Objetivos de la práctica:
- Conocer los límites de representación de los tipos de datos simples.
- Realizar programas de cálculo sencillo utilizando operadores aritméticos
- Utilizar funciones básicas de entrada salida con formato.

Tipos de Datos simples y sus límites de representación

Es importante fijarse en los rangos de representación que tienen los tipos de datos ya que pueden inducir a errores de funcionamiento del programa difíciles de detectar.

- **Tipo carácter:** Almacenan un carácter.
  - Sintaxis: **`char`** 1 byte (-128..0..127)
- **Tipo cadena**: Almacena una cadena de caracteres
  - Sintaxis: **string** Ocupa en memoria tantos bytes como nº de caracteres.
- **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 (no los vamos a usar).
- **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" "Pedro."`	*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/C++ permite definir símbolos e identificarlos con un valor constante.

- Sintaxis: **`#define NOMBRE_CTE valor`**

Ejemplo:

```
#define  L   'A'          /* define la  cte. L como un carácter de valor 'A' */
#define  I   5            /* define la cte. I como un entero de valor 5 */
#define  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_Variable, Nombre_Variable2,... Nombre_VariableN;`

Ejemplos:

```
char caracter1;          int i,j,k;          float altura;
```

Fijarse que:

`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;
```

Esto es equivalente a realizar lo siguiente:

```
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.
Durante este curso, NO ESTÁ PERMITIDO el uso de variables globales.

Operadores

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

Operadores Aritméticos

Los operadores aritméticos simples son:

- 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).

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 */
```

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

```
y = 10;
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

++ --
- (menos unario)
* / %
+ -
=, +=, -=, *=, /= , %=

Es necesario tener esa prioridad en cuenta a la hora de escribir expresiones aritméticas. Si se desea cambiar la precedencia y forzar operaciones se deben utilizar paréntesis.
Si existen varios operadores con la misma prioridad se ejecutan primero los situados más a la izquierda

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.
Por tanto, otro elemento importante a recordar es que los operadores trabajan de distinta forma en función de los tipos de datos a los que se enfrentan. Es necesario tener esto muy en cuenta ya que puede inducir a errores. Por ejemplo la división de dos enteros siempre será un entero. Para evitar estos problemas podemos recurrir a lo que se denomina conversión de tipos o *casting.*

Conversión forzada de tipos de datos

Además de la conversión automática, el C/C++ ofrece la posibilidad de forzar la conversión de un tipo de datos en otro tipo diferente. Esta conversión forzada es conocida como "casting" y se realiza poniendo delante del dato a cambiar, y entre paréntesis, el tipo al cual queremos cambiarlo:

```
(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`.

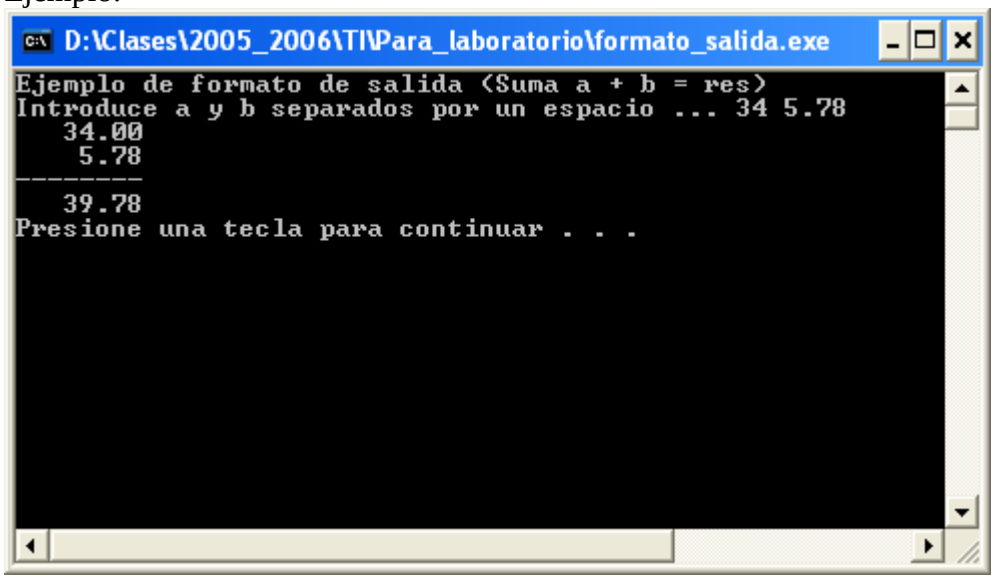
Salida con formato (cout):

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.

Mediante estos modificadores de formato de *cout*, podemos controlar ciertas características de la salida:

Modificador de formato	**Resultado**
cout.precision(N° Cifras)	La precisión decimal se limita a las cifras indicadas. (Fija el número de cifras detrás de la coma)
cout.width(N° caracteres)	Controla el tamaño mínimo de la salida (es decir, cuantos espacios ha de usar al enviar un elemento a la salida). Solamente se aplica a la siguiente salida cout.
cout.fill(carácter)	Indica el carácter con el que se completa una salida que no llega al mínimo de longitud indicado en width.
cout.setf(ios::fixed)	Activamos la "bandera" que indica el formato de punto fijo para los números.
cout.setf(ios::showpoint)	Activamos la "bandera" que indica que incluya un punto decimal en números en coma flotante. (Por defecto está activada).
cout.setf(ios::showpos)	Activamos la "bandera" que indica que aparezca el signo + para números positivos.
cout.setf(ios::left)	Activamos la "bandera" para que el siguiente número aparezca en el extremo izquierdo.
cout.setf(ios::right)	Activamos la "bandera" para que el siguiente número aparezca en el extremo derecho.
cout.unsetf(ios::showpos)	Desactiva la "bandera" del signo +.

