

**WUOLAH**



**Arnau\_FIB**  
[www.wuolah.com/student/Arnau\\_FIB](http://www.wuolah.com/student/Arnau_FIB)

11191

## **RESUM APUNTS PRO2 - PARCIAL 1.pdf** APUNTES PRO2 - PARCIAL 1



**1º Programación II**



**Grado en Ingeniería Informática**



**Facultad de Informática de Barcelona (FIB)**  
UPC - Universidad Politécnica de Catalunya



**Encontré un perro  
en tus apuntes.**

**WUOLAH**

## PRO 2. ESTRUCTURES LINEALS II : LLISTES

- Contenidor: Emmagatzemar objectes.
- Són template: en necessita un tipus de dada com a paràmetre
- Si usen iterators: apuntador que referencia la pos de mem d'un elem. d'una llista.

`list<int>:: iterator it = l.begin();`

`list<int>:: iterator it2 = l.end();`

Si `list<int>` és buida, `l.begin();` és igual a `l.end();`

Els iterators s'incrementen/decrementen fent `++it` / `--it` únicament.

`list<int>:: const_iterator it` → No permet modificar (`*it`)

No es pot accedir a (`*it`) si `it = l.end();`

Per esborrar: `it = l.erase(it);` Esborra l'element apuntat per `it` i `it` passa a ser el següent element.

Per inserir: `l.insert(it, s);` Introdueix l'element a la posició

on apunta `it`.  $l=[1, 3, 4, 5, 8] \rightarrow l.insert(it, s) \rightarrow l=[1, 5, \underset{\uparrow}{3}, 4, 5, 8]$

Per ajuntar dues llistes:  $l1 = [1, 2, 3, 4, 5, 6], l2 = [10, 20, 30]$

`l1.splice(it, l2); \rightarrow l1 = [1, 2, 3, 10, 20, 30, 4, 5, 6], l2 = []`

si volem concatenar llistes: `l1.splice(l1.end(), l2)`

## ARBRES

(Arbres binaris → BinTree)

Per crear arbres:

BinTree<int> a; crea un BinTree buit

BinTree<int> a(s); crea un BinTree amb node s i sense fills

BinTree<int> a(s, b1, b2); crea un BinTree amb node s; té com a fills els BinTree<int> b1 i b2.

Consultores:

a. empty(); retorna si està buit l'arbre

a. left(); retorna el fill esquerre

a. right(); retorna el fill dret

a. value(); retorna el valor de l'arrel

Recorreguts d'arbres:

• En profunditat:

Preordre:

- 1) Visitar arrel
- 2) Recórrer fill esquerre  
(en preordre)
- 3) Recórrer fill dret  
(en preordre)

Inordre:

- 1) Recórrer fill esquerre  
(en inordre)
- 2) Visitar arrel
- 3) Recórrer fill dret  
(en inordre)

Postordre:

- 1) Recórrer fill esquerre  
(en postordre)
- 2) Recórrer fill dret  
(en postordre)
- 3) visitar l'arrel

• Per nivells:

Es fa amb una cua:

- 1) Agafar primer arbre de la cua
- 2) Visitar la sera arrel
- 3) Ficar els seus dos fills a la cua que no són a la cua

template <typename T>

list<T> nivells (const BinTree& a) {

list<T> l;

if (not a.empty()) {

queue < BinTree<T> > c;

c.push(a);

while (not c.empty()) {

return l;

Bintree<T> aux = c.front(); c.pop(); l.push\_back(aux.value());  
if (not aux.left().empty()) c.push(aux.left());  
if (not aux.right().empty()) c.push(aux.right());

# CORRECTESA DE PROGRAMES ITERATIUS

## Correctesa de programes

Def: Per tots els valors inicials de les variables que satisfan la Pre, el programa acaba i els valors finals satisfan la Post.

Per demostrar que un programa es correcte:

- Raonament genèric → Inducció { Recursiu : Directament.  
Iteratiu : Amagada en l'invariant.

## Estats i assersions

Estat d'un programa: Tupla de valors de tots els variables.

Asserçió: Descripció d'un conjunt d'estats.

- ↳ La Pre és l'asserçió que se suposa que és certa al principi.
- ↳ La Post és l'asserçió que volem que sigui certa al final.

