Práctica Nº 3: Tipos de datos simples. Constantes y variables. Operadores aritméticos. Formato de salida.

Objetivos de la práctica:
- Mostrar la sintaxis de los diferentes tipos de datos presentes en C/C++.
- Definición de constantes y su diferencia con las variables presentes en los programas.
- Mostrar el funcionamiento de operadores aritméticos entre variables.
- La conversión de tipos y la conversión forzada.
- Iniciación al formato de salida con C++.

TIPOS DE DATOS SIMPLES

- **Tipo carácter:** Almacenan un carácter.
  - Sintaxis: **`char`** 1 byte (-128..0..127)

- **Tipo entero:** Almacenan números enteros.
  - Sintaxis: **`short`** 2 bytes (-32768..0..32767)
  - Sintaxis: **`long`** 4 bytes (-2147483648..0..2147483647)
  - Sintaxis: **`int`** 2-4 bytes(MSDOS: -32768..0..32767
    Linux: -2147483648..0..2147483647)

Modificador ***`unsigned`****: El rango de representación comienza en cero.*
  - Sintaxis: **`unsigned char`** 1 byte (rango 0-255)
  - Sintaxis: **`unsigned short`** 2 bytes (rango 0-65535)
  - ...

- **Tipo real:** Almacenan números reales en coma flotante. El de precisión simple (`float`) ocupan 4 bytess entre 3.4E-38 y 3.4E+38, y el de doble precisión (`double`) que ocupan 8 bytes en memoria y su rango oscila entre 1.7E-308 y 1.7E+308.
  - Sintaxis: **`float`** (simple precisión) 4 bytes (3.4E-38...3.4E+38)
  - Sintaxis: **`double`** (doble precisión) 8 bytes (1.7E-308..1.7E+308)

- **Tipo puntero**: Almacena valores de direcciones de memoria.

- **Tipo bool**: Este tipo existe en C++ pero no en C. Un dato de tipo `bool` produce uno de los dos valores: ***`true`*** (verdadero) *o* `false` (falso). Las expresiones booleanas se emplean en las instrucciones de bifurcación y ciclos que veremos más adelante.
  - Sintaxis: **`bool`**

- **Sin valor:**
  - Sintaxis: **`void`**

Cualquier otro tipo de dato más complejo es una agrupación de estos datos simples unidos para formar el nuevo tipo.

CONSTANTES Y VARIABLES

Constantes

Las constantes son aquellos valores fijos que no se pueden alterar durante la ejecución del programa.

Tipo de dato	Constante de ejemplo	Tipo de dato	Constante de ejemplo
Carácter	**'a' 'b' 'A' '$' '&'**	*Entero largo*	**3500 -4899090**
Cadena	**"hola Juan" "p"**	*Entero largo sin signo*	**3400007 65**
Entero	**3 123 -12340**	*Real*	**5.6 3.56E-12 -123**
Entero sin signo	**3 8923 65520**	*Doble precisión*	**3.14156 -0.8989999E100**

Las constantes de tipo carácter se encierran entre comillas simples (**' '**)
Las constantes de tipo cadena entre comillas dobles (**" "**)

El C++ permite definir símbolos e identificarlos con un valor constante.

- Sintaxis: **`const NOMBRE_TIPO NOMBRE_CTE= valor;`**

Ejemplo:

```
const char L='A';          /* define la  cte. L como un carácter de valor 'A' */
const int  I= 5;           /* define la cte. I como un entero de valor 5 */
const float PI= 3.14159;   /* define la cte. PI con un real de valor 3.14159 */
```

Variables

Las variables se declaran con el fin de almacenar valores que son alterados durante la ejecución del programa. Se puede pensar en ellas como posiciones de memoria donde se guardan datos de un tipo especifico y que tienen asociadas un identificador (nombre).

Declaración de variables

- Sintaxis: **`tipo NOMBRE_VBLE1, NOMBRE_VARIABLE2,...NOMBRE_VARIABLEN;`**

Ejemplo:

```
char caracter1;
float altura;
int i,j,k;        Es equivalente a:    int i;
                                       int j;
                                       int k;
```

Inicialización de variables

A una variable se le puede asignar un valor inicial al tiempo que es declarada, de la forma siguiente:

`tipo Nombre_Variable1, Nombre_Variable2 = valor,... Nombre_VariableN;`

Ejemplos:

```
char caracter1 = ´v´;   short i = -3, j , k = 9;    float altura = 7.5;
```

Son equivalentes a:

```
char caracter1;         short i, j , k;             float altura = 7.5;

caracter1 = 'v';        i = -3;                     altura = 7.5;
                        k = 9;
```

Dónde son declaradas las variables

Se puede hacer una separación de los tipos de variables atendiendo al lugar donde son declaradas: dentro de una función y fuera de cualquier función. El lugar delimita el ámbito de la variable; así, una variable declarada fuera de cualquier función es una variable global que puede ser utilizada dentro cualquier función del programa, y una variable declarada dentro de una función será una variable local que sólo podrá ser utilizada dentro de dicha función.