Ejemplo:




```
#include <iostream>
#include <stdlib.h>
using namespace std;
int main(void)
{
   float a, b, res;

   cout << "Ejemplo de formato de salida ";
   cout << "(Suma a + b = res) " ;
   cout  << endl;

   cout << "Introduce a y b separados por un espacio ... " ;
   cin >> a >> b;

   res = a + b;

   cout.precision(2);
   cout.setf(ios::fixed);
   cout.setf(ios::right);

   // cout.setf(ios::showpoint); … por defecto
   cout.width(8); cout.fill(' '); cout << a << endl;
   cout.width(8); cout.fill(' '); cout << b << endl;
   cout << "--------" << endl;
   cout.width(8); cout.fill(' '); cout << res << endl;
   system("pause");
   return 0;
}
```

En entrada y salida de caracteres podemos utilizar un par de variantes **cin.get(variable_char)** o **cout.put(variable_char).** Estos manejan de manera diferente los caracteres separadores como espacios o secuencias de escape.
La función **get** permite leer un carácter de entrada y guardarlo en una variable tipo *char* y la función **put** envía a la salida un carácter que está almacenado en la variable tipo *char*.

cin.ignore( ) ignora un carácter. En general **cin.ignore(int num, int delimitador)** ignora un número ***num*** de caracteres mientras no encuentra el carácter ***delimitador***.

Nota: El Dev-C++ pone por defecto el #include <stdlib.h>, para poder usar luego al final la sentencia: system ("pause"); Esta sentencia permite que el programa se pare y espere que pulses una tecla antes de cerrarse la ventana de ejecución del MS-DOS.

BLOQUE DE TIPOS DE DATOS

1. Prueba el siguiente programa, si te parece que no funciona correctamente modifícalo hasta que funcione como debería.

```
#include <iostream>
using namespace std;

int  main(void)
{
        short c1=20;
        short c2=5000;
        short c3;

        c3=c1*c2;
        cout << c1 <<"X"<<c2<<"=" <<c3<<endl;

        return 0;
}
```

BLOQUE DE CÁLCULO

2. Prueba el siguiente programa, si te parece que no funciona correctamente modifícalo hasta que funcione como debería.

```
#include <iostream>
using namespace std;

int main(void)
{
        short c1=3;
        short c2=2;
        float c3;

        c3=c1/c2;
        cout << c1 <<"entre "<<c2<<"=" <<c3<<endl;

        return 0;
}
```

BLOQUE ENTRADA SALIDA

3. Escribe un programa para calcular la suma de tres números reales con dos cifras decimales y los sume. Mostrar el resultado en el siguiente formato:

```
Introduce 3 números reales: 3.13 23.568 100.981
    003.14
    023.57
    100.98
 -----------
    127.69
```

4. Escribe un programa que lea 5 pares de valores formados por un carácter y un número real separados por coma, ejemplo: A,65.4 y los saque por pantalla organizados de la siguiente forma:

Letras	Numeros
A	###65.40
C	#2345.56
D	####4.78

5. Observa el comportamiento de este programa cuando la entrada que pretendemos introducirle es la siguiente:

'A' 'B' 'C'

```
#include <iostream>
using namespace std;

int main()
{
        char letra1,letra2, letra3;

        cout << "Introduce tres letras separadas por un espacio en blanco  \n";
        cout << "Y despues pulsa INTRO \n";
        cin.get(letra1);
        cin.get(letra2);
        cin.get(letra3);

        cout << "Las letras leidas son : ";
        cout << letra1 << letra2 << letra3;
        return 0;
}
```

¿Tiene el comportamiento que esperabas? ¿Por qué? Modifícalo para que funcione correctamente.

PROGRAMAS A REALIZAR:

[precision.cpp] Escribe un programa que te pida un número real, y muestre por pantalla el número real sucesivas veces con distintas precisiones. Una salida por pantalla:

6
6.0
6.01
6.012
6.0123
6.01234
6.012345

[intercambio.cpp] Realizar un programa que intercambie los valores de dos variables numéricas que se introducen por teclado.

[formulas.cpp] Escribe un programa que pide tres números por teclado a, b, c y d. Calcula las siguientes fórmulas matemáticas con estos valores y muestra los resultados por pantalla.

$$b^2 - 4ac \qquad a(b+d) \qquad \frac{1}{a^2+a+3} \qquad \frac{a+b}{c+d}$$

Realiza el programa usando variables enteras y luego haz lo mismo con variables reales. ¿Hay alguna diferencia en los resultados? Explica por qué.

[kilometros.cpp] Realizar un programa que introduciendo una longitud en kilómetros nos devuelva su equivalente en millas. (1 milla = 1.609 km).

[sector.cpp] Escribe un programa que calcule el área total o parcial (área del sector circular) de una circunferencia.

Sector circular

$$A = \frac{\pi R^2}{360} n$$