Informació vària

Llibreries i tal:

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

```
pro2/sessio1> p2++ -c PRO2.cc
pro2/sessio1> g++ -o PRO2.exe PRO2.o
pro2/sessio1> ./PRO2.exe
```

Crea PRO2.o
Crea PRO2.exe
Executa PRO2.exe

Ho podem executar tot en una sola línea 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 línea i salta a la següent

```
string line;
getline(cin, line);
```
Dona a line el valor de la línea 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 line corresponent a la seva iteració

Piles:

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

```
#include <stack>
stack<tipus_dada> nom_pila;
```

`pila.push(variable);`	Afegeix variable dalt la pila
`pila.pop();`	Treu el que hi hagi dalt la pila
`pila.empty();`	Retorna true si està buida
`pila.top();`	Retorna la variable que hi hagi **dalt** la **pila**
`pila.size();`	Retorna la mida de la pila

Cues:

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

```
#include <queue>
queue<tipus_dada> nom_cua;
```

`cua.push(variable);`	Afegeix variable a la cua
`cua.pop();`	Treu el que hi hagi primer a la cua
`cua.empty();`	Retorna true si està buida
`cua.front();`	Retorna la variable de **l'inici** de la **cua**
`cua.size();`	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**

```
pair<tipus_dada_1, tipus_dada_2> nom_pair;
```
Pair Buit

```
pair<int, string> p1(10, "Hola");
```
Constructora

```
pair<int, string> p2 = {10, "Adéu"};
```
Inicialització amb valors

```
cout << p1.first << " " << p1.second;
```
Accés als elements

```
p1.first = 50;
```
Modificar 1r element

```
p1.second = "Hola de nou";
```
Modificar 2n element

```
if (p1 > p2)
```
Compara primer per first, si són iguals compara per second

Maps (Diccionaris):

Emmagatzema parells {Clau,Valor} de forma **ordenada**

```
#include <map>
map<clau_tipus_dada, valor_tipus_dada> nom_mapa;
map<clau_tipus_dada, valor_tipus_dada>::iterator it;
```

```
mapa.begin()
```
Retorna un iterador apuntant a **l'inici**

```
mapa.end()
```
Retorna un iterador apuntant al **final**

```
mapa.insert({clau, valor});
```
Afegeix una parella clau-valor al mapa

```
mapa.erase(it);
```
Treu l'element de la posició **it,** retorna un iterador apuntant a la **següent posició**

```
mapa.at(clau);
```
Accedeix al valor associat a la clau donada **(Si no existeix fa excepció out_of_range)**

```
mapa[clau];
```
Accedeix al valor associat a la clau donada **(Si no existeix la crea)**

```
mapa.count(clau);
```
Retorna el nº d'elements amb clau (0 o 1)

```
mapa.find(clau);
```
Retorna un iterador apuntant a la clau (si no el troba torna mapa.end())

```
mapa.size();
```
Retorna la mida del mapa

```
mapa.clear();
```
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

Arbres Binaris:

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

Per construir un nou arbre:
```
BinTree<tipus_dada> nom_arbre(valor, fillEsquerre, fillDret);
```

`BinTree<tipus_dada>();`	Retorna un arbre buit
`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

⚠️ 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()
```

Arbres Generals:

