario 0.......1010
   i=i<<2; //Binario 0....101000
   cout << i<<endl;
   cout << (5 & (~5)); //AND bit a bit entre el 5 y su complementario dará 0

   return 0;
}
```