UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Politècnica Superior d'Enginyeria
de Vilanova i la Geltrú

PUBLICACIÓ DOCENT

FONAMENTS DE PROGRAMACIÓ

Apunts de teoria

AUTOR: Bernardino Casas i Rosa M. Jiménez
ASSIGNATURA: Fonaments de Programació
CURS: Primer
TITULACIONS: Grau en Enginyeria Informàtica
DEPARTAMENT: Ciències de la Computació (CS)

Vilanova i la Geltrú, 17 de setembre de 2021

Índex

Tema 0

Definicions i passos en el disseny

Un viatge de mil milles comença amb el primer pas.

Lao-tsé (570 aC-490 aC)

Fonaments de Programació

Definicions i passos en el disseny

Bernardino Casas

bcasas@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

- ✓ Saber què és un llenguatge de programació, i conèixer-ne alguns exemples.
- ✓ Conèixer els passos que a seguir en el disseny d'un programa.
- ✓ Poder fer algorismes de problemes molt senzills.
- ✓ Ser capaç de traduir un algorisme senzill a un llenguatge de programació (C++).

Índex

Índex

Índex

Què és un algorisme?

Definició 1: Algorisme

Un *algorisme* és un conjunt finit d'instruccions o passos que serveixen per executar una tasca o resoldre un problema.

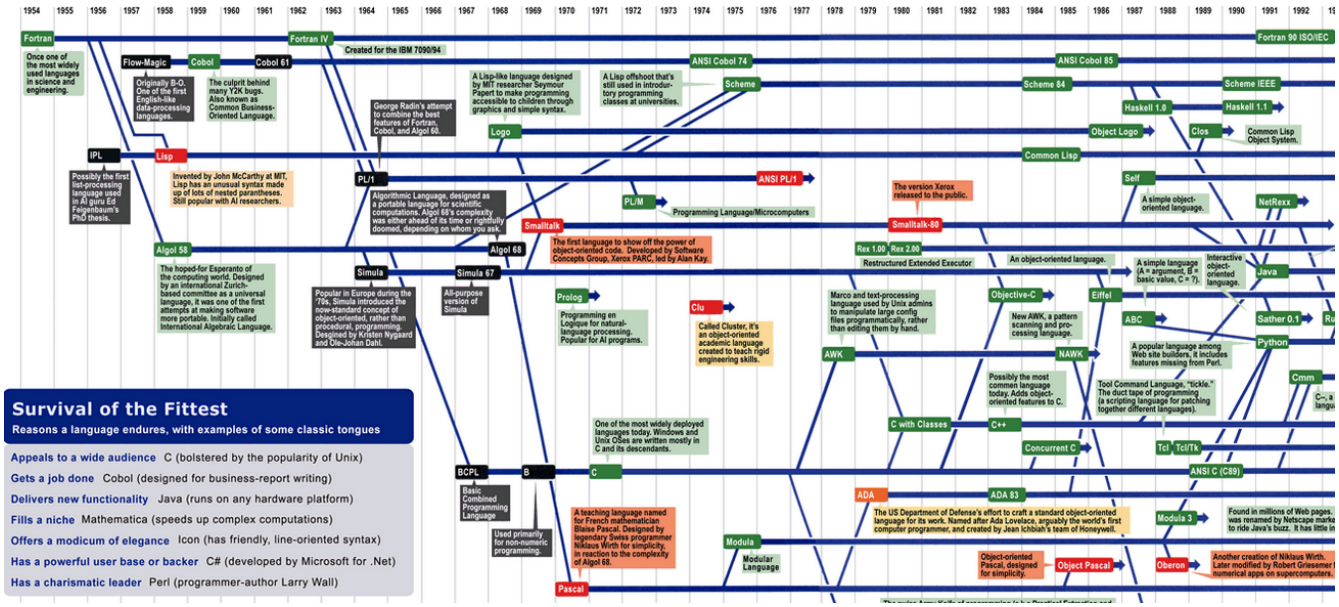
- En la vida quotidiana, s'empren algorismes en multitud d'ocasions per a resoldre diversos tipus de problemes. Per exemple:
  - posar una rentadora.
  - montar un moble d'Ikea®.
  - construir un aeroplà a escala.
  - fer trucs de màgia (passos per a fer el truc).
  - cuinar usant una recepta de cuina (passos de la recepta).

Exemple d'algorisme

- Com es pot saber si un nombre és **primer**?
  - Llegir el nombre (N).
  - Dividir N entre tots els nombres entre 2 i N-1 i calcular el residu de cada divisió.
  - Si tots els residus són diferents de zero, el nombre és primer. En cas contrari, el nombre no és primer.

Índex

Quins llenguatges de programació coneixeu? (I)



Font: `http://www.digibarn.com/collections/posters/tongues/`

Quins llenguatges de programació coneixeu? (II)

- Hi ha moltíssims llenguatges de programació. Possiblement us sonaran: C, Java, Python, C++, PHP, ...
- Cada llenguatge es va idear amb un propòsit concret, i per tant tots els llenguatges tenen virtuds i defectes que cal tenir en compte.
- Un bon programador hauria de conèixer diversos llenguatges de programació per poder triar el més adequat per desenvolupar un projecte.
- En aquest curs aprendrem els conceptes bàsics de programació usant C++. Aquest és un llenguatge de programació de propòsit general basat en C, creat a mitjans de la dècada de 1980 per Bjarne Stroustrup.

Què és un llenguatge de programació?

Definició 2: Llenguatge de programació

Un *llenguatge de programació* és un llenguatge que s'usa per descriure instruccions a un ordinador.

- Què hi ha en un llenguatge de programació?
  - Dades (nombres, cadenes de caràcters, estructures, ...).
  - Instruccions (aritmètiques, seqüències, repeticions, ...).
- Un llenguatge de programació té una sintaxi i una semàntica molt estricta, ja que l'ha de comprendre l'ordinador.
- Els llenguatges de programació es poden classificar en llenguatges de baix i alt nivell.

Llenguatge màquina

Definició 3: Llenguatge màquina

El *llenguatge màquina* és un sistema d'instruccions i dades codificat en codi **binari** que pot entendre un ordinador (processador).

- Cada arquitectura d'ordinador té el seu propi llenguatge màquina ja que cada arquitectura es diferencia en el nombre de registres, memòria, mida del bus, instruccions, ...
- El llenguatge màquina és un llenguatge de baix nivell i es refereixen a ell com a un llenguatge de primera generació.

Llenguatge assemblador

Definició 4: Llenguatge assemblador

El *llenguatge assemblador* consisteix en un conjunt de mnemònics que representen instruccions bàsiques del processador, que tenen una correspondència pràcticament d'un a un entre les instruccions en assemblador i les instruccions del codi màquina de l'arquitectura.

- El llenguatge d'assemblador és un llenguatge de baix nivell que va ésser creat durant la dècada de 1950, quan es van referir a ell com a llenguatge de segona generació.
- No és el mateix recordar `10110000 01100001` (llenguatge màquina), que `MOV AL, 0x61` (llenguatge d'assemblador).

Llenguatges baix vs alt nivell

- Per programar s'utilitzen habitualment els llenguatges d'alt nivell perquè:
  - Són més llegibles
  - Incrementen la productivitat
  - Són més fàcils de debugar (trobar els errors)
- Però per altra banda es pot perdre eficiència (tant en temps com en memòria).

Índex

Llenguatges compilats vs interpretats

- Atès que els processadors el que entenen és el llenguatge màquina en els llenguatges d'alt nivell cal traduir les intruccions d'alt nivell a llenguatge màquina.
- Depenent de quan es fa aquesta traducció els llenguatges de programació es classifiquen en diversos tipus (entre ells):
  - **compilats**: C++, C, Cobol
  - **semi-compilats**: Java
  - **interpretats**: Python, Lisp, Perl

Compilació

Definició 5: Compilació

La *compilació* és un procés de traducció d'un llenguatge de programació. Normalment aquest procés implica la traducció d'un llenguatge de programació d'alt nivell a un llenguatge de programació de baix nivell.

- La compilació la realitza un programa anomenat compilador.
- No tots els llenguatges de programació necessiten compilar-se, n'hi ha alguns que aquest procés de traducció el fan internament (per ex. Python).

Compilador

Definició 6: Compilador

Un *compilador* és una aplicació (programa) que realitza el procés de **compilació** del codi font d'un programa.

- El compilador genera el fitxer (executable) que permet provar (executar) el programa que hem escrit.
- Només genera l'executable si el nostre codi no té errors de compilació. Es produeix un error de compilació si alguna part del codi està mal escrita i per tant el compilador no l'entèn (error de sintaxi).

Quin compilador usarem?

- En aquest curs aprendrem a programar en C++. El compilador que usarem per traduir el codi C++ a llenguatge màquina és g++.

COMPTE!!

g++ NO està disponible per Windows. Es recomena usar Linux.

Índex

Què és un programa?

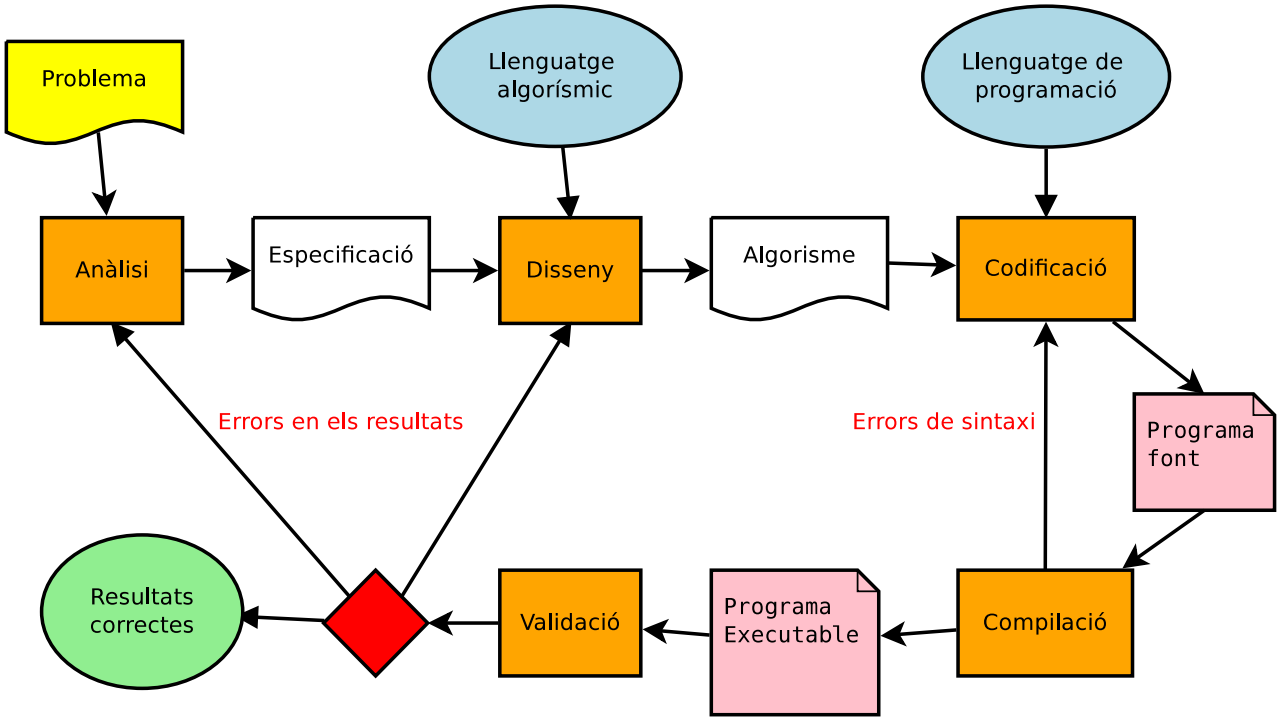
Definició 7: Programa

Un *programa* és un algorisme escrit en un cert llenguatge de programació que executa una tasca concreta.

- Per poder provar un programa cal escriure'l a l'ordinador i compilar-lo usant el compilador per generar l'executable corresponent.
- Cal tenir present que encara que haguem pogut compilar el nostre codi això no implica que no tingui errors.

Índex

Passos en el disseny d'un programa



Etapes

1. Especificació. La tasca que executi el programa ha de ser descrita rigurosament (sense ambigüitats).
2. Disseny de l'algorisme. Cal definir els passos que són necessaris per tal que es faci la tasca descrita en l'especificació.
3. Codificació. L'algorisme ha de ser escrit en un llenguatge de programació que ha de poder ser executat per un ordinador.
4. Execució i proves. El programa ha de ser executat amb un conjunt d'exemples que raonablement cobreixin tots els casos possibles respecte a les dades d'entrada. Si el programa no funciona correctament, l'algorisme ha de ser redissenyat (tornar al pas 2).

Exemple

Enunciat:
Dissenya un programa que donada una quantitat de segons, digui quantes hores, minuts i segons representa.

Entrada:
L'entrada consisteix en un natural.

Sortida:
Escriviu tres naturals h, m, s tals que representin el nombre de segons indicat, on $m < 60$ i $s < 60$.

Índex

Fent una especificació (I)

- Una manera de descriure una especificació és mitjançant la Precondició i la Postcondició.
  - Explicació: què és el que fa l'algorisme.
  - Precondició: especificació de les dades abans que s'executi l'algorisme.
  - Postcondició: especificació de les dades després que s'executi l'algorisme.
- Les especificacions també poden ser escrites en diferents formats. Per exemple, descrivim l'entrada i la sortida del programa, usant llenguatge natural, o amb proposicions formals lògiques.
- Fins i tot, quan estan escrites en llenguatge natural, les especificacions han de ser riguroses i sense ambigüitat.

Fent una especificació (II)

- Especificació de l'enunciat anterior:

  - Explicació: Transforma un nombre de segons en el seu equivalent nombre d'hores, minuts i segons.
  - Precondició: $n \geq 0$
  - Postcondició: $3600 * h + 60 * m + s = n$

Fent una especificació (II)

- Especificació de l'enunciat anterior:
  - Explicació: Transforma un nombre de segons en el seu equivalent nombre d'hores, minuts i segons.
  - Precondició: $n \geq 0$
  - Postcondició: $3600 * h + 60 * m + s = n$

Però aquesta especificació veritablement és correcta?
És incompleta

Especificació incorrecta

- L'especificació anterior no descriu correctament el que el programa ha de calcular.
- Exemple:
  - Si $n = 3815$
  - Una solució és: h=1, m=3, s=35, ja que compleix l'especificació (1*3600 + 3*60 + 35 = 3815)
  - Però les solucions h=0, m=30, s=2015 i h=0, m=0 i s=3815 també compleixen amb l'especificació anterior.
- **On és el problema?**
- **L'especificació ha de ser completa, no ambigua i precisa**. En aquesta especificació no s'indica el rang vàlid de la sortida.

