Objectius de la pràctica:
- Conèixer els límits de representació dels tipus de dades simples.
- Realitzar programes de càlcul senzill mitjançant operadors aritmètics.
- Utilitzar les funcions bàsiques d'entrada i eixida amb format.

Tipus de dades simples i els seus límits de representació

És important tindre en compte els rangs de representació que tenen els tipus de dades, ja que poden induir a errors de funcionament del programa difícils de detectar.

- **Tipus caràcter:** Emmagatzema un caràcter.
  - Sintaxi: `char` 1 byte (-128..0..127)

- **Tipus cadena:** Emmagatzema una seqüència de caràcters.
  - Sintaxi: `string` Ocupa tants bytes com el nº de caràcters

- **Tipus enter:** Emmagatzema nombres enters.
  - Sintaxi: `short` 2 bytes (-32768..0..32767)
  - Sintaxi: `long` 4 bytes (-2147483648..0..2147483647)
  - Sintaxi: `int` 2 bytes (-32768..0..32767)
    4 bytes (-2147483648..0..2147483647)
  - Sintaxi: `long long` 8 bytes (-9223372036854775808..0..9223372036854775807)

- *Modificador* `unsigned`: El rang de representació comença en zero.
  - Sintaxi: `unsigned char` 1 byte (0..255)
  - Sintaxi: `unsigned short` 2 bytes (0..65535)
  - Sintaxi: `unsigned long` 4 bytes (0.. 4294967295)
  - Sintaxi: `unsigned long long` 8 bytes (0..18446744073709551615)

- **Tipus real:** Emmagatzema nombres reals en coma flotant.
  - Sintaxi: `float` (simple precisió) 4 bytes (3.4E-38..3.4E+38)
    7 xifres
  - Sintaxi: `double` (double precisió) 8 bytes (1.7E-308..1.7E+308)
    15 xifres
  - Sintaxi: `long double` (double precisió) 12 bytes (1.18E-4932..1.18E+4932)
    18 xifres

- **Tipus punter:** Emmagatzema valors corresponents a adreces de memòria (no l'emprarem).

- **Sense valor:**
  - Sintaxi: `void`

Qualsevol altre tipus de dades més complex és una agrupació d'aquestes dades simples unides per a formar un nou tipus.

Constants

Les constants són aquells valors fixos que no es poden modificar al llarg de l'execució del programa.

Tipus de dades	**Constant d'exemple**	**Tipus de dades**	**Constant d'exemple**
Caràcter	'a' 'b' 'A' '$' '&'	Enter llarg	3500 -4899090
Cadena	“hola Joan” “p” “Pere.”	Enter llarg sense signe	3400007 65
Enter	3 123 -12340	Real simple precisió	5.6 3.56E-12 -123
Enter sense signe	3 8923 65520	Real doble precisió	3.14156 -0.8989999E100

Les constants de tipus caràcter es representen entre cometes senzilles (' ')
Les constants de tipus cadena es representen entre cometes dobles (“ “)
C/C++ permet definir símbols e identificar-los amb un valor constant.

- Sintaxi: `#define NOM_CONSTANT valor`

Exemples:

```
#define L  'A'       /* L és una constant amb el valor 'A' */
#define I  5         /* I és una constant de valor enter 5 */
#define PI 3.14159   /* PI és una constant de valor 3.14159 */
```

Variables

Les variables es declaren amb l'objectiu d'emmagatzemar valors que són alterats al llarg de l'execució del programa. Hom pot pensar en elles com posicions de memòria on es guardaran dades d'un tipus específic i que tenen associades un identificador textual, açò és, el seu nom.

Declaració de variables

- Sintaxi: `tipus nomVariable1, nomVariable2,..., nomVariableN;`

Exemples:

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

Adoneu-vos que `int i, j, k;` és equivalent a:

```
int i;
int j;
int k;
```

Inicialització de variables

A una variable se li pot assignar un valor inicial al mateix temps que és declarada, de la manera següent: `tipus nomVariable1, nomVariable2 = valor,..., nomVariableN;`

Exemples:

```
char caracter1 = 'v';    int i = -3, j, k = 9;
float llargaria = 7.5;
```

Açò és equivalent a:

```
char caracter1;          int i, j, k;        float llargaria;
caracter1 = 'v';         i = -3;             llargaria = 7.5;
                         k = 9;
```

Abast de les variables

Es pot fer una classificació dels tipus de variables segons el lloc en què es declaren: dins d'una funció o fora de qualsevol funció. El lloc delimita l'àmbit de la variable. Així, doncs, si una variable

ha estat declarada fora de qualsevol funció, es tracta d'una variable global que pot ser emprada des de qualsevol funció del programa. Pel contrari, una variable declarada dins d'una funció serà una variable local que sols podrà ser utilitzada des de dins d'aquesta funció.