Un programa és correcte si donada una Pre. compleix la Post.

## Correctesa dels programes iteratius (Exemplar al PDF)

Invariant: Una asserció que és certa després de qualsevol nombre d'iteracions. Que una asserció Inv sigui invariant es demostra per inducció.

Sempre caldrà demostrar que el bucle acaba.

Funció de fita: Variable / expressió que indica quantes iteracions queden.

## Disseny inductiu (Exemplar al PDF)

Donades una Pre i una Post, cal proposar:

- Un invariant que generalitzi les dues
- Inicialitzacions que (amb la Pre) assegurin l'invariant
- Un cos del bucle que mantingui l'invariant
- Una condició del bucle que negada impliqui la post ?



¿Puedo quedarme  
este perro?

WUOLAH

## DISENY RECURSiu

- Recursió és inducció
- Funció d'immersió: Afegir més paràmetres
- Immersió d'eficiència: Afegir paràmetres per recordar càlculs.

### Recursió, definicions recursives i inducció

Per aplicar recursivitat, primer cal trobar la definició recursiva del problema que se'n demana.

Exemple: Suma els elements d'una pila  $p$ :

$$\text{Suma}(p) = \begin{cases} 0 & , \text{ si } p \text{ es buida} \\ p.\text{top} + \text{suma}(p.\text{pop}), & \text{altrament} \end{cases}$$

(El codi no accepta  $\text{sum}(p.\text{pop})$ , en posa així "per millor comprensió")

### Principis de disseny recursiu

Cal identificar

- Casos base, podem satisfer la post amb càlculs directes
- Casos recursius, podrem satisfer la post amb paràmetres "més petits".

Cal demostrar: Amb tot valor  $x$  que satisfaci  $\text{Pre}(x)$ , l'algorisme acaba (#finit de criden recursives) i acaba satisfent  $\text{Post}(x)$ .

S'ha de demostrar que cada crida recursiva fa els paràmetres "més petits".

### Esguema de correctesa d'un algoritme recursiu

- Demostrar que  $\forall x \text{ tq } \text{Pre}(x) \text{ cert} \Rightarrow \text{en compleix Post}(x)$ 
  - 1) Si  $x$  cas base  $\Rightarrow$  Demo directa
  - 2) Hipòtesi d'inducció:  $\forall x' \text{ tq } \text{Pre}(x') \text{ cert i } x' < x \Rightarrow \text{en compleix Post}(x')$
  - 3) Si  $x$  cas recursiu  $\Rightarrow$  comprovar que totes les criden a  $x'$  ( $x' < x$ ) compleixen la  $\text{Pre}(x')$
  - 4) Apliquem H.I. per veure que en compleix  $\text{Post}(x')$
  - 5) Veiem que els càlculs ens porten a  $\text{Post}(x)$
- Finalment, comprovem que totes les accions, funcions i mètodes cridats satisfan les precondicions respectives

## Immersió o generalització d'una funció. Immersions per afegiment de la post

Es crea una funció d'immersió quan a la funció original no hi ha paràmetres suficients. La funció original arderà a la funció d'immersió i ignorarà algun dels resultats retornats.

↳ Afegiment de la post & Enfortiment de la pre.

(canviar i per 0,

canviar v.size()-1 per j,...)

• És important dir quina immersió es farà i especificar-la bé.

# MILLORES D'EFICIÈNCIA EN RECURSIÓ I ITERACIÓ

## Eliminació de càlculs repetits

### Iteracions:

- Afegim variables locals
- No apareixen ni a la Pre ni a la Post
- Apareixen al invariant

### Recursió

- Creem nous paràmetres d'entrada i/o sortida (immersió)
- Cal afegir-los a la Pre / Post
- La funció original no és recursiva, caldrà a la immersió

## Immersions d'eficiència (Exemples PDF)

Def: Introducció de paràmetres / resultats addicionals per transmetre valors ja calculats en / a altres crides.

## Eficiència : Consideracions generals

Funcions han de ser eficients en temps i en memòria.

Ex: Amb un vector de mida  $n$ , temps proporcional a :

Cerca seq( $n$ ), Cerca dizot( $\log_2 n$ ), Merge sort( $n \cdot \log_2 n$ ), ...

Més exemplars d'eliminació de càlculs repetits (al PDF)