**VARIABLES Y CONSTANTES EN C**

Nombres de las variables

Sólo están permitidas letras de la 'a' a la 'z' (la ñ no vale), números y el símbolo '\_', puede contener números, pero no en el primer carácter.

Ejemplos de nombres válidos:

camiones

numero

buffer

a1

j10hola29

num\_alumnos

Ejemplos de nombres no válidos:

1abc

nombre?

num/alumnos

Tampoco valen como nombres de variable las palabras reservadas que usa el compilador. Por ejemplo: *for, main, do, while.*

C distingue entre mayúsculas y minúsculas. Por lo tanto:

Nombre

nombre

NOMBRE

serían tres variables distintas.

¿Dónde se declaran las variables?

Tenemos dos posibilidades, una es declararla como global y otra como local. Es global aquella variable que se declara fuera de la función main y local la que se declara dentro:

|  |  |
| --- | --- |
| **Variable Global** | **Variable Local** |
| #include <stdio.h>  int x;  int main() {  } | #include <stdio.h>  int main() {  int x;  } |

Las variables globales se pueden usar en cualquier procedimiento y las locales sólo pueden usarse en el procedimiento en el que se declaran. Es buena costumbre usar variables locales en vez de globales.

Podemos declarar más de una variable en una sola línea:

int x, y;

En C++, las variables pueden ser declaradas en cualquier lugar dentro de un programa. No es necesario, como en C, que sean declaradas al comienzo de una función o de un bloque. Esto puede ser útil en códigos grandes, cuando una variable se utiliza en un trozo de código lejano al comienzo del bloque. En este caso, el declarar el tipo de la variable cerca del lugar donde se va a utilizar puede hacer que el programa sea más fácil de leer.

Un ejemplo de esta posibilidad de C++ es la declaración del contador dentro de un bucle. Por ejemplo:

# include

main () {

for (int i=0; i < 10; i++) {

cout << 'hola' << '\n';

}

}

El índice i se ha definido dentro del bucle for. En algunos compiladores, su validez se extiende hasta el final del bloque donde han sido definidos

Constantes.

Las constantes se declaran, como en C, igual que una variable normal, pero añadiendo la palabra **const** delante. Por ejemplo, para declarar una constante con valor 14:

const int numero = 14;

Estas constantes no pueden ser modificadas a lo largo del programa. Por eso deben ser definidas al mismo tiempo que declaradas.

Enumeraciones.

C++ permite definir nombres mnemotécnicos para enteros agrupados en conjuntos. Una enumeración es un conjunto ordenado de nombres:

enum Color {rojo, verde, azul, negro};

Esta declaración define un nuevo tipo, Color, y cuatro variables inalterables de ese tipo. Por ejemplo,

Color c=verde;

declara la variable c del tipo Color con valor inicial verde. Cada nombre en la enumeración se inicializa con un número entero, empezando en 0 e incrementándose en 1 al ir de izquierda a derecha. Se pueden especificar los valores enteros asociados a cada identificador en la enumeración:

enum Poligono {triángulo = 3, cuadrado = 4, pentágono =5};

Declaraciones typedef.

La declaración typedef da un nombre adicional a un tipo ya existente. Por ejemplo:

typedef float temperatura;

hace que temperatura sea un sinónimo de float, pero no define un nuevo tipo, y por tanto no afecta a las reglas de conversión de tipo. y p como variables de distinto tipo, ambas son en realidad de tipo float .

Conversiones de tipo.

**CONVERSIONES ARITMÉTICAS USUALES**

Si los operandos de un operador aritmético binario son del mismo tipo, ese tipo es el tipo del resultado. Pero si no es así uno de los operandos es convertido al tipo del otro, y el resultado de la operación tiene ese tipo común. El tipo común es el tipo del operando que aparece primero en la lista long double, double, float, long int, int. Losoperandos de tipo char y short son tratados como si fueran de tipo int

**CONVERSIONES FORZADAS POR EL PROGRAMADOR**

Las conversiones de tipo se hacen en C++ igual que en C:

c = (int)a;

En C++ las conversiones de tipo se pueden escribir también como una llamada a función:

c = int(a);

|  |  |
| --- | --- |
| **Operación** | **Acción** |
| x++ | Postincremento |
| ++x | Preincremento |
| x-- | Postdecremento |
| --x | Predecremento |
| +x | + unario |
| -x | - unario |
| x\*y | Multiplicación |
| x/y | División |
| x%y | Módulo |
| x+y | Suma |
| x-y | Resta |