```
#include "tree.hh"
Tree<tipus_dada> nom_arbre;
```

Per construir un nou arbre:
```
vector<Tree<tipus_dada>> fills = {fill1, fill2, fill3…};
Tree<tipus_dada> nom_arbre(valor, fills);
```

`Tree<tipus_dada>();`	Retorna un arbre **buit**
`arbre.empty();`	Retorna **true** si l'arbre és buit
`arbre.num_children();`	Retorna el **nombre de fills**
`arbre.child(i);`	Retorna el **fill i**
`arbre.value();`	Retorna el **valor** del node actual

Per recorre els fills:
```
for (int i = 0; i < t.num_children(); ++i) t.child(i);
```

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
                                                #El que s'executa quan fas make

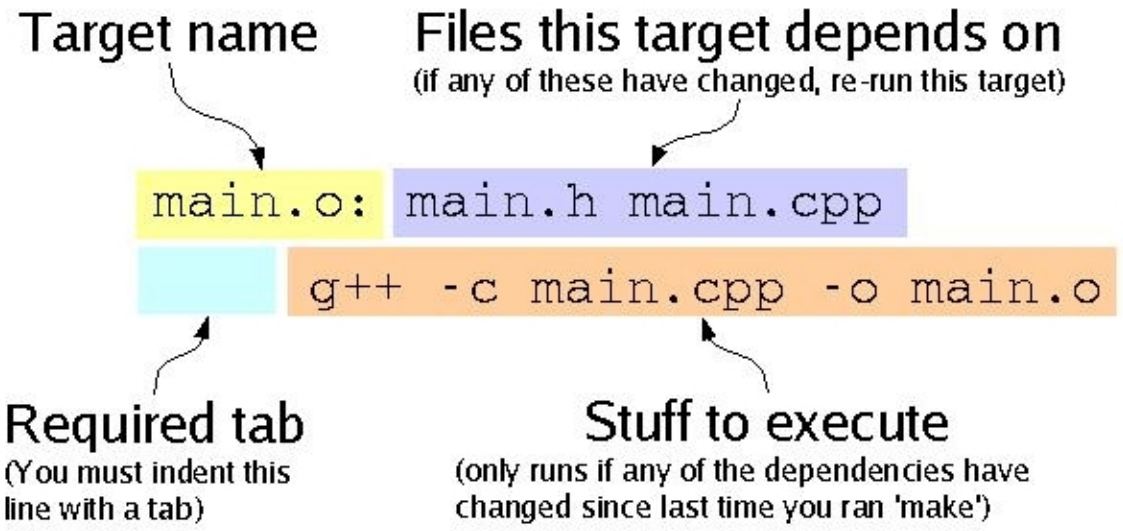
#Construeix l'executable a partir dels fitxers objectes
program.exe: main.o altres.o                    #Crida a main.o i altres.o
	p2++ -o program.exe main.o altres.o

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

tar:
	tar cf *.cc *.hh
```



Classe ‘Stack’ (Punters)

Cada element d’una stack conté un **valor** i un **punter** *apuntant cap el següent element*
Addicionalment la classe stack conté la mida d’aquesta i un punter anomenat ***ptopitem*** apuntant al primer element.

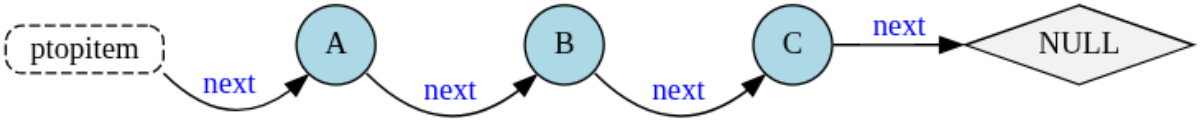


Diagrama Stack

Classe ‘Queue’ (Punters)

Cada element d’una queue conté un **valor** i un **punter** *apuntant cap el següent element*
Addicionalment la classe queue conté la **mida** d’aquesta i dos punters apuntant al primer i últim element anomenats **first** i **last**.

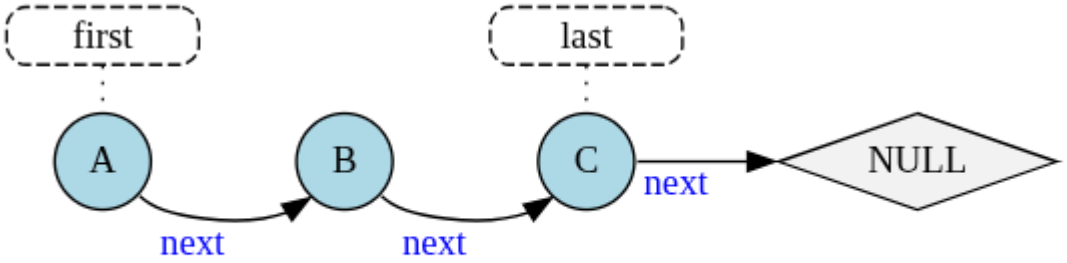


Diagrama Queue

Classe ‘List’ (Punters)

La list és ajuntar la queue i l’stack, cada element d’una queue conté un **valor** i dos **punters** **next** i **prev** *apuntant cap el següent element i cap a l’anterior.* Addicionalment la classe queue conté la **mida** d’aquesta i dos punters apuntant al primer i últim element anomenats **iteminf** i **itemsup**.

