

Informació varia

Llibreries i tal:

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

```
pro2/session1> p2++ -c PRO2.cc
pro2/session1> g++ -o PRO2.exe PRO2.o
pro2/session1> ./PRO2.exe
```

Crea PRO2.o
Crea PRO2.exe
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.dat:

```
./PRO2.exe < PRO2.dat
```

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

```
./PRO2.exe < PRO2.dat > PRO2.sal
```

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

```
tar cf nom.tar fitxer1.cc fitxer2.hh fitxer3.exe...
tar cf nom.tar *
```

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) {
    cout >> x;
}
```

Llegim ss separant-lo pels espais
x tindrà l'element de la línia corresponent a la seva iteració

Piles:

```
#include <stack>
stack<tipus_dada> nom_pila;
pila.push(variable);
pila.pop();
pila.empty();
pila.top();
pila.size();
```

Afegeix **variable** dalt la pila
Treu el que hi hagi dalt la pila
Retorna **true** si està buida
Retorna la **variable** que hi hagi **dalt** la **pila**
Retorna la **mida** de la pila

Cues:

```
#include <queue>
queue<tipus_dada> nom_cua;
cua.push(variable);
cua.pop();
cua.empty();
cua.front();
cua.size();
```

Afegeix **variable** a la cua
Treu el que hi hagi primer a la cua
Retorna **true** si està buida
Retorna la **variable** de l'**inici** de la **cua**
Retorna la **mida** de la cua

Iteradors:

```
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:

```
#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:

<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):

<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):

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

```
set.begin()
set.end()
```

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

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

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

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

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

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

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

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

Retorna 1 si l'element i és 0 si no

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

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

Retorna la **mida** del conjunt

Borra tots els elements del conjunt

BinTree:

```
#include "BinTree.hh"
BinTree<tipus_dada> nom_arbre;
BinTree<tipus_dada> nom_arbre(valor, fillEsquerre, fillDret);
```

```
arbre.empty();
```

Retorna true si l'arbre és buit

```
arbre.left();
```

Retorna el subarbre esquerre

```
arbre.right();
```

Retorna el subarbre dret

```
arbre.value();
```

Retorna el valor del node actual

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
```