Especificació incorrecta

- L'especificació anterior no descriu correctament el que el programa ha de calcular.
- Exemple:
  - Si $n = 3815$
  - Una solució és: h=1, m=3, s=35, ja que compleix l'especificació (1*3600 + 3*60 + 35 = 3815)
  - Però les solucions h=0, m=30, s=2015 i h=0, m=0 i s=3815 també compleixen amb l'especificació anterior.
- **On és el problema?**
- **L'especificació ha de ser completa, no ambigua i precisa**. En aquesta especificació no s'indica el rang vàlid de la sortida.

Definicions
Passos en el disseny d'un programa
Exercicis de repàs
Especificació
Disseny de l'algorisme
Codificació
Execució i proves

Especificació incorrecta

- L'especificació anterior no descriu correctament el que el programa ha de calcular.
- Exemple:
  - Si $n = 3815$
  - Una solució és: h=1, m=3, s=35, ja que compleix l'especificació (1*3600 + 3*60 + 35 = 3815)
  - Però les solucions h=0, m=30, s=2015 i h=0, m=0 i s=3815 també compleixen amb l'especificació anterior.
- **On és el problema?**
- **L'especificació ha de ser completa, no ambigua i precisa**. En aquesta especificació no s'indica el rang vàlid de la sortida.

Especificació bona

- Especificació correcta l'enunciat anterior:
  - Explicació: Transforma un nombre de segons en el seu equivalent nombre d'hores, minuts i segons.
  - Precondició: $n \geq 0$
  - Postcondició: $3600 * h + 60 * m + s = n$, $0 <= s < 60, 0 <= m < 60$

Índex

Disseny de l'algorisme

- Pot haver més d'un algorisme que compleixi la mateixa especificació i resolgui el mateix problema correctament.
- Quin és l'algorisme que cal usar?
- Sempre que es pugui s'ha d'utilitzar l'algorisme més eficient possible tant en temps com en ús de memòria.
- Però quin és el més eficient? La resposta no és senzilla.

Algorisme 1

```
s = n mod 60        // residu de la divisió
x = n / 60          // divisió entera
m = x mod 60
h = x / 60
```

Algorisme 2

```
h = n / 3600
m = (n mod 3600) / 60
s = n mod 60
```


Índex

El procés de codificació

- El procés de codificació implica traduir l'algorisme al llenguatge de programació triat (en el nostre cas C++), obtenint com a resultat el codi font.
- Aquest codi font per tal de poder-lo provar cal transcriure'l a l'ordinador usant un editor: gedit, kate, geany, emacs, ...
- El nom del fitxer on guardarem el codi font ha de tenir extensió `.cc`. Per exemple: `descomp_temps.cc`

Codificació en C++

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

// Aquest programa llegeix un nombre natural que
// representa una quantitat de temps en segons i escriu
// la seva descomposició en hores, minuts i segons.
int main() {
    int n;
    cin >> n;
    int h = n/3600;
    int m = (n%3600)/60;
    int s = n%60;
    cout << h << ' ' << m << ' ' << s;
}
```

Compilació

- Com ja hem vist, la **compilació** és el procés pel qual es tradueix el codi font d'un programa escrit en un llenguatge d'alt nivell a un altre de nivell inferior (llenguatge màquina).
- La compilació si no hi ha cap error genera el fitxer executable. Per compilar en C++ cal escriure en un terminal:

```
g++ -Wall -o executable.e codifont.cc
```

- Per exemple:

```
g++ -Wall -o descomp_temps.e descomp_temps.cc
```


Índex

Compte amb els valors d'entrada

COMPTE!!

L'algorisme només ha de funcionar per aquells valors d'entrada que compleixen la precondició.

Execució i proves

```
> ./descomp_temps.e
3815
1 3 35

> ./descomp_temps.e
60
0 1 0
```

Índex

Exercicis de repàs

1. Escriu un algorisme que representi el procés de posar una rentadora.
2. A partir de l'altra versió de l'algorisme de conversió de segons, escriu el programa corresponent en C++.
3. Feu un programa que escrigui per pantalla la *Resposta a la Pregunta Definitiva sobre la Vida, l'Univers i Tot Plegat*. Segons el llibre de ciència-ficció *Guia Galàctica per Autoestopistes*[1] aquesta resposta és **42**.

[1] https://ca.wikipedia.org/wiki/La_Resposta_a_la_Pregunta_Definitiva_sobre_la_Vida,_l%27Univers_i_Tot_Plegat

Reconeixement i llicència

Aquesta sessió s'ha basat en el següent material:

- **Material docent de l'assignatura PRO1 de la FIB**. `http://www.cs.upc.edu/~pro1/` fet per en *Jordi Cortadella*, *Ricard Gavaldà* i *Fernando Orejas*.
- **Portal minidosis**. `http://www.minidosis.org/` fet per en *Pau Fernández*.

Tema 1

Estructura bàsica d'un programa i Variables

Fonaments de Programació

Estructura bàsica d'un programa i Variables

Bernardino Casas

bcasas@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

- ✓ Conèixer l'estructura bàsica d'un programa.
- ✓ Ser capaç de declarar variables i constants donant-li noms correctes i adequats.
- ✓ Conèixer i saber usar els diferents literals que hi ha per cada tipus de dades.
- ✓ Conèixer i saber com funciona la instrucció d'assignació.
- ✓ Ser capaç d'indicar el valor d'una variable després d'un seguit d'assignacions.

Índex

Índex

Índex

Què és una instrucció?

Definició 1: Instrucció

Una *instrucció* és un element del programa que controla com i en quin ordre es manipula la informació que gestiona el programa.

- Cada instrucció indica una tasca a realitzar per l'ordinador.
- Un **programa està format per diverses instruccions** que quan s'executen fa que l'ordinador faci quelcom.
- Quan més complex és un programa més instruccions té.

- Una instrucció pot constar d'una sola línia de codi que finalitza en un punt i coma o d'una sèrie d'instruccions d'una sola línia en un bloc.
- Un bloc d'instruccions s'escriu entre claus {} i pot contenir blocs niats (altres blocs).
- En els següents temes anirem aprenent noves instruccions, i també aprendem a usar blocs i blocs niats.

Índex

Main

- Un programa en C++ està format com a mínim per un bloc main. **Main** en anglès vol dir "principal".

```
int main() {


}
```

- Les instruccions que formen el programa van entre les dues claus **{ }**.
- Les instruccions s'executen per ordre començant per la que està més aprop de la clau d'obertura **{**.

El primer programa

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5        cout << "Hola món!" << endl;
6  }
```

- Les línies 1 i 2 serveixen per utilitzar les instruccions de lectura i escriptura (ho veurem en el següent tema). Aquestes dues línies s'han de posar sempre!
- Les línies 4 i 6 s'han de posar sempre ja que indiquen per on ha començar l'ordinador l'execució del programa.
- Aquest programa és molt senzill i el que fa és escriure el missatge **"Hola món!"** per pantalla.

Índex

Què és una variable?

Definició 2: Variable

Una *variable* és un espai on es pot emmagatzemar un valor, que com el seu nom indica pot canviar el contingut durant l'execució del programa.

- Una variable és una porció de memòria, que en funció del tipus de valor que emmagatzemi ocupa més o menys espai.
- Es pot veure com una capsa o recipient que conté quelcom.

Característiques de les variables

- Les variables tenen sempre:
  - *tipus*: indica quins valors pot emmagatzemar la variable. Hi ha variables que contenen enters, reals, caràcters, ...
  - *nom*: com s'identifica a la variable. El nom ha de ser únic dins del seu àmbit.
  - *valor*: tota variable sempre té un valor associat.
- Les variables SEMPRE s'han de crear (declarar) abans d'usar-les.

Els literals

Definició 3: Literal

Un *literal* és una notació per representar un valor fix dins del codi font.

- S'acostuma a veure com dades que estan presents directament en el codi.
- Per exemple:

  1, −3.14, 5e−10, `false`, `"hola"`, ...

Índex

Regles per posar un nom

1. No es poden usar paraules reservades del llenguatge com a noms de variables. Per exemple:

   `int`, `double`, `char`, `bool`, `if`, `else`, `while`, `long` ...
2. El primer caràcter del nom d'una variable ha de ser una lletra (majúscula o minúscula) o el caràcter _ (la "barra baixa", "guió baix" o "subratllat").
3. La resta dels caràcters d'una variable han de ser lletres (majúscules o minúscules) de l'alfabet anglès, dígits o el caràcter _. Per tant no són caràcters vàlids les vocals accentuades o amb dièresi, la lletra ñ o la lletra ç.

Quins noms són correctes?(I)

- Indica quins noms són correctes i quins no.

```
?guay
X2
π
edat
QUANT
_suma4
```

```
2x
unitatdemesura
mEnOr
màxim
area_triang
xy 2
```


Quins noms són correctes?(II)

- Indica quines variables són correctes i quines no.

```
?guay   ✗
X2      ✓
π       ✗
edat    ✓
QUANT   ✓
_suma4  ✓
```

```
2x ✗
unitatdemesura ✓
mEnOr ✓
màxim ✗
area_triang ✓
xy 2 ✗
```

Índex

Declaració d'una variable

- Una variable SEMPRE s'ha de crear (declarar) abans de poder usar-la.
- Per declarar una variable cal escriure en una línia primer el tipus de dades, després el nom que li posarem a la variable i per últim un punt i coma.

```
tipus nom;
```

- Per exemple:

```
int a;
double total;
```

Declaració de vàries variables del mateix tipus

- També es poden declarar vàries variables del mateix tipus en una sola línia.
- S'escriu primer el tipus, a continuació els noms de les variables separats entre comes i per últim el punt i coma.

```
tipus nom1, nom2, nom3;
```

- Per exemple:

```
int a, b, c;
double total, div;
```


Què és una constant?

Definició 3: Constant

Una *constant* és una variable que no es pot modificar.

- Si en un programa s'intenta modificar el valor d'una constant, en compilar es mostraria un error i per tant no es generaria l'executable.
- Per conveni els noms de les constants són en majúscula.

Declaració d'una constant

- Per declarar una constant cal fer:

```
const tipus NOM = expressio;
```

- En no poder canviar el valor d'una constant, és important inicialitzar-la. No fer-ho és un error, perquè no sabrem quin valor té i no tindrà sentit usar-la.
- Per exemple:

```
const int MAX = 32;
const double PRECISIO = 0.0001;
```


Índex

Assignar valors a una variable

- L'assignació és una sentència fonamental en els llenguatges de programació imperatius. S'usa d'aquesta manera:

```
variable = expressio
```

- El seu funcionament és el següent:
  1. Es calcula el valor de l'expressió que hi ha a la dreta del =.
  2. Es descarta el valor que hi hagués emmagatzemat en la variable.
  3. S'emmagatzema el resultat del càlcul anterior en la variable.

Més coses sobre les variables

- Les variables es poden inicialitzar en el moment que es declaren.
- Una seqüència de sentències (no necessàriament assignacions) s'executen seqüencialment, una rera l'altra començant per la primera d'elles.

Recomanació sobre variables

Declareu les variables quan les necessiteu (no pas al principi del programa principal), i inicialitzeu les variables en la mateixa declaració sempre que sigui possible.

Exemple d'assignació (I)

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5    int a;
6    a = 10;   // quant val a?
7
8    int b = 4;
9    b = b + 1;   // quant val b?
10
11   int c = a * b;   // quant val c?
12
13   int d = 0;
14   d = a + b + c + d;   // quant val d?
15 }
```


Exemple d'assignació (II)

```
1→ int a;
2  a = 10;
3
4  int b = 4;
5  b = b + 1;
6
7  int c = a * b;
8
9  int d = 0;
10 d = a + b + c + d;
```

n^o	a	b	c	d
1	??			

Exemple d'assignació (II)

```
1  int a;
2  a = 10;
3
4  int b = 4;
5  b = b + 1;
6
7  int c = a * b;
8
9  int d = 0;
10 d = a + b + c + d;
```

n^o	a	b	c	d
1	??			
2	10			

Estructura bàsica d'un programa
Variables i constants
Exercicis de repàs
Regles per posar els noms
Declaració de variables i constants
Assignació

Exemple d'assignació (II)

```
1  int a;
2  a = 10;
3
4  int b = 4;
5  b = b + 1;
6
7  int c = a * b;
8
9  int d = 0;
10 d = a + b + c + d;
```

n^o	a	b	c	d
1	??			
2	10			
4	10	4		

Exemple d'assignació (II)

```
1  int a;
2  a = 10;
3
4  int b = 4;
5  b = b + 1;
6
7  int c = a * b;
8
9  int d = 0;
10 d = a + b + c + d;
```

n^o	a	b	c	d
1	??			
2	10			
4	10	4		
5	10	5		

Estructura bàsica d'un programa
Variables i constants
Exercicis de repàs
Regles per posar els noms
Declaració de variables i constants
Assignació

Exemple d'assignació (II)

```
1  int a;
2  a = 10;
3
4  int b = 4;
5  b = b + 1;
6
7  int c = a * b;
8
9  int d = 0;
10 d = a + b + c + d;
```

n^o	a	b	c	d
1	??			
2	10			
4	10	4		
5	10	5		
7	10	5	50	

Exemple d'assignació (II)

```
1  int a;
2  a = 10;
3
4  int b = 4;
5  b = b + 1;
6
7  int c = a * b;
8
9  int d = 0;
10 d = a + b + c + d;
```

n^o	a	b	c	d
1	??			
2	10			
4	10	4		
5	10	5		
7	10	5	50	
9	10	5	50	0

Estructura bàsica d'un programa
Variables i constants
Exercicis de repàs
Regles per posar els noms
Declaració de variables i constants
Assignació

Exemple d'assignació (II)

```
1  int a;
2  a = 10;
3
4  int b = 4;
5  b = b + 1;
6
7  int c = a * b;
8
9  int d = 0;
10 d = a + b + c + d;
```

n^o	a	b	c	d
1	??			
2	10			
4	10	4		
5	10	5		
7	10	5	50	
9	10	5	50	0
10	10	5	50	65

Índex

Exercicis de repàs

1. Feu un programa on es declarin tres variables per guardar el DNI, l'edat i el sou que guanya una persona.
2. Escriviu la taula de valors de les variables quan s'executa aquest programa. Feu el desenvolupament pas a pas.

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

int main() {
  int a, c;
  a = 2;
  int b = a*2*10;
  c = a*b;
  int d = c+a+a+a;
}
```

Reconeixement i llicència

Aquesta sessió s'ha basat en el següent material:

- **Material docent de l'assignatura PRO1 de la FIB**. `http://www.cs.upc.edu/~pro1/` fet per en *Jordi Cortadella*, *Ricard Gavaldà* i *Fernando Orejas*.
- **Portal minidosis**. `http://www.minidosis.org/` fet per en *Pau Fernández*.

Tema 2

Entrada i sortida

Fonaments de Programació

Entrada i sortida

Bernardino Casas

bcasas@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

- ✓ Ser capaç d'usar el canal d'entrada (cin) per obtenir informació del teclat.
- ✓ Ser capaç d'usar el canal de sortida per mostrar informació per la pantalla.
- ✓ Ser capaç de formatejar la sortida d'un programa per tal que hi hagi salts de línia.