Operadores

Operadores: Símbolos que designan operaciones aritméticas y lógicas, combinados con variables y constantes forman ***expresiones*****.**

Operadores aritméticos simples

- Operador asignación (**=**): sirve para asignar un valor a una variable.
- Operador suma (**+**): realiza la adición de dos valores.
- Operador resta (**-**): realiza la sustracción de dos valores. También se utiliza para cambiar el signo de un valor.
- Operador producto (*****): multiplica dos valores.
- Operador división (**/**): divide dos valores.
- Operador modulo (**%**): devuelve el resto del cociente entre dos valores enteros (sólo se utiliza con datos de tipo entero).
- Operador decremento (**--**): resta uno a la variable sobre la que se aplica. (Sólo sobre variables de tipo entero y carácter).
- Operador Incremento (**++**): suma uno a la variable. (Sólo sobre variables de tipo entero y de tipo carácter).

Los operadores incremento (**++**) y decremento (**--**) pueden preceder o seguir a la variable, produciendo resultados distintos. Cuando se aplica dentro de una expresión precediendo la variable, esta se incrementa/decrementa su valor antes de evaluar la expresión. Si se aplica después, entonces se evalúa primero la expresión y luego se incrementa/decrementa la variable.

Ejemplo:

```
y = 10;                                  y = 10;
x = y++;   // x vale 10 e y vale 11      x = ++y;      /* x e y valen 11*/
```

Operadores aritméticos de asignación compuesta

Combinan en un solo operador, el operador de asignación con otro operador aritmético.

Símbolo	*Ejemplo*	*Significado*	*Símbolo*	*Ejemplo*	**Significado**
+=	**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**			

Orden de prioridad de los operadores aritméticos.
Es el orden en el que se ejecutan los operadores dentro de una expresión. El orden de ejecución comienza por aquellos encerrados entre paréntesis y luego siguen ejecutándose en siguiente orden:

Mayor prioridad ---------->				Menor prioridad
++ --	**-** (menos monario)	*** / %**	**+ -**	**= += -= *= /= %=**

Si existen varios operadores con la misma prioridad se ejecutan primero los situados más a la izquierda
Ejemplo:

```
int coef1 = 9, coef2 = 8;
int x;

x = 3 / 2 + coef1 / 4 + coef2 * coef2 /16;
  /*    1 +    2      +     4          sobre x se guarda el valor 7 */
```

Conversión de Tipos en las expresiones.

Cuando se evalúa una expresión que contiene diferentes tipos de datos, el compilador convierte todos ellos a un tipo único, el de mayor precisión. La conversión se hace operación a operación.

Ejemplo:.

```
char ch;
int i;
float f;
double d;

res = ( ch  /   i )  +  (   f   *   d   )  -  (   f   +   i );
       char    int        float   double        float    int
                                 ⇓
      ( ch  /   i )  +  (   f   *   d   )  -  (   f   +   i );
        int    int        float   double        float    int
                                 ⇓
      ( ch  /   i )  +  (   f   *   d   )  -  (   f   +   i );
          int           double   double        float    int
                                 ⇓
      ( ch  /   i )  +  (   f   *   d   )  -  (   f   +   i );
          int           double   double        float   float
                                 ⇓
      ( ch  /   i )  +  (   f   *   d   )  -  (   f   +   i );
          int                double                 float
                                 ⇓
                 double                          float
                                 ⇓
                               double
```

En el ejemplo anterior cuando se ejecutaba la operación `(f * d)`, veíamos que `f` promocionaba a `double`. Esto no supone que a partir de ese momento la variable `f` cambie de tipo, `f` continuará siendo de tipo `float` en el resto del programa.

Conversión forzada de tipos de datos

Además de la conversión automática, el C ofrece la posibilidad de forzar la conversión de un tipo de datos en otro tipo diferente. Esta conversión forzada es conocida como "casts":

```
(tipo) expresión
```

Su utilidad queda claramente expresada en el siguiente ejemplo:

```
int a = 3, b = 2;
float c;

c = a / b;
```

La expresión asigna a `c` el valor `1.0` en vez del valor `1.5`. Ello se debe a que `a` y `b` son variables de tipo entero, por lo tanto se realiza una división entera con resultado entero (`1`). A continuación ese valor `1` se convierte a real para guardarse en `C`. Si lo que se desea es que la división sea real, debe forzarse la conversión de al menos uno de los operandos:

```
c = (float)a / b;
```

Una vez convertida la variable `a` a `float`, y aplicando las reglas de conversión automática de tipos, la división se convierte en una división real, para dar como resultado final `1.5`.

Salida con formato (`cout`):

En C++ podemos controlar el formato de órdenes que determinan detalles tales como la notación que queremos utilizar para expresar un número real (notación e o notación en formato de punto fijo), o el número de dígitos en punto decimal.

setf: es una abreviatura de *setflags*, que significa establecer banderas. Establecer una bandera significa darle un determinado valor a esa bandera para que escriba las salidas en ese flujo de una manera específica.	**cout.setf (ios::fixed);**	Hace que el flujo envíe a la salida los números en formato de punto fijo (no notación e).
	cout.setf (ios:: showpoint);	Le dice al flujo que siempre incluya un punto decimal en los números en punto flotante.
precision: Fijar el número de cifras después de la coma (fijar la precisión del número real).	**cout.precision(2);**	Mostrará los números reales con dos decimales.
unsetf: Cualquier bandera establecida se puede desactivar con esta opción.	**cout.unsetf(ios::fixed);**	Al deshabilitar la bandera "fixed", los números reales se imprimirán en notación e.
Nota: Una vez establecida una bandera o valor se mantiene en todo el programa a no ser que se utilice la función *unsetf* o se varíe el valor con el método adecuado.		