En aquesta assignatura, NO ESTÀ PERMÈS l'ús de variables globals.

Operadors

Els operadors són símbols que designen operacions aritmètiques i lògiques i es combinen amb variables i constants per a formar ***expressions***.

Operadors aritmètics

Els operadors aritmètics simples són:

- Operador assignació (=): serveix per a assignar un valor a una variable.
- Operador suma (+): realitza l'addició de dos valors.
- Operador resta (-): realitza la subtracció de dos valors. També s'empra per a canviar el signe d'un valor.
- Operador producte (*): multiplica dos valors.
- Operador divisió (/): divideix dos valors.
- Operador mòdul (%): torna la resta de la divisió entera entre dos valors (sols s'aplica a dades de tipus enter).
- Operador decrement (--): resta un a la variable sobre la que s'aplica (sols sobre variables de tipus enter i caràcter).
- Operador increment (++): suma un a la variable (sols sobre variables de tipus enter i caràcter).

Exemples:

```
int coef1 = 9, coef2 = 8;
int x;
x = 3 / 2 + coef1 / 4 + coef2 * coef2 / 16;
/*    1   +       2   +           4          x guardarà un 7*/
```

Els operadors increment (++) i decrement (--) poden precedir o anar darrere de la variable però produiran resultats diferents. Quan s'aplica dins d'una expressió precedint la variable, aquesta s'incrementa/decrementa abans d'avaluar l'expressió. Si s'aplica després, primer s'avalua l'expressió amb el valor actual de la variable i, en acabant, s'incrementa/decrementa el valor de la variable.

Exemples:

```
y = 10;                                  y = 10;
x = y++; /* x val 10 i y val 11 */       x = ++y; /* x i y valen 11*/
```

Operadors aritmètics d'assignació composta

Combinen, en un sol operador, l'operador d'assignació amb un altre operador aritmètic.

Símbol	Exemple	Significat	Símbol	Exemple	Significat
+=	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			

Ordre de prioritat dels operadors aritmètics
És l'ordre en què s'executen els operadors dins d'una expressió. L'ordre d'execució comença per aquells tancats entre parèntesis i després segueix en l'ordre següent:

Major prioritat ++ --
↑ - (menys unari)
| * / %
| + -
| = += -= *= /= %=
Menor prioritat

Si existeixen diversos operadors amb la mateixa prioritat s'executen primer aquells que es troben més a l'esquerra.
Cal tenir aquesta prioritat en compte quan s'escriuen expressions aritmètiques. Si es desitja canviar la precedència i forçar les operacions s'han de fer servir els parèntesis.

Conversió de tipus en les expressions

Quan s'avalua una expressió que conté diversos tipus de dades, el compilador els converteix tots a un tipus únic: aquell que tinga major precisió. La conversió es duu a terme operació per operació.

Exemple:
```
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       float  float
                           ↓
      ( ch / i ) + (  f   *   d ) - (  f   +  i );
          int           double            float
                           ↓
                double                     float
                           ↓
                          double
```

En l'exemple anterior, quan s'executa l'operació `(f * d)`, f passava a `double`. Açò no suposa que d'ara endavant la variable `f` canvie de tipus, ans al contrari, `f` continuarà essent de tipus `float` en la resta del programa.

Llavors, un altre element que convé recordar és que els operadors treballen de manera diferent en funció del tipus de dades amb què operen. És necessari tindre en compte aquest aspecte perquè pot dur a errors. Per exemple, la divisió de dos enters sempre serà un enter. Per a evitar aquests problemes cal recórrer a la conversió de tipus (també coneguda com *casting*).

Conversió forçosa de tipus de dades

A més a més de la conversió automàtica que mostrava l'apartat anterior, C/C++ ofereix la possibilitat de forçar la conversió d'un tipus de dades en un altre diferent. Aquesta conversió forçosa rep el nom anglosaxó de "casting" i es realitza quan posem davant de la dada a canviar, entre parèntesis, el tipus de dades en el qual volem convertir-lo. Per exemple: `(tipus) expressió.`

La seua utilitat queda clarament expressada en el codi següent:

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

L'expressió anterior assigna a la variable `c` el valor 1.0 en comptes del valor 1.5, ja que `a` i `b` són variables enteres i l'operador duu a terme una divisió entera amb resultat 1. A continuació, aquest valor es converteix a real per a guardar-lo en `c`, però els decimals ja no s'hi troben. Si el que es desitja és que la divisió siga real, cal forçar la conversió de almenys un dels operands:

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

En convertir la variable a a `float`, per aplicació de la conversió automàtica de tipus, la divisió es converteix en una divisió real que donarà 1.5 com a resultat final.