Índex

Índex

Índex

Entrada estàndar

- El canal d'*entrada estàndar* és la font d'informació per defecte dels programes. És la forma típica que tenen els programas d'obtenir dades de l'exterior.
- Típicament, un programa obté unes dades, fa uns càlculs i produeix uns resultats.
- L'entrada estàndar, normalment, és el teclat.
- La instrucció de lectura que permet obtenir dades del teclat és la següent:

```
cin >> variable;
```

On `variable` és el nom de la variable que volem utilitzar per guardar el valor llegit.

Com funciona la instrucció de lectura

- Una instrucció de lectura el que fa és:
  1. Aturar el programa i esperar a que s'escrigui quelcom al teclat.
  2. Emmagatzemar a la memòria tot el que s'escrigui fins que el premi la tecla `Return` (o `Enter`).
  3. Processar el que s'ha escrit per "traduir" els caràcters a valors del tipus adequat.
  4. Si el que s'ha escrit és insuficient per omplir totes les variables de la instrucció, es repeteix el procés. Si hi ha dades suficients, s'omplen les variables i el programa continua.

Biblioteca iostream

COMPTE!!

Per tal d'usar les instruccions de lectura i el canal d'entrada `cin` cal incloure la biblioteca `iostream`:

```
#include <iostream>
```

Índex

Primers exemples

- Lectura d'una variable

```
int n;

cin >> n; // llegim una variable
```

- Es pot llegir més d'una variable en una línia, però sempre ha d'aparèixer l'operador >> entre les variables.

```
int a, b, c;

cin >> a >> b >> c; // OK: llegim tres variables

cin >> a, b; // INCORRECTE!!! Falta l'operador >>
             // entre les variables
```

Exemple amb caràcters

- Si al canal d'entrada s'escriu `1234`. Quins valors tindran les variables en el següent programa?

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

int main() {
  char a, b, c;
  cin >> a >> b >> c;
}
```


Entrada
Sortida
Exercicis de repàs
Ús i funcionament del canal d'entrada
Exemples

Exemple amb caràcters

- Si al canal d'entrada s'escriu `1234`. Quins valors tindran les variables en el següent programa?

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

int main() {
  char a, b, c;
  cin >> a >> b >> c;
}
```

a val '1', b val '2' i c val '3'

Exemple amb enters

- Si al canal d'entrada s'escriu `1234`. Quins valors tindran les variables en el següent programa?

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

int main() {
    int a, b, c;
    cin >> a >> b >> c;
}
```


Exemple amb enters

- Si al canal d'entrada s'escriu `1234`. Quins valors tindran les variables en el següent programa?

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

int main() {
    int a, b, c;
    cin >> a >> b >> c;
}
```

a val 1234, i el programa es queda esperant a que s'introdueixi més informació per omplir les altres variables

Índex

Índex

Canal de sortida estàndar

- El canal de *sortida estàndar* és el lloc per defecte on els programes escriuen la informació quan s'executen.
- La sortida estàndar, normalment, és la pantalla.
- La instrucció d'escriptura que permet escriure dades a la pantalla és la següent:

```
cout << expressio;
```

- Per exemple:

```
cout << "Hola Mon";
```

escriuria a la pantalla la cadena de caràcters **Hola Mon**

Com funciona el cout

- Una instrucció d'escriptura el que fa és:
  1. Avaluar l'expressió.
  2. El resultat de l'avaluació s'escriu a la pantalla en el lloc on està en aquests moments el cursor.

IMPORTANT: Veritablement no apareix la informació per pantalla fins que no s'escriu un `endl` o `flush`, o s'acaba el programa.

Biblioteca iostream

COMPTE!!

Per tal d'usar les instruccions d'escriptura i el canal de sortida `cout` cal incloure la biblioteca `iostream`:

```
#include <iostream>
```

Incloent un cop la biblioteca es poden usar les instruccions de lectura i d'escriptura, i el canal d'entrada i el canal de sortida.

Índex

Varis cout junts

- Diferents instruccions cout es poden ajuntar en una de sola.
- Per exemple:

```
cout << 8 + 8;
cout << "jutges mengen fetge";
cout << '!';
```

És el mateix que:

```
cout << 8 + 8 << "jutges mengen fetge" << '!';
```

Què apareixerà per pantalla?

Entrada
Sortida
Exercicis de repàs
Ús i funcionament del canal de sortida
Exemples

Varis cout junts

- Diferents instruccions cout es poden ajuntar en una de sola.
- Per exemple:

```
cout << 8 + 8;
cout << "jutges mengen fetge";
cout << '!';
```

És el mateix que:

```
cout << 8 + 8 << "jutges mengen fetge" << '!';
```

Què apareixerà per pantalla?

16jutges mengen fetge!

Els espais

- Només apareixen els espais en blanc que s'indiquen a la instrucció.
- Per exemple:

```
cout << 1 << 2 << 3;
```

mostra per pantalla: 123

- Si es vol que aparegui un espai entre cada número cal indicar-ho. Per exemple:

```
cout << 1 << ' ' << 2 << ' ' << 3;
```

mostra per pantalla: 1 2 3

El salt de línia

- Tot el que escrivim apareix en una mateixa línia. Si es vol fer un canvi de línia cal usar `endl`.
- Per exemple:

```
cout << 8 + 8 << endl;
cout << "jutges mengen" << endl << '!';
cout << endl << '!';
```

mostra per pantalla:

16
jutges mengen
!
!

Índex

Exercicis de repàs

1. Feu un programa que escrigui un quadrat de 5 x 5 caràcters amb la lletra 'x'.
2. Feu un programa que escrigui per pantalla el següent:

```
%%%          %%%     ########      @@@@@@@
 %%%        %%%      ###           @@      @
  %%%%%%%%           ###           @@
   %%%%%             ######          @@@@@@@
   %%%                ###                 @@
   %%%                ###           @     @@
   %%%                ########      @@@@@@@
```

Reconeixement i llicència

Aquesta sessió s'ha basat en el següent material:

- **Material docent de l'assignatura PRO1 de la FIB**. `http://www.cs.upc.edu/~pro1/` fet per en *Jordi Cortadella*, *Ricard Gavaldà* i *Fernando Orejas*.
- **Portal minidosis**. `http://www.minidosis.org/` fet per en *Pau Fernández*.

Tema 3

Tipus de dades

Fonaments de Programació

Tipus de dades

Bernardino Casas

bcasas@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

- ✓ Conèixer i saber usar els tipus de dades enters, reals, booleans, caràcters i strings.
- ✓ Conèixer les diferents operacions que es poden fer amb els tipus de dades.

Índex

Definicions
Tipus de dades
Exercicis de repàs

Índex

Què és un tipus de dades?

Definició 4: Tipus de dades

El tipus de dades d'una variable indica:

- el conjunt de valors que pot emmagatzemar una variable
- el conjunt d'operacions que es poden realitzar amb aquesta variable.

- El tipus de dades en C++ són:
  - bàsics: `int`, `double`, `bool`, `char`, , ...
  - complexos: string, vector

Índex

Índex

int

- El tipus de dades `int` representa el conjunt de nombres enters.
- A la pràctica els ordinadors tenen una limitació en la representació dels nombres enters.
- Per una arquitectura de 32 bits es poden representar els nombres que estan a l'interval $[-2^{31}, 2^{31} - 1]$ $[-2147483648, 2147483647]$.
- Operadors: +, -, *, /, %.
- Exemples: 11, 1714, -189212

double

- El tipus de dades `double` representa el conjunt de nombres reals.
- A la pràctica els ordinadors tenen una limitació en la representació dels nombres reals respecte un cert interval i una certa precisió.
- Nombres més propers a cero: $[\pm 5x10^{-324}]$
- Nombres més llunyans al cero: $[\pm 1.7976931348623157x10^{308}]$
- Representacions especials pel 0, $+\infty$ i $-\infty$
- Operadors: +, -, *, /.
- Exemples: 3.14159, 5e-17, -5.86733

Índex

bool

- El tipus de dades `bool` representa els valors lògics.
- Només té dos valors possibles: `true` (verdader), `false` (fals).
- Operadors: `not`, `and`, `or`.
- Exemples: `true`, `false`

Propietats dels booleans

Commutativa

a `and` b = b `and` a

a `or` b = b `or` a

Associativa

(a `and` b) `and` c = a `and` (b `and` c)

(a `or` b) `or` c = a `or` (b `or` c)

Distributiva

a `and` (b `or` c) = (a `and` b) `or` (a `and` c)

a `or` (b `and` c) = (a `or` b) `and` (a `or` c)

Doble negació

`not` (`not` a) = a

Llei de Morgan

`not` (a `and` b) = (`not` a) `or` (`not` b)

`not` (a `or` b) = (`not` a) `and` (`not` b)

Índex

char

- El tipus de dades `char` emmagatzema caràcters. Un caràcter és una lletra, dígit, signe de puntuació, espai, etc.
- Cada caràcter es representa amb un codi (un número). Hi ha diversos codis estàndars:
  - ASCII: American Standard Code for Information Interchange.
  - Unicode: Més extens que l'ASCII.
- Els literals caràcters s'escriuen entre cometes simples:
  `'A','1','?',')','\t'`
- Operadors: atès que els caràcters es codifiquen com un nombre, els operadors aritmètics es poden usar, encara que només té sentit la suma i la resta.
  `'C'+1='D','m'+4='q'`

Taula ASCII

Hex	Dec	Char		Hex	Dec	Char		Hex	Dec	Char		Hex	Dec	Char	
0x00	0	NULL	null	0x20	32	Space		0x40	64	@		0x60	96	`	
0x01	1	SOH	Start of heading	0x21	33	!		0x41	65	A		0x61	97	a	
0x02	2	STX	Start of text	0x22	34	"		0x42	66	B		0x62	98	b	
0x03	3	ETX	End of text	0x23	35	#		0x43	67	C		0x63	99	c	
0x04	4	EOT	End of transmission	0x24	36	$		0x44	68	D		0x64	100	d	
0x05	5	ENQ	Enquiry	0x25	37	%		0x45	69	E		0x65	101	e	
0x06	6	ACK	Acknowledge	0x26	38	&		0x46	70	F		0x66	102	f	
0x07	7	BELL	Bell	0x27	39	'		0x47	71	G		0x67	103	g	
0x08	8	BS	Backspace	0x28	40	(		0x48	72	H		0x68	104	h	
0x09	9	TAB	Horizontal tab	0x29	41	)		0x49	73	I		0x69	105	i	
0x0A	10	LF	New line	0x2A	42	*		0x4A	74	J		0x6A	106	j	
0x0B	11	VT	Vertical tab	0x2B	43	+		0x4B	75	K		0x6B	107	k	
0x0C	12	FF	Form Feed	0x2C	44	,		0x4C	76	L		0x6C	108	l	
0x0D	13	CR	Carriage return	0x2D	45	-		0x4D	77	M		0x6D	109	m	
0x0E	14	SO	Shift out	0x2E	46	.		0x4E	78	N		0x6E	110	n	
0x0F	15	SI	Shift in	0x2F	47	/		0x4F	79	O		0x6F	111	o	
0x10	16	DLE	Data link escape	0x30	48	0		0x50	80	P		0x70	112	p	
0x11	17	DC1	Device control 1	0x31	49	1		0x51	81	Q		0x71	113	q	
0x12	18	DC2	Device control 2	0x32	50	2		0x52	82	R		0x72	114	r	
0x13	19	DC3	Device control 3	0x33	51	3		0x53	83	S		0x73	115	s	
0x14	20	DC4	Device control 4	0x34	52	4		0x54	84	T		0x74	116	t	
0x15	21	NAK	Negative ack	0x35	53	5		0x55	85	U		0x75	117	u	
0x16	22	SYN	Synchronous idle	0x36	54	6		0x56	86	V		0x76	118	v	
0x17	23	ETB	End transmission block	0x37	55	7		0x57	87	W		0x77	119	w	
0x18	24	CAN	Cancel	0x38	56	8		0x58	88	X		0x78	120	x	
0x19	25	EM	End of medium	0x39	57	9		0x59	89	Y		0x79	121	y	
0x1A	26	SUB	Substitute	0x3A	58	:		0x5A	90	Z		0x7A	122	z	
0x1B	27	FSC	Escape	0x3B	59	;		0x5B	91	[		0x7B	123	{	
0x1C	28	FS	File separator	0x3C	60	<		0x5C	92	\		0x7C	124	|	
0x1D	29	GS	Group separator	0x3D	61	=		0x5D	93	]		0x7D	125	}	
0x1E	30	RS	Record separator	0x3E	62	>		0x5E	94	^		0x7E	126	~	
0x1F	31	US	Unit separator	0x3F	63	?		0x5F	95	_		0x7F	127	DEL	

string

- El tipus de dades string emmagatzema textos, és a dir, cadenes de caràcters de qualsevol mida (incloent la cadena buida i la d'un un sol caràcter).
- Per usar els strings cal afegir `#include` <string> al principi del programa.
- Els literals strings s'escriuen entre dues cometes dobles.
- Operadors: + (concatenació)
- Exemples: `"Hola"`, `"3.14159"`, `";-)"`

Resum de tipus de dades

Tipus	Descripció	Bytes	Exemple
`int`	enters	4	123, -50
`double`	reals	8	3.14, -5e-7
`char`	caràcters	1	`'a'`, `'4'`
`bool`	booleans	1	`true`, `false`
string	cadenes de caràcters	tants bytes com char + 1	`"Hola!"`, `"1234"`

Índex

Exercicis de repàs

1. Feu un programa que llegeixi tres nombres reals i escrigui la seva suma.
2. Feu un programa que llegeixi la base i l'alçada d'un triangle, en calculi la seva àrea i la mostri per pantalla.
3. Feu un programa que calculi quants anys li falten per jubilar-se a una persona. El programa et demana l'edat actual i a continuació et mostra els anys que li queden per jubilar-se. Feu la suposició que la jubilació és als 65 anys (aquesta informació l'heu de guardar en una constant).

Reconeixement i llicència

Aquesta sessió s'ha basat en el següent material:

- **Material docent de l'assignatura PRO1 de la FIB**. `http://www.cs.upc.edu/~pro1/` fet per en *Jordi Cortadella*, *Ricard Gavaldà* i *Fernando Orejas*.
- **Portal minidosis**. `http://www.minidosis.org/` fet per en *Pau Fernández*.

Tema 4

Operadors i expressions

Fonaments de Programació

Operadors i expressions

Bernardino Casas

bcasas@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

- ✓ Conèixer els diferents operadors que hi ha per cada tipus de dades.
- ✓ Saber calcular el resultat de cada operació de C++ del tipus de dades enter, real i booleà.
- ✓ Ser capaç de crear expressions enteres, reals i booleanes.
- ✓ Ser capaç d'avaluar qualsevol expressió per obtenir el valor final.