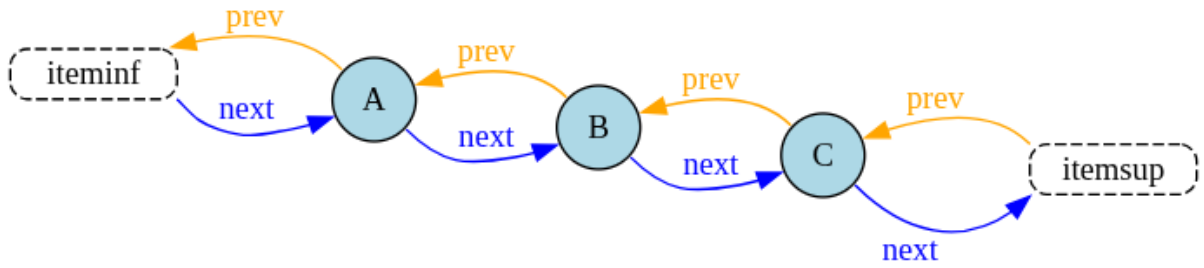
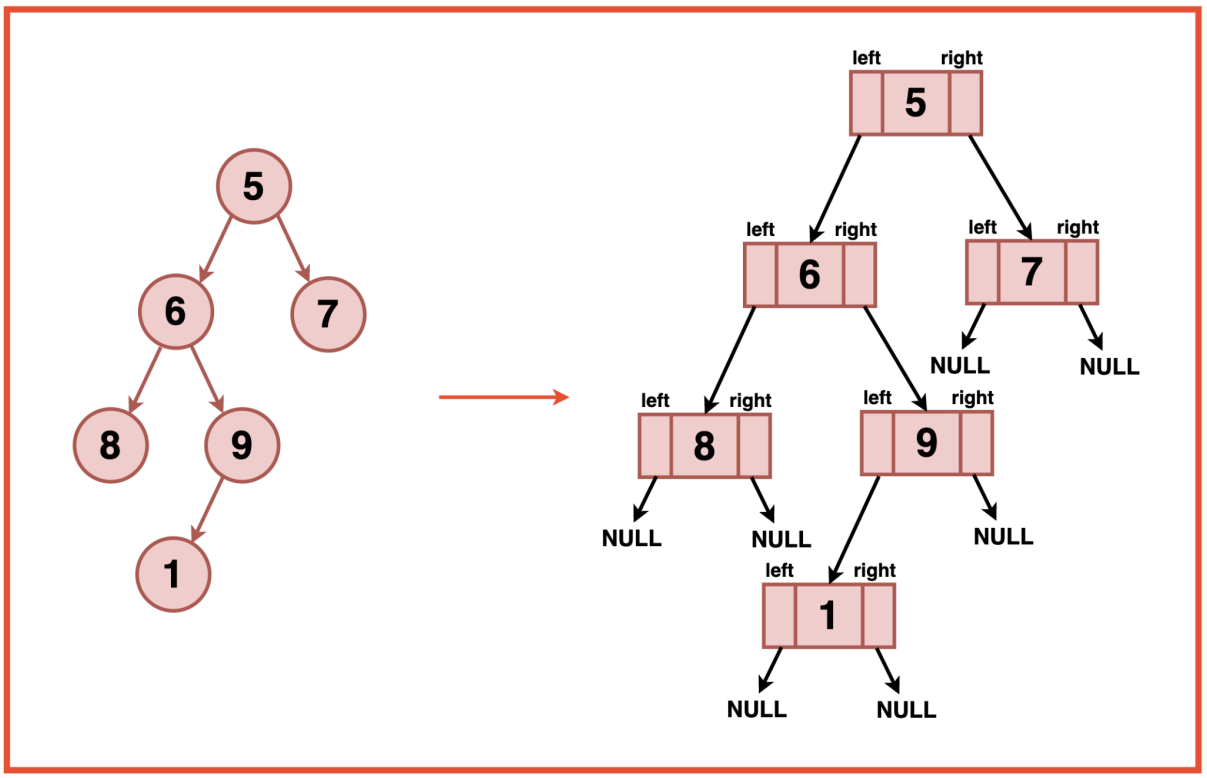


Diagrama Llista

Classe ‘Arbre’ (Punters)

Els arbres a més de contenir el valor (**info**) contenen dos punters cap als seus fills **segE** i **segD** on contenen un altre punter si hi ha un node o NULL si no hi ha res.



Binary Tree Representation in C++ - Tutorial