

Informació varia

Llibreries i tal:

```
using namespace std;
#include <iostream>
#include <vector>
#include <string>
#include <cmath>
#include <algorithm>
```

<code>pro2/session1> p2++ -c PRO2.cc</code>	Crea PRO2.o
<code>pro2/session1> g++ -o PRO2.exe PRO2.o</code>	Crea PRO2.exe
<code>pro2/session1> ./PRO2.exe</code>	Executa PRO2.exe

Ho podem executar tot en una sola línia així:

```
p2++ -o PRO2.exe PRO2.cc && ./PRO2.exe
```

Si volem executar `PRO2.exe` amb les dades del fitxer `PRO2.in`:

```
./PRO2.exe < PRO2.in
```

Si a més volem que els resultats s'escriguin en un altre fitxer `PRO2.out`, executarem:

```
./PRO2.exe < PRO2.in > PRO2.out
```

Si volem comprimir un fitxer.tar: (`cf` → create file)

```
tar cf nom.tar fitxer1.cc fitxer2.hh fitxer3.exe...
tar cf nom.tar *
'*' → Vol dir tots els fitxers del directori
```

Si volem extreure un fitxer.tar: (`xf` → extract file)

```
tar xf nom.tar
```

Strings

A part d'agafar dades i imprimir-les per la terminal amb `cin` i `cout` també podem:

```
#include <string>
#include <sstream>
```

```
cin.ignore(); Ignora la resta d'entrades de la línia i salta a la següent
```

```
string line;
getline(cin, line); Dona a line el valor de la línia sencera com a únic string
stringstream ss(line); Permet tractar ss com si fossin strings de un cin
while (ss >> x) { Llegim ss separant-lo pels espais
    cout >> x; x tindrà l'element de la line corresponent a la seva iteració
}
```

Piles:

Emmagatzema elements, **només puc inserir i extreure per l'inici**

<code>#include <stack></code>	
<code>stack<tipus_dada> nom_pila;</code>	
<code>pila.push(variable);</code>	Afegeix variable dalt la pila
<code>pila.pop();</code>	Treu el que hi hagi dalt la pila
<code>pila.empty();</code>	Retorna true si està buida
<code>pila.top();</code>	Retorna la variable que hi hagi dalt la pila
<code>pila.size();</code>	Retorna la mida de la pila

Cues:

Emmagatzema elements, **només puc inserir al final i extreure a l'inici**

<code>#include <queue></code>	
<code>queue<tipus_dada> nom_cua;</code>	
<code>cua.push(variable);</code>	Afegeix variable a la cua
<code>cua.pop();</code>	Treu el que hi hagi primer a la cua
<code>cua.empty();</code>	Retorna true si està buida
<code>cua.front();</code>	Retorna la variable de l' inici de la cua
<code>cua.size();</code>	Retorna la mida de la cua

Iteradors:

Punter per recórrer qualsevol format de dades

```
estructura<tipus_dada>::iterator it;
```

Per exemple: `list<int>::iterator it;`

També podem usar: `auto it;`

```
llista.begin()
```

Retorna un **iterador** apuntant a l'**inici**

```
llista.end()
```

Retorna un **iterador** apuntant al **final**

```
++it;
```

Accedeix al següent element

```
advance(it, numero);
```

Avança numero elements

```
llista.insert(it, variable);
```

Afegeix **variable** a la posició apuntada per **it**

```
llista.erase(it);
```

Treu l'element de la posició **it**, retorna un **iterador** apuntant a la **següent posició**

Llistes:

Emmagatzema elements, **permet inserir i extreure tant per l'inici com pel final**

```
#include <list>
```

```
list<tipus_dada> nom_llista;
```

```
llista.front();
```

Retorna la **variable** que hi hagi a l'**inici**

```
llista.back();
```

Retorna la **variable** que hi hagi al **final**

```
llista.push_front(variable);
```

Afegeix **variable** a l'**inici**

```
llista.push_back(variable);
```

Afegeix **variable** al **final**

```
llista.pop_front(variable);
```

Treu l'element que hi hagi a l'**inici**

```
llista.pop_back(variable);
```

Treu l'element que hi hagi al **final**

```
llista.remove(variable);
```

Treu tots els elements amb el **valor variable**

```
llista.clear();
```

Borra tota la llista

```
llista.size();
```

Retorna la **mida** de la llista

```
llista.sort();
```

Ordena la llista

Pairs:

Emmagatzema dos valors de tipus possiblement diferents

<code>pair<tipus_dada_1, tipus_dada_2> nom_pair;</code>	Pair Buit
<code>pair<int, string> p1(10, "Hola");</code>	Constructora
<code>pair<int, string> p2 = {10, "Adéu"};</code>	Inicialització amb valors
 <code>cout << p1.first << " " << p1.second;</code>	Accés als elements
<code>p1.first = 50;</code>	Modificar 1r element
<code>p1.second = "Hola de nou";</code>	Modificar 2n element
 <code>if (p1 > p2)</code>	Compara primer per first, si són iguals compara per second

Maps (Diccionaris):

Emmagatzema parells {Clau,Valor} de forma ordenada

<code>#include <map></code>	
<code>map<clau_tipus_dada, valor_tipus_dada> nom_mapa;</code>	
<code>map<clau_tipus_dada, valor_tipus_dada>::iterator it;</code>	
<code>mapa.begin()</code>	Retorna un iterador apuntant a l'inici
<code>mapa.end()</code>	Retorna un iterador apuntant al final
 <code>mapa.insert({clau, valor});</code>	Afegeix una parella clau-valor al mapa
<code>mapa.erase(it);</code>	Treu l'element de la posició it , retorna un iterador apuntant a la següent posició
 <code>mapa.at(clau);</code>	Accedeix al valor associat a la clau donada (Si no existeix fa excepció out_of_range)
<code>mapa[clau];</code>	Accedeix al valor associat a la clau donada (Si no existeix la crea)
 <code>mapa.count(clau);</code>	Retorna el nº d'elements amb clau (0 o 1)
<code>mapa.find(clau);</code>	Retorna un iterador apuntant a la clau (si no el troba torna mapa.end())
 <code>mapa.size();</code>	Retorna la mida del mapa
<code>mapa.clear();</code>	Borra tots els elements del mapa

Sets (conjunts):

Emmagatzema de forma **ordenada** una **única vegada** cada **element**

```
#include <set>
set<tipus_dada> nom_conjunt;
set<tipus_dada>::iterator it;
```

```
set.begin()
```

Retorna un **iterador** apuntant a l'**inici**

```
set.end()
```

Retorna un **iterador** apuntant al **final**

```
set.insert(valor);
```

Afegeix un element al conjunt (**si no hi era**)

```
set.erase(valor);
```

Treu un element al conjunt (**si hi era**)

```
set.erase(it);
```

Treu l'element de la posició **it**, retorna un **iterador** apuntant a la **següent posició**

```
set.erase(valor);
```

```
set.count(valor);
```

Retorna 1 si l'element i és 0 si no

```
set.find(valor);
```

Retorna un iterador apuntant al valor
(si no el troba torna set.end())

```
set.size();
```

Retorna la **mida** del conjunt

```
set.clear();
```

Borra tots els elements del conjunt

BinTree:

```
#include "BinTree.hh"
BinTree<tipus_dada> nom_arbre;
```

Per construir un nou arbre:

```
BinTree<tipus_dada> nom_arbre(valor, fillEsquerre, fillDret);
```

<code>BinTree<tipus_dada>();</code>	Retorna un arbre buit
<code>arbre.empty();</code>	Retorna true si l'arbre és buit
<code>arbre.left();</code>	Retorna el subarbre esquerre
<code>arbre.right();</code>	Retorna el subarbre dret
<code>arbre.value();</code>	Retorna el valor del node actual

⚠ Si intenteu accedir al `value()` d'un node buit **petarà**, cal posar el cas base:

```
if (arbre.empty()) return elQueSigui;
```

Un node és una **fulla** si

```
arbre.left().empty() && arbre.right().empty()
```

Modularitat i Makefile

Fitxers d'Encapçalament (.hh):

- Contenen capçaleres de funcions i definicions de tipus de dades.
- Hi ha dos tipus:
 - Fitxers que creen nous tipus de dades (es criden amb `dada.funció()`).
 - Fitxers que contenen únicament la implementació a funcions.

Fitxers de Codi (.cc):

- Contenen la implementació de funcions i mètodes.

Makefile:

```
all: program.exe                                #Objectiu principal makefile

#Construeix l'executable a partir dels fitxers objectes
program.exe: main.o altres.o                    #Crida a main.o i altres.o
    p2++ -o program.exe main.o altres.o

altres.o: altres.cc altres.hh                    #Compila els objectes
    p2++ -c altres.cc -o altres.o

main.o: main.cc llibreries.hh altres2.hh        #Compila main amb llibreries
    p2++ -c main.cc -o main.o

clean:                                           #Esborra els fitxers temporals
    rm -f *.o                                   El cridem amb "make clean"
    rm -f program.exe
```