Índex

Índex

Índex

Operadors sobre enters i reals

Operació	int	double
Suma	+	+
Resta	-	-
Multiplicació	*	*
Divisió	/	/
Mòdul[1]	%	no existeix
Canvi de signe	-	-

[1] resta de la divisió entera

Divisió entera

- Quan es divideix entre enters (els dos operands de la divisió són enters) el resultat és un enter, no un real com un podria esperar.
- Si algun dels dos operands és un real el resultat és un real.
- Per exemple:

```
int a = 3 / 4;   // a val 0
int b = 15 / 4;  // b val 3
int c = 16 / 2.5;   // l'operació dóna com a resultat 6.4, però
                    // com s'ha de guardar dins un enter, c val 6
int d = -18 / 4;  // d val -4
```


Mòdul (I)

- El *mòdul* és el residu de la divisió i es representa amb el símbol **%**.
- El mòdul només s'aplica entre operands enters. Si s'intenta operar amb reals genera un error en compilar.
- El resultat del mòdul segons la definició matemàtica sempre és positiu ($0 \leq residu < |divisor|$), però la majoria dels llenguatges de programació no segueixen aquesta definició.

Signe del residu

En C++, el mòdul permet resultats negatius. **El signe del residu serà el mateix que el signe del dividend**.

Mòdul (II)

- Per exemple:

```
int a = 15 % 2 // a val 1
int b = 111 % 3 // b val 0
int c = 14 % 19 // c val 14

int d = -18 % 4 // d val -2
int e = 18 % -4 // e val 2
```

- Per més informació:
  `en.wikipedia.org/wiki/Modulo_operation`

Índex

Operador booleà `not`

- L'operador booleà `not` és un operador unari (només té un operand) i correspon a la negació lògica. Per representar aquest operador abans s'usava també el símbol **!**.
- El comportament d'aquest operador és el següent:

a	`not` a
`true`	`false`
`false`	`true`

- Per exemple:

```
bool b1 = true;
bool b2 = not b1; //b2 val false
```


Operador booleà `and`

- L'operador booleà `and` és un operador binari i correspon a la conjunció lògica. Per representar aquest operador abans s'usaven també els símbols **&&**.
- El comportament d'aquest operador és el següent:

a	b	a `and` b
`false`	`false`	`false`
`false`	`true`	`false`
`true`	`false`	`false`
`true`	`true`	`true`

- Per exemple:

```
bool b1 = true;
bool b2 = false;
bool b3 = b1 and b2; //b3 val false
```

Operador booleà `or`

- L'operador booleà `or` és un operador binari i correspon a la disjunció lògica. Per representar aquest operador abans s'usaven també els símbols ||.
- El comportament d'aquest operador és el següent:

a	b	a `or` b
`false`	`false`	`false`
`false`	`true`	`true`
`true`	`false`	`true`
`true`	`true`	`true`

- Per exemple:

```
bool b1 = true;
bool b2 = false;
bool b3 = b1 or b2; //b3 val true
```


Índex

Operadors de comparació

- Els diferents operadors binaris que permeten comparar dos valors de tipus `int`, `double`, `char` o string són:

<	menor
>	major
<=	menor o igual
>=	major o igual
==	igualtat
!=	diferència

- El resultat de cadascun d'aquests operadors és un booleà.

= i ==

IMPORTANT

NO confondre l'assignació (=) amb la igualtat (==).

Recordeu: = s'usa per guardar un valor dins una variable, mentre que == s'usa per comparar el valors.

Índex

Operador ++ i −−

- Els operadors d'increment (++) i de decrement (−−) permeten incrementar i decrementar en una unitat una variable.
- Aquests operadors són unaris i s'usen d'aquesta forma:

```
++variable
--variable;
```

- Per exemple:

```
int x = 4;
++x;   // és equivalent a x = x+1; x val 5
--x;   // és equivalent a x = x-1; x val 4
```

Operadors amb assignació

- Hi ha operadors que tenen inclosa l'assignació en el seu comportament. Alguns són: $+=$, $-=$, $*=$, $/=$, $\%=$
- Aquests operadors són equivalents a l'operació corresponent i una assignació.

variable **op=** expressio

≡

variable = variable **op** expressio

- Per exemple:

```
int x = 8;
x *= 5;   // és equivalent a x = x * 5; x val 40
```


Operador de conversió de tipus

- Les conversions de tipus permeten convertir un valor d'un tipus a un altre (per exemple, d'enter a real).
- Per convertir quelcom a un altre tipus, es posa entre parèntesis l'expressió i davant el tipus al que volem convertir.
- Per exemple:

```
int a = 65;
int b = 10;
double x = a / double(b);  // x val 6.5
char c;
c = char(a);  // c val 'A'
```

Índex

Operadors
Expressions
Exercicis de repàs
Definició d'expressió
Avaluació d'expressions

Índex

1 Operadors
- Operadors enters i reals
- Operadors booleans
- Operadors relacionals
- Altres operadors

2 Expressions
- Definició d'expressió
- Avaluació d'expressions

3 Exercicis de repàs

Què és una expressió?

Definició 4: Expressió

Una *expressió* és una combinació de literals, variables, operadors i funcions que es pot avaluar i generar un valor.

- Depenent del tipus que una expressió genera parlem d'expressions enteres, reals, booleanes, etc.
- Per exemple:

```
4 + 3*(8 - 1)          // expressió entera
sqrt(x)*log(4*n)       // expressió real
(i - 3) <= x           // expressió booleana
(a != b) and (s <= "abc")   // expressió booleana
```


Índex

Avaluació d'expressions

1. Primerament es substitueix cada variable de l'expressió pel valor que tinguin en aquest moment.
2. S'avalua primerament tot el que estigui entre parèntesis i les crides a funcions.
3. S'avaluen els operadors segons el seu ordre de precedència.
4. Quan dos o més operadors tenen la mateixa prioritat, s'avaluen aquests operadors depenent de la seva associativat (*grouping*). Primer els que estan més a l'esquerra (esquerra-dreta) o al revés (dreta-esquerra).

Nota: En general l'associativat dels operadors és d'esquerra a dreta.

Precedència d'operadors

Level	Precedence group	Operator	Description	Grouping
1	Scope	`::`	scope qualifier	Left-to-right
2	Postfix (unary)	`++ --`	postfix increment / decrement	Left-to-right
		`()`	functional forms	
		`[]`	subscript	
		`. ->`	member access	
3	Prefix (unary)	`++ --`	prefix increment / decrement	Right-to-left
		`~ !`	bitwise NOT / logical NOT	
		`+ -`	unary prefix	
		`& *`	reference / dereference	
		`new delete`	allocation / deallocation	
		`sizeof`	parameter pack	
		`(type)`	C-style type-casting	
4	Pointer-to-member	`.* ->*`	access pointer	Left-to-right
5	Arithmetic: scaling	`* / %`	multiply, divide, modulo	Left-to-right
6	Arithmetic: addition	`+ -`	addition, subtraction	Left-to-right
7	Bitwise shift	`<< >>`	shift left, shift right	Left-to-right
8	Relational	`< > <= >=`	comparison operators	Left-to-right
9	Equality	`== !=`	equality / inequality	Left-to-right
10	And	`&`	bitwise AND	Left-to-right
11	Exclusive or	`^`	bitwise XOR	Left-to-right
12	Inclusive or	`|`	bitwise OR	Left-to-right
13	Conjunction	`&&`	logical AND	Left-to-right
14	Disjunction	`||`	logical OR	Left-to-right
15	Assignment-level expressions	`= *= /= %= += -= >>= <<= &= ^= |=`	assignment / compound assignment	Right-to-left
		`?:`	conditional operator	
16	Sequencing	`,`	comma separator	Left-to-right

Precedència d'operadors

Prioritat + / —

Grup	Operador	Descripció	Assoc.
Unari (postfix)	++ −−	post-increment/decrement	E-D
	()	parentèsis	
	. −>	operacions d'accés	
Unari (prefix)	++ −−	pre-increment/decrement	D-E
	not !	negació lògica	
	+ −	canvi de signe	
	sizeof	mida	
	(tipus)	conversió de tipus	
Aritmètic: escalat	* / %	multiplicació, divisió, mòdul	E-D
Aritmètic: addició	+ −	suma, resta	E-D
Relacional	< > <= >=	operadors de comparació	E-D
Igualtat	== !=	igualtat / diferència	E-D
Conjunció	**and** &&	and lògic	E-D
Disjunció	**or** ||	or lògic	E-D
Assignació	= *= /= %= += −=	operadors d'assignació	D-E

Exemple d'avaluació d'expressions

```
int a = 10; int b = 4;
a%7 + b*2 >= a*b
≡
10%7 + 4*2 >= 10*4
≡
3 + 4*2 >= 10*4
≡
3 + 8 >= 10*4
≡
3 + 8 >= 40
≡
11 >= 40
≡
false
```

Exemple d'avaluació d'expressions

```
int a = 10; int b = 4;
a%7 + b*2 >= a*b
≡
10%7 + 4*2 >= 10*4
≡
3 + 4*2 >= 10*4
≡
3 + 8 >= 10*4
≡
3 + 8 >= 40
≡
11 >= 40
≡
false
```


Operadors
Expressions
Exercicis de repàs
Definició d'expressió
Avaluació d'expressions

Exemple d'avaluació d'expressions

```
int a = 10; int b = 4;
a%7 + b*2 >= a*b
≡
10%7 + 4*2 >= 10*4
≡
3 + 4*2 >= 10*4
≡
3 + 8 >= 10*4
≡
3 + 8 >= 40
≡
11 >= 40
≡
false
```

Exemple d'avaluació d'expressions

```
int a = 10; int b = 4;
a%7 + b*2 >= a*b
≡
10%7 + 4*2 >= 10*4
≡
3 + 4*2 >= 10*4
≡
3 + 8 >= 10*4
≡
3 + 8 >= 40
≡
11 >= 40
≡
false
```


Operadors
Expressions
Exercicis de repàs
Definició d'expressió
Avaluació d'expressions

Exemple d'avaluació d'expressions

```
int a = 10; int b = 4;
a%7 + b*2 >= a*b
≡
10%7 + 4*2 >= 10*4
≡
3 + 4*2 >= 10*4
≡
3 + 8 >= 10*4
≡
3 + 8 >= 40
≡
11 >= 40
≡
false
```

Exemple d'avaluació d'expressions

```
int a = 10; int b = 4;
a%7 + b*2 >= a*b
≡
10%7 + 4*2 >= 10*4
≡
3 + 4*2 >= 10*4
≡
3 + 8 >= 10*4
≡
3 + 8 >= 40
≡
11 >= 40
≡
false
```


Operadors
Expressions
Exercicis de repàs
Definició d'expressió
Avaluació d'expressions

Exemple d'avaluació d'expressions

```
int a = 10; int b = 4;
a%7 + b*2 >= a*b
≡
10%7 + 4*2 >= 10*4
≡
3 + 4*2 >= 10*4
≡
3 + 8 >= 10*4
≡
3 + 8 >= 40
≡
11 >= 40
≡
false
```

Índex

Exercicis de repàs

1. Avalua les següents expressions suposant que les variables *a*, *b* i *c* són de tipus booleà amb els valors `true`, `false` i `false`, respectivament.
   a) a **`or`** b **`and not`** c
   b) (a **`or`** b) **`and not`** (a **`or`** c)
2. Avalua les següents expressions suposant que les variables *x*, *y* i *z* són de tipus enter amb els valors 2, 7, 11, respectivament.
   a) $-z*x + x*y - z\%y*z$
   b) $y\%(z + x) <= z + x/y$

Reconeixement i llicència

Aquesta sessió s'ha basat en el següent material:

- **C++ Language - C++ Tutorials**. `http://www.cplusplus.com/doc/tutorial/`.
- **Portal minidosis**. `http://www.minidosis.org/` fet per en *Pau Fernández*.
- **Material docent de l'assignatura PRO1 de la FIB**. `http://www.cs.upc.edu/~pro1/` fet per en *Jordi Cortadella*, *Ricard Gavaldà* i *Fernando Orejas*.

Tema 5

Estructura d'un programa i ús de biblioteques

Fonaments de Programació

Estructura general d'un programa i ús de biblioteques

Bernardino Casas

bcasas@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

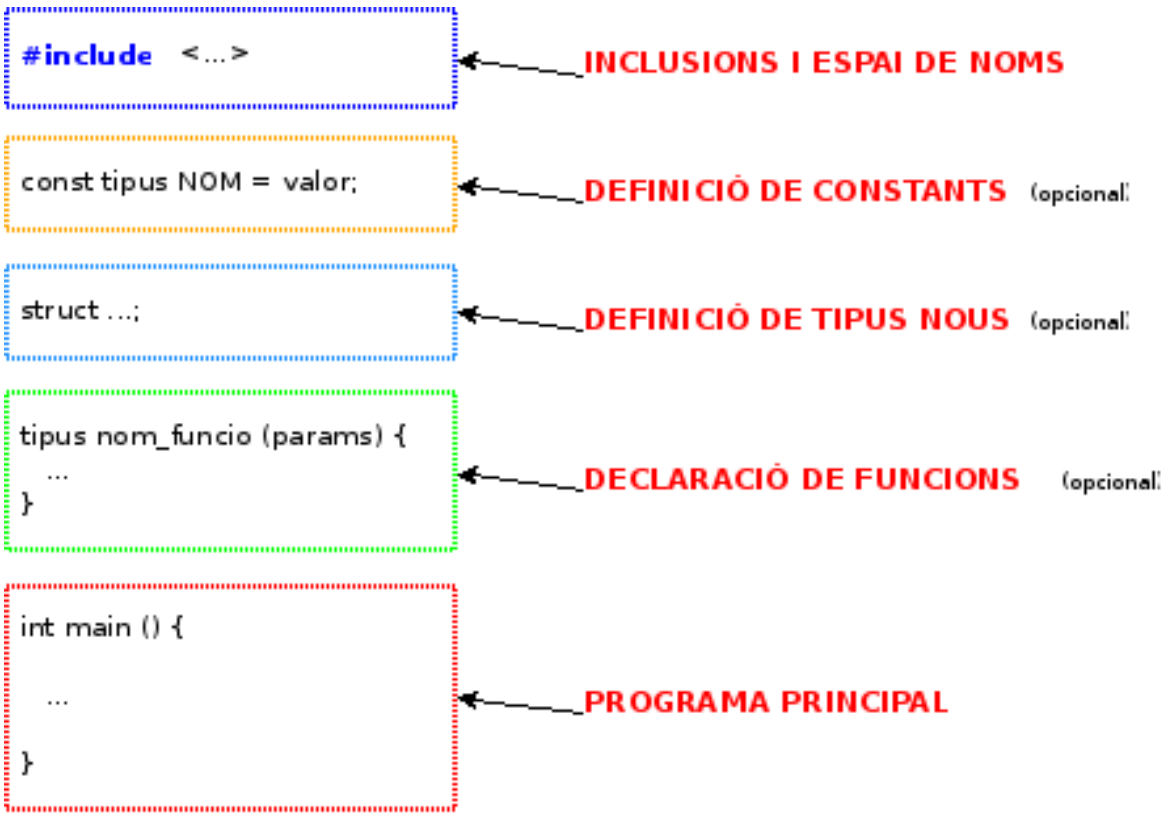
- ✓ Conèixer l'estructura general d'un programa en C++.
- ✓ Ser capaç d'usar biblioteques en C++.
- ✓ Ser capaç d'usar classes en C++.