Eixida amb format (`cout`)

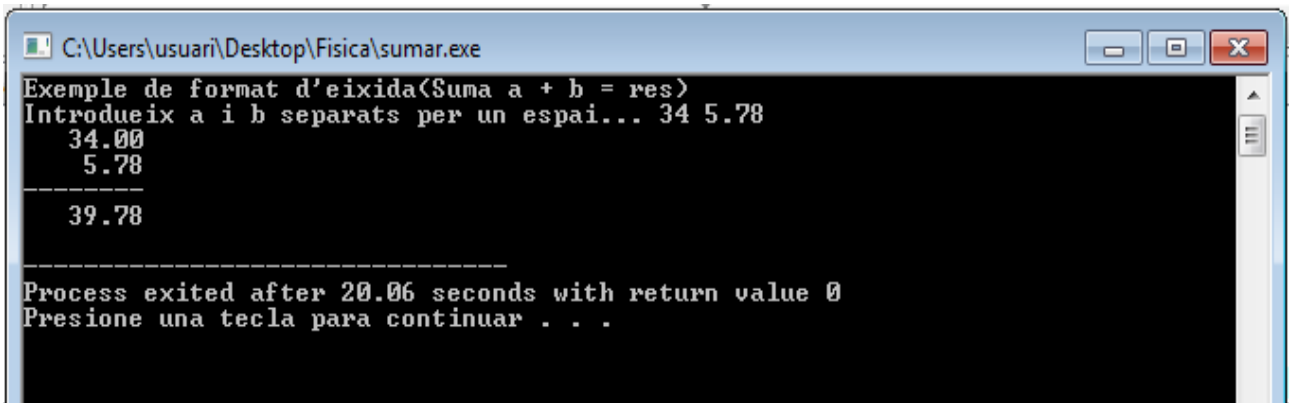
En C/C++ podem controlar el format d'eixida amb un conjunt d'ordres que determinen detalls com ara la notació que volem emprar per a mostrar un valor real (notació científica o notació on format de punt fix) o el nombre de dígits en punt decimal. Mitjançant aquests modificadors de format de `cout`, podem controlar certes característiques de l'eixida:

Modificador de format	**Resultat**
`cout.precision(nXifres)`	Fixa el nombre de xifres darrere de la coma. La precisió decimal es limita a les xifres indicades.
`cout.width(nCaracters)`	Estableix la grandària mínima de l'eixida, açò és, quants espais ha d'ocupar cada element enviat a l'eixida. Només té validesa per a la següent operació d'eixida.
`cout.fill(caracter)`	Indica el caràcter amb què s'ha de completar una eixida que no arriba al mínim de la llargària establida amb el modificador "width".
`cout.setf(ios::fixed)`	Activa el mode que mostra els nombres en format de punt fix.
`cout.setf(ios::showpoint)`	Activa el mode que mostra els nombres en coma flotant amb un punt decimal (per defecte està activat).
`cout.setf(ios::showpos)`	Activa el mode que mostra el símbol +. en els nombres positius.

`cout.setf(ios::1eft)`	Activa el mode que fa que el següent nombre aparega alineat a l'esquerra dins de la grandària definida amb el modificador "width".
`cout.setf(ios::right)`	Activa el mode que fa que el següent nombre aparega alineat a la dreta dins de la grandària definida amb el modificador "width".
`cout.unsetf(ios::showpos)`	Desactiva el mode que mostra el símbol + per als nombres positius.

Exemple:

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

int main(void)
{
  float a, b, res;

  cout << "Exemple de format d'eixida";
  cout << "(Suma a + b = res)";
  cout << endl;

  cout << "Introdueix a i b separats per un espai... ";
  cin >> a >> b;

  res = a + b;

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

  //cout.setf(ios::showpoint) // per defecte
  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;

  return 0;
}
```



A més a més de les opcions vistes fins ara, per a l'entrada i eixida de caràcters també podem emprar les variants `cin.get(variableChar)` i `cout.put(variableChar)`. Aquestes instruccions gestionen de manera diferent els caràcters separadors o les seqüències d'escapament.

La funció get permet llegir un caràcter d'entrada i guardar-lo en una variable de tipo `char`. La funció put envia a l'eixida un caràcter que està emmagatzemat en una variable de tipo `char`.

La instrucció `cin.ignore()` ignora un caràcter. La variant `cin.ignore(int n, int delimitador)` ignora un nombre de caràcters (n) mentrestant no trobe el caràcter delimitador.