Índex

Índex

Índex

Estructura d'un programa



Noves funcions

- La definició de noves funcions permet al programador estructurar millor el codi. Un programador pot crear les seves pròpies funcions i després utilitzar-les en el **main** o en d'altres funcions.
- Un programa no ha d'incorporar necessàriament funcions addicionals (sobretot els programes molt senzills).
- En propers temes del curs veurem com es defineixen i es creen noves funcions.

Índex

Com s'executa un programa

- Un programa sempre comença amb la primera instrucció del **main**.
- A continuació s'executen seqüencialment instrucció rera instrucció fins arribar a l'última instrucció del main.
- La *composició condicional* o la *composició repetitiva* (que veurem en propers temes) alteren l'ordre d'execució.
- Les crides a *funcions* també alteren l'ordre d'execució seqüencial.

Índex

Inclusions

- El llenguatge C++ permet incloure biblioteques o classes per usar-les en el nostre programa.
- Per incloure una biblioteca o una classe cal fer:

```
#include < nom de la biblioteca >
```

- IMPORTANT: La instrucció **include** no acaba mai amb **;**.
- A continuació de les inclusions, cal afegir sempre la línia: **using namespace** std;
- Per exemple:

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


Índex

Què és una biblioteca?

Definició 1: Biblioteca

Una *biblioteca* o també dita *llibreria* (anglès: library) és un repositori amb funcions ja implementades. Un programador pot incloure una biblioteca per usar aquestes funcions en un programa.

- Les biblioteques que podeu usar són:
  - cmath: funcions matemàtiques (`sin`, `cos`, `sqrt`, `pow`, `log`, ...).
  - algorithm: funcions relacionades amb les seqüències d'elements.
  - iostream: permet interactuar amb el canal d'entrada i el canal de sortida.

Usant la biblioteca iostream (I)

- Si volem escriure per pantalla o llegir de teclat usarem la biblioteca `iostream`.

```
#include <iostream>

int main() {
  std::cout << "Hello world!" << std::endl;
}
```

Usant la biblioteca iostream (II)

- Per no haver d'escriure **std::** davant de les instruccions de lectura i escriptura (entre d'altres) farem servir:

```
#include <iostream>

using namespace std; // això ens permet evitar d'escriure std:: davant
                     // de cout i endl

int main() {
  cout << "Hello world!" << endl;
}
```


Índex

Què és una classe?

Definició 2: Classe

Una *classe* és una representació d'un tipus d'objecte, és a dir, una estructura de dades concreta i una col·lecció de subrutines (funcions).

- Així com un plànol es pot utilitzar per construir diversos edificis, una classe es pot utilitzar per crear diverses còpies d'un objecte.
- Les classes extres que podem usar en els nostres programes són:

  `string` permet usar les cadenes de caràcters.
  `vector` permet usar la classe vector.

Índex

Exercicis de repàs

1. Feu un programa que donat el radi d'una esfera, calculi i mostri per pantalla el volum d'aquesta esfera.
2. Resolt el mateix problema de l'exercici anterior però usant la constant PI definida a la biblioteca `cmath`.
3. Feu un programa que donat una quantitat de graus, calculi i mostri per pantalla el sinus, el cosinus i la tangent d'aquests graus. Per calcular el sinus, el cosinus i la tangent useu la biblioteca `cmath`.

Reconeixement i llicència

Aquesta sessió s'ha basat en el següent material:

- **C++ Language - C++ Tutorials**. `http://www.cplusplus.com/doc/tutorial/`.
- **Portal minidosis**. `http://www.minidosis.org/` fet per en *Pau Fernández*.

Tema 6

Composició condicional

Fonaments de Programació

Composició condicional

Bernardino Casas bcasas@cs.upc.edu
Rosa M. Jiménez jimenez@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

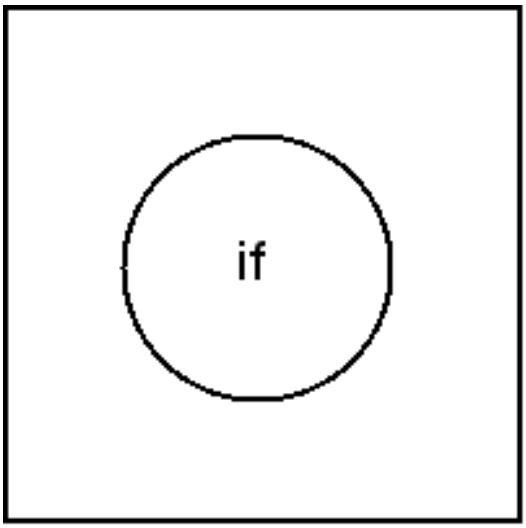
- ✓ Ser capaç de seguir manualment un programa que inclogui una composició condicional.
- ✓ Saber usar la composició condicional per poder seleccionar una opció entre dues possibles opcions.
- ✓ Saber usar la composició condicional per poder seleccionar una opció entre més de dues opcions.

Índex

Índex

Índex

if



Sintaxis i semàntica

- Sintaxis:

```
if ( condicio ) {
  S1
}
```

- Semàntica:
  - La `condicio` és una expressió booleana.
  - Si la `condicio` avalua a cert llavors s'executen les sentències `S1`, i sinó continua l'execució del programa normalment.

Exemple composició condicional simple

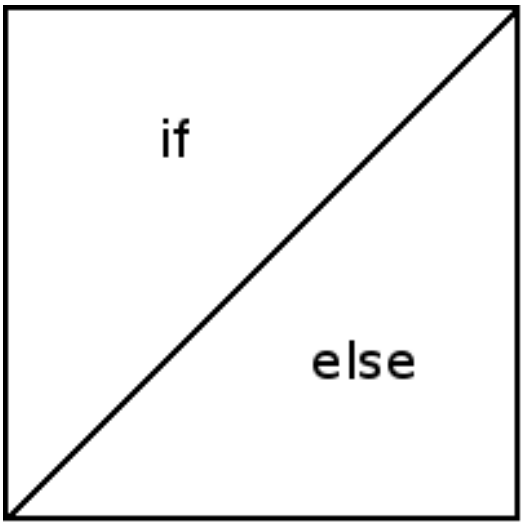
```
#include <iostream>
using namespace std;

int main() {
  // Calcula el valor absolut d'un nombre enter.
  int a;
  cin >> a;

  if (a < 0) {
    a = -a;
  }
  cout << "Valor absolut: " << a << endl;
}
```

Índex

if / else



Condició amb else

- Sintaxis:

```
if ( condicio ) {
    S1
} else {
    S2
}
```

- Semàntica:
  - La `condicio` és una expressió booleana.
  - Si la `condicio` avalua a cert llavors s'executen les sentències `S1`, i sinó s'executen les sentències `S2`.
  - MAI es poden executar les sentències `S1` i `S2` alhora.

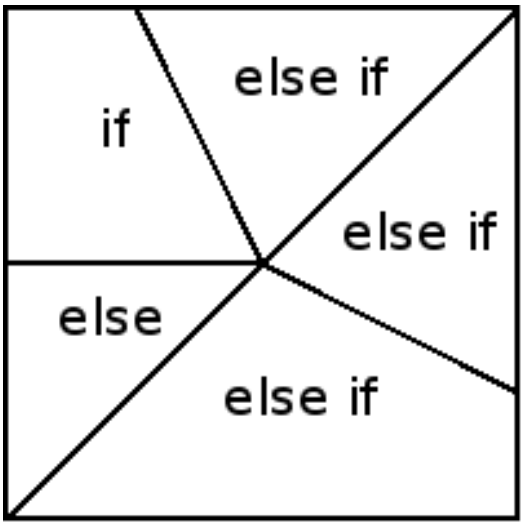
Exemple composició condicional amb else

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

int main() {
  // Calcula el mínim de dos nombres enters.
  int a, b;
  cin >> a >> b;
  int m;
  if (a <= b) {
    m = a;
  } else {
    m = b;
  }
  cout << "Minim: " << m << endl;
}
```

Índex

if / else if / . . . / else



Concatenació de composicions condicionals

- Diverses estructures **if + else** es poden concatenar una rera l'altre, per d'aquesta manera poder comprovar un rang de valors.
- La semàntica d'aquesta concatenació és la següent:
  1. Es comença l'execució per la primera condició de les composicions condicionals concatenades.
  2. En cas que aquesta condició avaluï a **true** llavors s'executaran les sentències associades i anirem al final de les composicions condicionals.
  3. En cas contrari, es procedirà amb la següent condició fins a trobar una que avaluï a **true** o no hi hagi cap altra condició.

Exemple concatenació de composicions condicionals

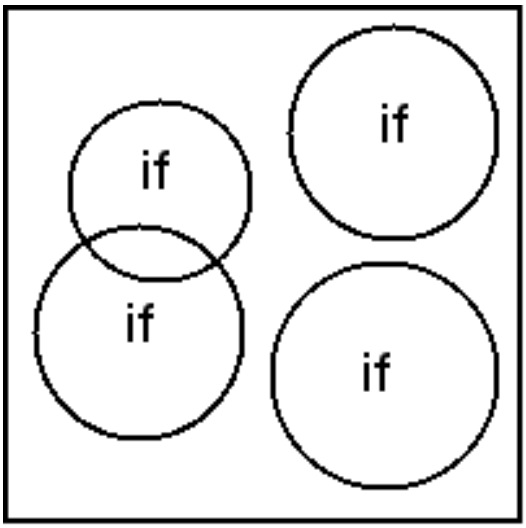
```
#include <iostream>
using namespace std;

int main() {
  // Mostra un missatge indicant si el valor llegit és un nombre positiu,
  // negatiu o zero.
  int x;
  cin >> x;

  if (x > 0) {
    cout << x << " es un nombre positiu." << endl;
  } else if (x < 0) {
    cout << x << " es un nombre negatiu." << endl;
  } else {
    cout << x << " es un 0." << endl;
  }
}
```

Índex

if / . . . / if



Seqüència de composicions condicionals

- Cada condició de la seqüència s'avalua indepentment.
- No s'ha de confondre amb la concatenació de composicions condicionals. A diferència de la concatenació aquí es pot executar més d'una opció cada vegada.
- Si només volem que s'executi una de les diferents opcions llavors NO és una seqüència de condicions.

Exemple erroni de seqüència de condicions

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

int main() {
  // Mostra un missatge indicant si el valor llegit és un nombre positiu,
  // negatiu o zero.
  int x;
  cin >> x;

  // ERROR: Aplicació incorrecte de la seqüència de condicions!!
  // Seria una concatenació de condicions
  if (x > 0) cout << x << " es un nombre positiu." << endl;
  if (x < 0) cout << x << " es un nombre negatiu." << endl;
  if (x == 0) cout << x << " es un 0." << endl;
}
```

Exemple correcte de seqüència de condicions

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

int main() {
  // Mostra un missatge indicant si el valor llegit és un nombre múltiple
  // de tres i/o té tres xifres.
  int x;
  cin >> x;

  if (x%3 == 0) {
    cout << x << " es un multiple de 3." << endl;
  }
  if (x >= 100 and x < 1000) {
    cout << x << " es un nombre de 3 xifres." << endl;
  }
}
```


Índex

Exercicis de repàs

1. Feu un programa que donats dos nombres enters, mostri per pantalla el màxim i el mínim.
2. Feu un programa que mostri per pantalla el màxim de tres nombres enters.
3. Feu un programa que donat un caràcter indiqui si és una vocal, una consonant o no és una lletra. En cas que sigui una lletra cal indicar si és majúscula o minúscula.
4. Feu un programa que donat un número entre 1 i 12 indiqui el nombre de dies que té aquest mes (sense tenir en compte els anys de traspàs). Intenta que tingui el menor nombre de línies possible.

Reconeixement i llicència

Aquesta sessió s'ha basat en el següent material:

- **Portal minidosis**. `http://www.minidosis.org/` fet per en *Pau Fernández*.
- **Material docent de l'assignatura PRO1 de la FIB**. `http://www.cs.upc.edu/~pro1/` fet per en *Jordi Cortadella*, *Ricard Gavaldà* i *Fernando Orejas*.

Tema 7

Composició repetitiva

Fonaments de Programació

Composició repetitiva

Bernardino Casas

bcasas@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

- ✓ Conèixer l'estructura repetitiva while, com funciona i com es forma.
- ✓ Saber seguir manualment un programa que inclogui un while.
- ✓ Saber usar la composició repetitiva per poder repetir diverses instruccions.
- ✓ Saber escriure programes que utilitzin bucles.

Índex

Necessitat del bucle
Bucle while
Exercicis de repàs

Índex

Per què serveix un bucle? (I)

Exercici. Feu un programa que escrigui per pantalla els nombres de l'1 al 10.

Una possible solució

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

int main() {
    cout << 1 << ' ' << 2 << ' ' << 3 << ' ';
    cout << 4 << ' ' << 5 << ' ' << 6 << ' ';
    cout << 7 << ' ' << 8 << ' ' << 9 << ' ';
    cout << 10 << endl;
}
```

Per què serveix un bucle? (II)

Exercici. Feu un programa que escrigui per pantalla els nombres 1 al 100.

- Podem intentar resoldre l'exercici seguint la mateixa idea que en l'exercici anterior, però la solució serà llarga i podem cometre errors en escriure-la.

Necessitat del bucle
Bucle while
Exercicis de repàs

Per què serveix un bucle? (II)

Exercici. Feu un programa que escrigui per pantalla els nombres 1 al 100.

- Podem intentar resoldre l'exercici seguint la mateixa idea que en l'exercici anterior, però la solució serà llarga i podem cometre errors en escriure-la.

7/30
Bernardino Casas Fonaments de Programació

Per què serveix un bucle? (III)

Exercici. Feu un programa que donat un nombre enter positiu *n* escrigui els nombres que van de 1 a *n*.

- Per aquest tercer exercici la solució sense cap estructura algorísmica addicional ja es veu que és força més complexa.
- Necessitem alguna manera de repetir instruccions $\longrightarrow$ bucles.

Necessitat del bucle
Bucle while
Exercicis de repàs

Per què serveix un bucle? (III)

Exercici. Feu un programa que donat un nombre enter positiu *n* escrigui els nombres que van de 1 a *n*.

- Per aquest tercer exercici la solució sense cap estructura algorísmica addicional ja es veu que és força més complexa.
- Necessitem alguna manera de repetir instruccions $\longrightarrow$ bucles.

8/30
Bernardino Casas Fonaments de Programació

Índex

Índex

Com funciona la composició repetitiva?

- Sintaxis:

```
while ( condicio ) {
    cos_bucle
}
```

- Semàntica:
  - La `condicio` és una expressió booleana.
  - Si la `condicio` avalua a cert llavors s'executen les sentències `cos_bucle`, i en acabar es torna a avaluar altra cop la `condicio`.
  - Si la `condicio` avalua a fals se surt del bucle i, en cas que n'hi hagi, executa la primera instrucció després del bucle (després de la **}**). .

Observacions importants sobre els bucles

- Les estructures repetitives també s'anomenen **estructures iteratives** o **bucles**.
- En el disseny de codis on hi ha estructures repetitives s'ha de tenir en compte que la condició ha de deixar de complir-se en algun moment (la condició ha d'avaluar a false) per tal de poder sortir de les repeticions.
- En el tema següent s'expliquen procediments per dissenyar estructures repetitives correctes.
- Existeixen d'altres estructures repetitives que veurem també més endavant.

Índex

Exemple 1: Enunciat

Problema. Feu un programa que donat un nombre enter positiu *n* escrigui els nombres que van de 1 a *n*.

Exemple 1: Solució

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5    // Programa que escriu els nombres 1 .. n
6    int n;
7    cin >> n;
8
9    int i = 1;
10   while (i <= n) {
11     cout << i << endl;
12     i = i+1;
13   }
14 }
```


Exemple 1: Un altra solució

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5    // Programa que escriu els nombres 1 .. n
6    int n;
7    cin >> n;
8
9    int i = 0;
10   while (i < n) {
11     i = i+1;
12     cout << i << endl;
13   }
14 }
```

Exemple 2: Enunciat

Problema: Què apareixerà per pantalla si en el canal d'entrada hi ha un 10?

```
1  int main() {
2    int n;
3    cin >> n;
4
5    int i = 1;
6    while (i <= n) {
7      cout << i << endl;
8      i = i*2;
9    }
10   cout << n << endl;
11   cout << i << endl;
12 }
```


Exemple 2: Solució

```
1
2
4
8
10
16
```

Exemple 3: Enunciat

Problema: Què apareixerà per pantalla si en el canal d'entrada hi ha 2 i 7?

```
1  int main() {
2    int a, b;
3    cin >> a >> b;
4
5    int i = a;
6    while (i <= b) {
7      if (i%2 == 0) ++i;
8      else i *= 2;
9      if (i%5 == 0) i = i + 1;
10   }
11   cout << i << endl;
12 }
```


Exemple 3: Solució

```
14
```

Exemple 4: Enunciat

Problema: Feu un programa que donat un nombre enter positiu, calculi quants dígits té aquest nombre.

Exemple 4: Solució

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5    // Programa que indica el nombre de dígits d'un nombre donat.
6    int n;
7    cin >> n;
8
9    int digits = 0;
10   while (n > 9) {
11     digits = digits + 1;
12     n = n/10;
13   }
14   cout << digits + 1 << endl;
15 }
```

Exemple 5: Enunciat

Problema: Feu un programa que donat un nombre enter positiu *n* indiqui si aquest nombre és primer.

Exemple 5: Solució

```
1  [...]
2  int main() {
3    // Programa que indica si un nombre donat és primer.
4    int n;
5    cin >> n;
6
7    int divisor = 2;
8    bool es_primer = true;
9    while (divisor < n) {
10     if (n%divisor == 0) es_primer = false;
11     ++divisor;
12   }
13   if (es_primer) cout << "Es primer" << endl;
14   else cout << "No es primer" << endl;
15 }
```

Exemple 5: Solució

```
1  [...]
2  int main() {
3    // Programa que indica si un nombre donat és primer.
4    int n;
5    cin >> n;
6
7    int divisor = 2;
8    bool es_primer = true;
9    while (divisor < n) {
10     if (n%divisor == 0) es_primer = false;
11     ++divisor;
12   }
13   if (es_primer) cout << "Es primer" << endl;
14   else cout << "No es primer" << endl;
15 }
```

COMPTE! El nombre 1 no és primer

Exemple 5: Solució mooolt millorable

```
1  [...]
2  int main() {
3    // Programa que indica si un nombre donat és primer.
4    int n;
5    cin >> n;
6
7    int divisor = 2;
8    bool es_primer = n>=2;
9    while (divisor < n) {
10     if (n%divisor == 0) es_primer = false;
11     ++divisor;
12   }
13   if (es_primer) cout << "Es primer" << endl;
14   else cout << "No es primer" << endl;
15 }
```

Exemple 5: Millorant la solució

- Tan aviat com trobem un nombre que és divisor de n, ja no és necessari comprovar si hi ha algun divisor més. Una possible millora seria aturar les iteracions en el moment en que es trobi un divisor.
- Atès que els nombres parells (exceptuant el 2) no són primers, es pot incrementar el divisor de dos en dos.
- Es pot aprofitar el teorema d'Erastótenes que diu $i \leq \sqrt{n} \equiv i^2 \leq n$ per fer un menor nombre d'iteracions.

Índex

Exercicis de repàs

1. Feu un programa que donats dos nombres enters, calculi la multiplicació d'aquests dos nombres usant només els operadors de suma i resta.
2. Feu un programa que donats dos nombres enters a i b (sent $a < b$), escrigui tots els nombres enters que hi ha a l'interval [a, b].
3. Feu la versió millorada del programa que calculi si un nombre és primer.

Solució exercici 3

```
1  [...]
2  int main() {
3    // Programa que indica si un nombre donat és primer.
4    int n;
5    cin >> n;
6    bool es_primer = n>=2;
7
8    if (n != 2) {
9      int div = 3;
10     while (div*div <= n and es_primer) {
11       if (n%div == 0) es_primer = false;
12       else div = div + 2;
13     }
14   }
15   if (es_primer) cout << "Es primer" << endl;
16   else cout << "No es primer" << endl;
17 }
```

Reconeixement i llicència

Aquesta sessió s'ha basat en el següent material:

- **Portal minidosis**. `http://www.minidosis.org/` fet per en *Pau Fernández*.
- **Material docent de l'assignatura PRO1 de la FIB**. `http://www.cs.upc.edu/~pro1/` fet per en *Jordi Cortadella*, *Ricard Gavaldà* i *Fernando Orejas*.

Tema 8

Seqüències i esquemes iteratius

Fonaments de Programació

Seqüències i esquemes iteratius

Bernardino Casas

bcasas@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

- ✓ Saber identificar els elements bàsics d'una seqüència: primer, següent i últim element.
- ✓ Conèixer els esquemes iteratius de recorregut i cerca.
- ✓ Saber reconèixer quin esquema iteratiu cal aplicar per resoldre un problema.

Índex

Seqüències
Esquemes iteratius sobre seqüències
Exercicis de repàs

Índex

1. Seqüències
2. Esquemes iteratius sobre seqüències
   - Esquema de recorregut
   - Esquema de cerca
   - Metologia del disseny iteratiu
3. Exercicis de repàs

Què és una seqüència? (I)

Definició 1: Seqüència

Una *seqüència* és una llista ordenada d'elements, on n'hi ha un primer element (sempre i quan no sigui la seqüència buida) i un últim element.

En una seqüència podem identificar:

- **Primer element**. El primer element de la seqüència que es vol tractar.
- **Relació de successió**. El mecanisme que permet obtenir el següent element de la seqüència a partir d'un element donat.
- **Últim element**. Ens indica si hem arribat al final de la seqüència.

Què és una seqüència? (II)

- Les seqüències apareixen de manera implícita en multitud de problemes.
- De fet, **SEMPRE** que resolem un problema usant una estructura repetitiva, estem tractant una seqüència.
- Exemples de problemes que tracten seqüències:
  - Calcular la suma dels 100 primers naturals.
  - Calcular el factorial d'un nombre.
  - Calcular la suma dels dígits d'un nombre.
  - Comprovar si un nombre és primer.

Exemple de seqüència

Problema: Feu un programa que mostri la suma dels 100 primers naturals.

- La seqüència implícita en aquest problema es pot identificar d'aquesta manera:

  Primer element `i` = 1
  Relació de successió incrementar `i` en 1
  Últim element el darrer element és el 100 (`i` == 100)

Índex

Què és un esquema?

Definició 2: Esquema algorísmic

Un *esquema algorísmic* és una plantilla de solució d'un problema presentat en format algorísmic, de manera que es pot aplicar a un conjunt de problemes similars simplement adequant-lo al problema particular que es vulgui solucionar.

- Un esquema algorísmic és una mena de patró d'algorisme que, amb petites adaptacions, resol tota una família de problemes semblants.

Esquemes iteratius

- Qualsevol algorisme que es pot plantejar sobre seqüències pertany a una de les dues classes següents:
  - **Recorregut** algorismes que recorren i tracten tots els elements d'una seqüència.
  - **Cerca** algorismes que recorren els elements d'una seqüència buscant un element que compleixi una propietat determinada i s'atura quan l'ha trobat.
- La utilització d'esquemes és útil i pràctic en el disseny d'algorismes iteratius.

Avantatges dels esquemes

Entre els avantatges de fer servir esquemes es compten:

- El disseny ja no recau en la inspiració sinó en la identificació correcta del problema.
- Si es tria l'esquema adequat i l'adaptació es fa correctament, el programa funcionarà doncs els esquemes són correctes.

Índex

Quan s'aplica l'esquema de recorregut?

- L'esquema de recorregut s'aplica en els problemes on s'han de tractar tots els elements d'una seqüència.
- Exemples:
  - Mostrar per pantalla tots els múltiples de 5 menors de 1000.
  - Sumar tots els elements d'una seqüència d'enters.
  - Mostrar per pantalla les vocals d'una cadena de caràcters.
  - ...
- A continuació es mostra una versió genèrica de l'esquema de recorregut.

Esquema genèric de recorregut

```
obtenir_primer_element
while ( not ultim_element ) {
    tractar_element
    obtenir_seguent_element
}
tractar_ultim_element
[tractament_final]
```

El `tractament_final` és optatiu.

Índex

Quan s'aplica l'esquema de cerca?

- L'esquema de cerca s'aplica en els problemes on s'ha de determinar si existeix un element en una seqüència que compleixi una propietat donada.
- Exemples:
  - Comprovar si un nombre es primer.
  - Determinar si una seqüència d'enters donada està ordenada de forma creixent.
  - Determinar si en una cadena de caràcters hi ha alguna paraula que comenci per la lletra 'p'.
  - ...
- A continuació es mostren dues versions de l'esquema de cerca. En la segona versió es fa servir una variable booleana.

Esquema genèric de cerca – 1a versió

```
obtenir_primer_element
while ( not elem_compleix_cond and
        not ultim_element ) {
   obtenir_seguent_element
}
if (elem_compleix_cond) {
   tractament_trobat
} else {
   tractament_no_trobat
}
```


Esquema genèric de cerca – 2a versió

```
bool trobat = false;
obtenir_primer_element
while ( not trobat and not ultim_element ) {
   if (elem_compleix_cond) {
      trobat = true;
   } else {
      obtenir_seguent_element
   }
}
if (trobat) {
   tractament_trobat
} else {
   tractament_no_trobat
}
```

Índex

Passos per resoldre un problema iteratiu (I)

A grans trets, el disseny de programes mitjançant esquemes té dues fases:

1. Identificació del problema entre mans i per tant de l'esquema a fer servir per solucionar-lo.
2. Adaptació de l'esquema al problema particular que es vulgui resoldre.

Passos per resoldre un problema iteratiu (II)

1. **Identificar la seqüència**.
   - Com s'obté el primer element.
   - Com s'obté el següent element.
   - Quin és l'últim element.
2. **Identificar l'esquema de tractament adient**. Recorregut o cerca
3. Si és un recorregut, **identificar el tractament elemental**. Si és una cerca, **identificar la propietat que estem buscant**.
4. Opcional. Identificar les **inicialitzacions** que cal aplicar abans del while i el **tractament final**.
5. **Aplicar l'esquema algorísmic corresponent**.

Índex

Exercicis de repàs

1. Determina el primer, el següent i l'últim element de les següents seqüències:
   - 10, 20, 30, 40, . . . , 100
   - 500, 400, 300, 200, 100
   - 2, 5, 11, 23, 47, 95
2. Indica si els següents programes s'haurien de fer seguint un esquema de recorregut o cerca:
   - Calcular la suma dels *n* primers nombres senars.
   - Determinar si un nombre enter positiu donat conté algun 0.
   - Determinar si un nombre enter positiu donat conté tants 0's com 1's.
   - Determinar si una cadena de caràcters és capicua.
3. Identifica la seqüència implícita en cadascun dels programes anteriors.

Reconeixement i llicència

Aquesta sessió s'ha basat en el següent material:

- **Seqüències i esquemes algorísmics**. Apunts d'Informàtica fet per na *Neus Català* i en *Mario Martín*.
- **Material docent de l'assignatura PRO1 de la FIB**. `http://www.cs.upc.edu/~pro1/` fet per en *Jordi Cortadella*, *Ricard Gavaldà* i *Fernando Orejas*.

Tema 9

Invariants i exemples d'aplicació dels esquemes iteratius

Fonaments de Programació

Invariants i exemples d'aplicació dels esquemes algorísmics

Bernardino Casas

bcasas@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

- ✓ Conèixer què és un invariant i per què serveix.
- ✓ Saber escriure de manera informal l'invariant d'un bucle senzill.
- ✓ Saber reconèixer quin esquema iteratiu caldria aplicar per resoldre un problema.
- ✓ Saber aplicar la metodologia de disseny d'algorismes iteratius per resoldre un problema.

Índex

Índex

Què és un invariant?

Definició 1: Invariant d'un bucle

L'*invariant d'un bucle* és una condició que es compleix en cada iteració del bucle.

- Els invariants ajuden a:
  - definir com les variables han de ser inicialitzades abans d'un bucle.
  - definir la condició necessària per arribar a la postcondició.
  - definir el cos del bucle.
  - detectar si un bucle acaba.
- L'invariant és una proposició que es compleix: al principi del bucle, al principi de cada iteració i al final del bucle.

Recomanació sobre ús d'invariants

- És crucial, però no sempre fàcil, triar un bon invariant.
- Es recomana usar raonament basat en invariants per idear qualsevol bucle (possiblement de manera **informal**).
- Per bucles més complexos es recomana usar raonament **formal** basat en invariant.
- En aquesta primera assignatura de programació farem un ús dels invariants de manera informal. En properes assignatures es veuran invariants formals.

Raonament general per bucles

```
Inicialitzacio;
// Invariant
while (condicio) {
    // Invariant ∧ condicio
    Cos del bucle
}
// Invariant ∧ ¬ condicio
```

L'invariant és una proposició que es compleix:

- al principi del bucle
- al principi de cada iteració
- al final del bucle

Índex

Índex

Passos per resoldre un problema iteratiu

1. **Identificar la seqüència**.
   - Com s'obté el primer element.
   - Com s'obté el següent element.
   - Quin és l'últim element.
2. **Identificar l'esquema de tractament adient**. Recorregut o cerca
3. Si és un recorregut, **identificar el tractament elemental**. Si és una cerca, **identificar la propietat que estem buscant**.
4. Opcional. Identificar les **inicialitzacions** que cal aplicar abans del while i el **tractament final**.
5. **Aplicar l'esquema algorísmic corresponent**.

Índex

Exemple 1: Enunciat

Problema: Feu un programa que calculi el factorial d'un enter positiu *n*.

Idea: Es tracta de fer el producte de la seqüència de nombres de 1 fins a *n*:

$$n! = 1 \cdot 2 \cdot 3 \ldots (n-1) \cdot n$$

Exemple 1: Enunciat

Problema: Feu un programa que calculi el factorial d'un enter positiu *n*.

Idea: Es tracta de fer el producte de la seqüència de nombres de 1 fins a *n*:

$$n! = 1 \cdot 2 \cdot 3 \ldots (n-1) \cdot n$$

Exemple 1: 1. Identificació de la seqüència

Problema: Feu un programa que calculi el factorial d'un enter positiu *n*.

Els elements a tractar són els enters entre 1 a *n* on:

- Primer element: e = 1
- Donat un element *e* que ja hem tractat obtenir el següent: e = e +1
- Últim element: e = n

Exemple 1: 2. Recorregut o cerca?

Problema: Feu un programa que calculi el factorial d'un enter positiu *n*.

Quin esquema segueix? Recorregut o cerca?

Invariants
Exemples
Exercicis de repàs
Recordatori: Metodologia
Exemple 1: factorial
Exemple 2: nombre primer

Exemple 1: 2. Recorregut o cerca?

Problema: Feu un programa que calculi el factorial d'un enter positiu *n*.

Quin esquema segueix? Recorregut o cerca?

- Els elements a tractar (multiplicar) són **tots** els que hi ha entre 1 i *n*.
- Per tant és un **recorregut**.

Exemple 1: 3. Tractament

Problema: Feu un programa que calculi el factorial d'un enter positiu *n*.

Què hem de fer amb cada un dels elements de la seqüència?

Invariants
Exemples
Exercicis de repàs
Recordatori: Metodologia
Exemple 1: factorial
Exemple 2: nombre primer

Exemple 1: 3. Tractament

Problema: Feu un programa que calculi el factorial d'un enter positiu *n*.

Què hem de fer amb cada un dels elements de la seqüència?

- Tindrem un variable *f* on guardarem el factorial.
- Al final s'ha de complir que

$$f = 1 \cdot 2 \ldots (n-1) \cdot n$$

- Per tant, cada element s'haurà de multiplicar per *f*

```
f = f*e
```

Exemple 1: 4. Inicialitzacions i tractament final

Problema: Feu un programa que calculi el factorial d'un enter positiu *n*.

Com s'haurà d'inicialitzar la variable *f*?

Quin és el tractament final?

Invariants
Exemples
Exercicis de repàs
Recordatori: Metodologia
Exemple 1: factorial
Exemple 2: nombre primer

Exemple 1: 4. Inicialitzacions i tractament final

Problema: Feu un programa que calculi el factorial d'un enter positiu *n*.

Com s'haurà d'inicialitzar la variable *f*?

- *f* s'inicialitzarà a 1 (l'element neutre de l'operació producte).

Quin és el tractament final?

Exemple 1: 4. Inicialitzacions i tractament final

Problema: Feu un programa que calculi el factorial d'un enter positiu *n*.

Com s'haurà d'inicialitzar la variable *f*?

- *f* s'inicialitzarà a 1 (l'element neutre de l'operació producte).

Quin és el tractament final?

- Es mostrarà la variable *f* pel canal de sortida:

```
cout << f << endl;
```


Exemple 1: 5. Aplicació (I)

Esquema general de recorregut

```
obtenir_primer_element
while ( not ultim_element ) {
    tractar_element
    obtenir_seguent_element
}
tractar_ultim_element
[tractament_final]
```

Exemple 1: 5. Aplicació (II)

```
int e = 1;
while ( not ultim_element ) {
    tractar_element
    obtenir_seguent_element
}
tractar_ultim_element
[tractament_final]
```


Exemple 1: 5. Aplicació (III)

```
int e = 1;
while (e < n) {
    tractar_element
    obtenir_seguent_element
}
tractar_ultim_element
[tractament_final]
```

Exemple 1: 5. Aplicació (IV)

```
int e = 1
while (e < n) {
    tractar_element
    e = e + 1;
}
tractar_ultim_element
[tractament_final]
```


Exemple 1: 5. Aplicació (V)

```
int e = 1;
int f = 1;
while (e < n) {
    f = f*e;
    e = e + 1;
}
tractar_ultim_element
[tractament_final]
```

Exemple 1: 5. Aplicació (VI)

```
int e = 1;
int f = 1;
while (e < n) {
    f = f*e;
    e = e + 1;
}
f = f*e;
[tractament_final]
```


Exemple 1: 5. Aplicació (VII)

```
int e = 1;
int f = 1;
while (e < n) {
    f = f*e;
    e = e + 1;
}
f = f*e;
cout << f << endl;
```

Exemple 1: 5. Aplicació (VIII)

```
int e = 1;
int f = 1;
while (e <= n) {
    f = f*e;
    e = e + 1;
}
cout << f << endl;
```


Exemple 1: 5. Aplicació (IX)

```
...
int main() {
    // Mostra el factorial d'un nombre natural.
    int n;
    cin >> n;

    int e = 1;
    int f = 1;
    while (e <= n) {
        // Inv:  f = e!
        f = f*e;
        e = e + 1;
    }
    cout << f << endl;
}
```

Índex

Exemple 2: Enunciat

Problema: Feu un programa que donat un enter positiu, indiqui si és un nombre primer.

Idea: Es tracta de comprovar si algun dels nombres que van de 2 fins a *n-1* és divisor de n.

Exemple 2: Enunciat

Problema: Feu un programa que donat un enter positiu, indiqui si és un nombre primer.

Idea: Es tracta de comprovar si algun dels nombres que van de 2 fins a *n-1* és divisor de n.

Exemple 2: 1. Identificació de la seqüència

Problema: Feu un programa que donat un enter positiu, indiqui si és un nombre primer.

Els elements a tractar són els enters entre 2 a *n-1* on:

- Primer element: e = 2
- Donat un element *e* que ja hem tractat obtenir el següent: e = e + 1
- Últim element: e = n - 1

Exemple 2: 2. Recorregut o cerca?

Problema: Feu un programa que donat un enter positiu, indiqui si és un nombre primer.

Quin esquema segueix? Recorregut o cerca?

Invariants
Exemples
Exercicis de repàs
Recordatori: Metodologia
Exemple 1: factorial
Exemple 2: nombre primer

Exemple 2: 2. Recorregut o cerca?

Problema: Feu un programa que donat un enter positiu, indiqui si és un nombre primer.

Quin esquema segueix? Recorregut o cerca?

- Es vol trobar si algun dels elements de la seqüència és un divisor propi de n. En el moment que trobem un divisor ja no cal mirar la resta dels elements ja que n no serà un nombre primer.
- Per tant, és una **cerca**.

Exemple 2: 3. Condició cerca

Problema: Feu un programa que donat un enter positiu, indiqui si és un nombre primer.

Quina és la condició de la cerca?

Invariants
Exemples
Exercicis de repàs
Recordatori: Metodologia
Exemple 1: factorial
Exemple 2: nombre primer

Exemple 2: 3. Condició cerca

Problema: Feu un programa que donat un enter positiu, indiqui si és un nombre primer.

Quina és la condició de la cerca?

- Cal comprovar si l'element actual és divisor propi de n

```
n%e == 0
```

Exemple 2: 4. Inicialitzacions i tractament final

Problema: Feu un programa que donat un enter positiu, indiqui si és un nombre primer.

Quin és el tractament final?

Invariants
Exemples
Exercicis de repàs
Recordatori: Metodologia
Exemple 1: factorial
Exemple 2: nombre primer

Exemple 2: 4. Inicialitzacions i tractament final

Problema: Feu un programa que donat un enter positiu, indiqui si és un nombre primer.

Quin és el tractament final?

- Es mostrarà un missatge indicant si n és un nombre primer.
- Tractament trobat:

```
cout << "no primer"<< endl;
```

- Tractament no trobat:

```
cout << "primer"<< endl;
```

Exemple 2: 5. Aplicació (I)

Esquema general de cerca

```
bool trobat = false;
obtenir_primer_element
while (not trobat and not ultim_element) {
    if (elem_compleix_cond) {
        trobat = true;
    } else {
        obtenir_seguent_element
    }
}
if (trobat) {
    tractament_trobat
} else {
    tractament_no_trobat
}
```


Exemple 2: 5. Aplicació (II)

```
bool trobat = false;
int e = 2;
while (not trobat and e < n) {
  if (n%e == 0) trobat = true;
  else e = e + 1;
}
if (trobat) cout << "no primer" << endl;
else cout << "primer" << endl;
```

Exemple 2: 5. Aplicació (II)

```
bool trobat = false;
int e = 2;
while (not trobat and e < n) {
  if (n%e == 0) trobat = true;
  else e = e + 1;
}
if (trobat) cout << "no primer" << endl;
else cout << "primer" << endl;
```

Compte amb el nombre 1!!
No és un nombre primer!!

Exemple 2: 5. Aplicació (III)

```
...
int main() {
  // Indica si un nombre natural és primer.
  int n;
  cin >> n;

  bool trobat = n <= 1;
  int e = 2;
  while (not trobat and e < n) {
    // Inv:  tots els enters entre [2..e-1] no divideixen
    //       a n
    if (n%e == 0) trobat = true;
    else e = e + 1;
  }
  if (trobat) cout << "no primer" << endl;
  else cout << "primer" << endl;
}
```

Exemple 2: Millora (I)

- Si un nombre *n* no és primer, es poden trobar dos nombres *a* i *b*, tals que:
  $n = a * b$, amb $1 < a \leq b < n$
  i amb la següent propietat: $a \leq \sqrt{n}$
- No hi ha necessitat de comprovar fins el nombre *n-1*. Ens podem aturar abans.
- Nota: $a \leq \sqrt{n}$ és equivalent a $a^2 \leq n$
- Amb aquesta millora aconseguim fer menys iteracions en el bucle per indicar si un nombre no és primer.

Exemple 2: Millora (II)

```
...
int main() {
  // Indica si un nombre natural és primer.
  int n;
  cin >> n;

  bool trobat = n <= 1;
  int e = 2;
  while (not trobat and e*e <= n) {
    // Inv:  tots els enters entre [2..e-1] no divideixen
    //       a n
    if (n%e == 0) trobat = true;
    else e = e + 1;
  }
  if (trobat) cout << "no primer" << endl;
  else cout << "primer" << endl;
}
```

Índex

Exercicis de repàs

Seguiu la metodologia de disseny iteratiu per resoldre els següents exercicis. A més a més, troba l'invariant de cada bucle.

1. Feu un programa que donat un nombre enter n, calculi la suma dels n primers nombres senars.
2. Feu un programa que donat un enter positiu, determini si aquest nombre conté algun 0.
3. Feu un programa que donat un enter positiu, determini si aquest nombre conté tants 0's com 1's.

Reconeixement i llicència

Aquesta sessió ha utilitzat el següent material:

- **Seqüències i esquemes algorísmics**. Apunts d'Informàtica fets per na *Neus Català* i en *Mario Martín*.
- **Material docent de l'assignatura PRO1 de la FIB**. `http://www.cs.upc.edu/~pro1/` fet per en *Jordi Cortadella*, *Ricard Gavaldà* i *Fernando Orejas*.

Tema 10

Fluxos de dades i esquemes iteratius

Fonaments de Programació

Fluxos de dades i esquemes iteratius

Bernardino Casas

bcasas@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

- ✓ Saber identificar els elements bàsics d'un flux de dades: primer, següent i últim element.
- ✓ Conèixer els esquemes iteratius de recorregut i cerca sobre fluxos de dades.
- ✓ Saber reconèixer quin esquema iteratiu caldria aplicar per resoldre un problema de fluxos de dades.
- ✓ Saber aplicar la metodologia de disseny d'algorismes iteratius per resoldre problemes de fluxos de dades.

Índex

Índex

Què és un flux de dades? (I)

Definició 1: Flux de dades

Un *flux de dades* (anglès: stream) és una seqüència d'informació que va variant en el temps.

- Un flux de dades es pot veure com una cinta transportadora on els elements es van tractant d'un en un.

Què és un flux de dades? (II)

- Els fluxos de dades apareixen en moltes circumstàncies però especialment en processos de comunicació, on un emissor (ordinador servidor) envia dades de forma contínua a un receptor (ordinador client o usuari).
- Exemples:
  - Un radar de carretera.
  - Un aparell metereològic que mesura la velocitat del vent.
  - Un sensor que indiqui si una porta està oberta o tancada.
  - Un aparell que pren les constants vitals d'un pacient.
  - ...

Cal programar bé els fluxos de dades

COMPTE!!

Una bona tècnica en el tractament dels fluxos de dades permet treballar amb les dades **a mida que es van rebent** sense haver d'esperar que es rebi la totalitat de les dades.

Com simulem els fluxos?

- Atès que no tenim cap sensor del qual poguem obtenir un flux de dades, els fluxos els haurem de simular.

  Els programes sobre fluxos que dissenyem sempre rebran les dades pel canal d'entrada (per ex. teclat) d'una a una.

- Per exemple: en el cas d'un flux d'enters, alimentarem el programa amb nombres entrant-los d'un en un separant-los amb un salt de línia o un espai (nombre nombre . . .).
- Depenent de com estigui definit el flux acabarem el flux quan entrem un valor que anomenarem sentinella o, si estem utilitzant el teclat, pulsem ctrl D .

Quan acaba un flux de dades?

- Com que els fluxos de dades tenen una longitud finita però indeterminada, per reconèixer la finalització de la transmissió de les dades es pot fer de tres maneres:

Tipus de fluxos de dades

- S'indica el nombre d'elements pel que està format el flux.
- La transmissió acaba amb un element especial anomenat sentinella.
- El final de la transmissió es detecta amb l'`EOF` (End Of File) mitjançant la instrucció `cin`.

Quan acaba un flux de dades? (II)

COMPTE!!

Cada flux tindrà associat en la seva descripció la manera com es reconeix la finalització de la transmissió de les dades.
S'indicarà si el flux té un nombre determinat d'elements, un element final distintiu o cap dels anteriors (EOF).

Índex

Flux amb el nombre d'elements acotat

- El cas més senzill de flux de dades és el que ens indiquen el nombre d'elements del flux.
- En aquest tipus de flux només cal:

1. llegir inicialment el nombre que indica quants elements té el flux.
2. llegir tants elements com s'ha indicat.

Índex

Sentinella

Definició 2: Sentinella

Un sentinella és el darrer element d'un flux de dades i la seva única finalitat és assenyalar el final de la transmissió (de les dades).

- Exemples de fluxos de dades amb sentinella:
  - Flux d'enters acabat en `-1`.
  - Flux de caràcters acabat en `'.'`.
  - Flux de reals acabat en `0.0`.
  - ...

Altres consideracions sobre el sentinella

- El sentinella **no s'ha de tractar** com la resta d'elements. Ni en els recorreguts ni en les cerques.
- Estrictament parlant, el sentinella **no forma part del flux** i no és més que un simple marcador de final de la transmissió.
- El sentinella ha de ser **diferenciable de la resta d'elements** del flux (un element especial), sinó no sabríem quan un element forma part del flux o és el sentinella.

Índex

Llegint amb cin (I)

- La instrucció de lectura `cin>>` es pot usar per detectar el final d'un flux de dades i llegir un nou element a la vegada.
- El canal d'entrada `cin` es pot tractar com una expressió booleana ja que automàticament es converteix a booleà i val:

  true si l'operació acaba correctament.
  false si l'operació falla.

- Quan la lectura retorna `false` pot ser perquè no hi hagi més dades disponibles (`EOF`) o les dades no hagin estat formatejades correctament (intentar llegir un real quan l'entrada és una cadena de caràcters).

Llegint amb cin (II)

COMPTE!!

Si la lectura retorna un valor fals (el final del flux es detecta), la variable on es guarda la informació no es veu modificada.

Índex

Aplicant els esquemes (I)

- El disseny d'algorismes que tracten fluxos de dades es pot fer amb els esquemes de cerca i recorregut que ja coneixem, doncs un flux de dades no és més que una seqüència.
- En el cas de fluxos acotats es pot aplicar l'esquema iteratiu general afegint una variable per comptar els elements processats.
- En els altres dos casos, atès que el final de la seqüència ve marcat per un sentinella o amb EOF, tindrem dues per versions per l'esquema de recorregut i dues pel de cerca.

Aplicant les esquemes (II)

	Flux acotat [1]	Flux sentinella [2]	Flux EOF [3]
primer	`cin` >> e	`cin` >> e	`cin` >> e
següent	`cin` >> e	`cin` >> e	`cin` >> e
últim	i == n	e == sentinella	`cin` >> e

[1] Cal comptar el nombre d'elements llegits. Per això cal una variable que porti aquest comptatge (variable *i*).
[2] L'element sentinella no s'ha de tractar.
[3] La mateixa operació de lectura ens indica si hem acabat de llegir elements del flux.

Índex

Esquema general de recorregut

Esquema general de recorregut

```
obtenir_primer_element
while ( not ultim_element ) {
  tractar_element
  obtenir_seguent_element
}
tractar_ultim_element
[tractament_final]
```


Recorregut per fluxos de dades acotats

```
obtenir_nombre_total_elements
i = 0
while ( i < nombre_total_elements ) {
   obtenir_element
   tractar_element
   ++i;
}
[tractament_final]
```

El `tractament_final` és optatiu.

Recorregut per fluxos de dades amb sentinella

```
obtenir_primer_element
while ( not element_sentinella ) {
    tractar_element
    obtenir_seguent_element
}
[tractament_final]
```

El `tractament_final` és optatiu.

Recorregut per fluxos de dades amb EOF

```
while ( obtenir_element ) {
    tractar_element
}
[tractament_final]
```

El `tractament_final` és optatiu.

Índex

Esquema general de cerca

Esquema general de cerca

```
bool trobat = false;
obtenir_primer_element
while ( not trobat and not ultim_element ) {
  if (elem_compleix_cond) {
    trobat = true;
  } else {
    obtenir_seguent_element
  }
}
if (trobat) {
  tractament_trobat
} else {
  tractament_no_trobat
}
```

Cerca per fluxos de dades acotats

```
obtenir_nombre_total_elems
bool trobat = false;
i = 0
while (not trobat and i < nombre_total_elems) {
    obtenir_element
    if (elem_compleix_cond)  trobat = true;
    else ++i;
}
if (trobat) {
    tractament_trobat
} else {
    tractament_no_trobat
}
```


Cerca per fluxos de dades amb sentinella

```
bool trobat = false;
obtenir_primer_element
while (not trobat and not element_sentinella) {
    if (elem_compleix_cond) {
        trobat = true;
    } else {
        obtenir_seguent_element
    }
}
if (trobat) {
    tractament_trobat
} else {
    tractament_no_trobat
}
```

Cerca per fluxos de dades amb EOF

```
bool trobat = false;
while (not trobat and obtenir_element) {
    if (elem_compleix_cond) {
        trobat = true;
    }
}
if (trobat) {
    tractament_trobat
} else {
    tractament_no_trobat
}
```


Eliminar el booleà de l'esquema?

COMPTE!!

Per qüestions d'eficiència, és possible eliminar el booleà dels esquemes de cerca?

Índex

Exemple 1: Enunciat

Problema: Feu un programa que donat un flux d'enters, calculi i mostri per pantalla la seva suma.

Exemple 1: Identificació flux

Problema: Feu un programa que donat un flux d'enters, calculi i mostri per pantalla la seva suma.

Hi ha sentinella? S'indiquen el nombre d'elements del flux?
No hi ha sentinella, ja que no s'indica com acaba el flux.
Tampoc s'indica quants elements té el flux.

Exemple 1: Recorregut o cerca?

Problema: Feu un programa que donat un flux d'enters, calculi i mostri per pantalla la seva suma.

Quin esquema segueix? Recorregut o cerca?
És un recorregut ja que hem de tractar **SEMPRE** tots els elements del flux per tal de calcular la suma.

Per tant cal identificar el tractament que s'ha de fer amb cada element del flux de dades.

Exemple 1: Tractament

Problema: Feu un programa que donat un flux d'enters, calculi i mostri per pantalla la seva suma.

Quin és el tractament?
Cal tenir una variable on anem emmagatzemant la suma dels elements del flux. Per tant, cada element l'hem d'afegir a aquesta variable:

$$s = s + e$$

Exemple 1: Inicialitzacions i tractament final

Problema: Feu un programa que donat un flux d'enters, calculi i mostri per pantalla la seva suma.

Inicialitzacions
Cal inicialitzar la variable *s*:

```
int s = 0;
```

Tractament final
Mostrar el resultat al final de tot.

```
cout << s << endl;
```

Exemple 1: Recordatori esquema de recorregut EOF

```
while ( obtenir_element ) {
    tractar_element
}
[tractament_final]
```

El `tractament_final` és optatiu.

Exemple 1: Solució

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

int main() {
  // Mostra per pantalla la suma dels elements d'un flux d'enters
  int e, s = 0;

  while (cin >> e) {
    // Inv: s conté la suma dels elements del flux tractats fins al moment.
    s = s + e;
  }
  cout << s << endl;
}
```

Exemple 2: Enunciat

Problema: Feu un programa que donat un flux de caràcters acabat en `'.'`, indiqui quants cops apareix el caràcter `'a'`.

Exemple 2: Identificació flux

Problema: Feu un programa que donat un flux de caràcters acabat en `'.'`, indiqui quants cops apareix el caràcter `'a'`.

Hi ha sentinella? S'indiquen el nombre d'elements del flux?

Hi ha sentinella i és el '.' (com s'indica a l'enunciat).

Exemple 2: Recorregut o cerca?

Problema: Feu un programa que donat un flux de caràcters acabat en `'.'`, indiqui quants cops apareix el caràcter `'a'`.

Quin esquema segueix? Recorregut o cerca?
És un recorregut ja que cal recórrer tots els elements del flux, sinó no sabem si l'últim element serà una lletra 'a'.

Per tant, cal identificar el tractament que s'ha de fer amb cada element del flux de dades.

Exemple 2: Tractament

Problema: Feu un programa que donat un flux de caràcters acabat en `'.'`, indiqui quants cops apareix el caràcter `'a'`.

Quin és el tractament?
Cal tenir una variable on anem emmagatzemant el nombre de vegades que apareix la lletra 'a' en el flux. I a més cal comprovar per cada element del flux si és una 'a':

```
if (c == 'a') ++num;
```

Exemple 2: Inicialitzacions i tractament final

Problema: Feu un programa que donat un flux de caràcters acabat en `'.'`, indiqui quants cops apareix el caràcter `'a'`.

Inicialitzacions
Cal inicialitzar la variable *num*:

```
int num = 0;
```

Tractament final
Mostrar el resultat al final de tot.

```
cout << "Hi ha "<< num << " lletres a"<< endl;
```


Exemple 2: Recordatori esquema de recorregut amb sentinella

```
obtenir_primer_element
while ( not element_sentinella ) {
    tractar_element
    obtenir_seguent_element
}
[tractament_final]
```

El `tractament_final` és optatiu.

Exemple 2: Solució

```
...
int main() {
  // Indica quants cops apareix el caràcter 'a' en un flux de caràcters
  // acabat en '.'.
  char c;

  cin >> c;
  int num = 0;
  while (c != '.') {
    // Inv: num conté el nombre de vegades que ha aparegut el caràcter 'a'
    // en els elements tractats fins al moment.
    if (c == 'a') ++num;
    cin >> c;
  }
  cout << "Hi␣ha␣" << num << "␣lletres␣a" << endl;
}
```


Exemple 3: Enunciat

Problema: Feu un programa que donat un flux d'enters positius, indiqui si tots els nombres són parells.

Exemple 3: Identificació flux

Problema: Feu un programa que donat un flux d'enters positius, indiqui si tots els nombres són parells.

Hi ha sentinella? S'indiquen el nombre d'elements del flux?
No hi ha sentinella, ja que no s'indica com acaba el flux.
Tampoc s'indica quants elements té el flux.

Exemple 3: Recorregut o cerca?

Problema: Feu un programa que donat un flux d'enters positius, indiqui si tots els nombres són parells.

Quin esquema segueix? Recorregut o cerca?
És una cerca ja que NO sempre cal tractar tots els elements del flux per tal de saber si tots els elements del flux són parells. Si trobem un element senar no cal mirar la resta.

Per tant cal identificar la condició de la cerca del flux de dades.

Exemple 3: Condició de la cerca

Problema: Feu un programa que donat un flux d'enters positius, indiqui si tots els nombres són parells.

Quina és la condició de la cerca?
Estem buscant si algun element del flux no és parell, per tant la condició serà:

$$e\%2 != 0$$

Exemple 3: Inicialitzacions i tractament final

Problema: Feu un programa que donat un flux d'enters positius, indiqui si tots els nombres són parells.

Inicialitzacions No hi ha cap inicialització a fer.

Tractament trobat

```
cout << "No tots els nombres son parells"<< endl;
```

Tractament no trobat

```
cout << "Tots els nombres son parells"<< endl;
```

Exemple 3: Recordatori esquema de cerca EOF

```
bool trobat = false;
while (not trobat and obtenir_element) {
    if (elem_compleix_cond) {
        trobat = true;
    }
}
if (trobat) {
    tractament_trobat
} else {
    tractament_no_trobat
}
```


Exemple 3: Solució

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

int main() {
  // Indica si tots els elements d'un flux d'enters positius són parells.
  bool trobat = false;
  int e;
  while (not trobat and cin >> e) {
    // Inv: els elements del flux tractats fins el moment són parells.
    if (e%2 != 0) trobat = true;
  }
  if (trobat) {
    cout << "No tots els nombres son parells" << endl;
  } else {
    cout << "Tots els nombres son parells" << endl;
  }
}
```

Exemple 4: Enunciat

Problema: Feu un programa que donat un flux d'enters acabat en 0 format per com a mínim dos elements, indiqui l'element del flux anterior al valor màxim del flux.

Exemples d'entrades i sortides esperades:

Entrada	Sortida
`1 -2 41 -23 31 13 7 0`	`31`
`10 -2 0`	`-2`

Exemple 4: Identificació flux

Problema: Feu un programa que donat un flux d'enters acabat en 0 format per com a mínim dos elements, indiqui l'element del flux anterior al valor màxim del flux.

Hi ha sentinella? S'indiquen el nombre d'elements del flux?

En l'enunciat s'indica que el flux acaba en 0, per tant el 0 és el sentinella.

Exemple 4: Recorregut o cerca?

Problema: Feu un programa que donat un flux d'enters acabat en 0 format per com a mínim dos elements, indiqui l'element del flux anterior al valor màxim del flux.

Quin esquema segueix? Recorregut o cerca?
És un recorregut ja que per trobar el màxim cal recórrer tots els elements del flux.

Per tant, cal identificar el tractament que s'ha de fer amb cada element del flux de dades.

Exemple 4: Tractament (I)

Problema: Feu un programa que donat un flux d'enters acabat en 0 format per com a mínim dos elements, indiqui l'element del flux anterior al valor màxim del flux.

Quin és el tractament?
Cal tenir dues variables: `mx` i `ant`, una on emmagatzemem l'element màxim i l'altra on guardem l'element anterior al màxim dels elements tractats del flux.

Exemple 4: Tractament (II)

Per cada element del flux cal comprovar si és major o no que el màxim guardat fins el moment, i en cas que sigui així actualitzar les dues variables.
`ant` s'actualitza amb el valor màxim actual.
`mx` s'actualitza amb el valor de l'element que estem tractant del flux.

```
if (e >= mx) {
   ant = mx;
   mx = e;
}
```


Exemple 4: Inicialitzacions

Problema: Feu un programa que donat un flux d'enters acabat en 0 format per com a mínim dos elements, indiqui l'element del flux anterior al valor màxim del flux.

Cal inicialitzar les variables *mx* i *ant*:

```
int e, mx, ant;
cin >> mx >> e;
if (e > mx) {
    ant = mx;
    mx = e;
}
else {
    ant = e;
}
```

Exemple 4: Tractament final

Problema: Feu un programa que donat un flux d'enters acabat en 0 format per com a mínim dos elements, indiqui l'element del flux anterior al valor màxim del flux.

Mostrar el resultat al final de tot.

```
cout << ant << endl;
```


Exemple 4: Recordatori esquema de recorregut amb sentinella

```
obtenir_primer_element
while ( not element_sentinella ) {
    tractar_element
    obtenir_seguent_element
}
[tractament_final]
```

El `tractament_final` és optatiu.

Exemple 4: Solució (I)

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

int main() {
  // Indica l'element anterior al màxim en un flux d'enters acabat en 0.
  int e, mx;
  cin >> mx >> e;
  int ant;
  if (e > mx) {
    ant = mx;
    mx = e;
  }
  else {
    ant = e;
  }
  ... // continua —>
```


Exemple 4: Solució (II)

```
  ...
  cin >> e;
  while (e != 0) {
    // Inv: ant conté l'element anterior al màxim dels elements tractats
    //      fins al moment.
    if (e >= mx) {
      ant = mx;
      mx = e;
    }
    cin >> e;
  }
  cout << ant << endl;
}
```

Índex

Exercicis de repàs

1. Feu un programa que donat un flux d'enters format per 100 enters indiqui el nombre enter més gran.
2. Feu un programa que donat un flux d'enters no buit acabat en 0 indiqui si hi ha algun nombre negatiu.
3. Feu un programa que donat un flux de reals no buit indiqui el nombre real més gran.
4. Feu un programa que donat un flux d'strings, calculi quantes vegades apareix repetida la primera paraula en el flux.

Reconeixement i llicència

Aquesta sessió ha utilitzat el següent material:

- **Seqüències genèriques**. Apunts d'Informàtica fets per na *Neus Català* i en *Mario Martín*.
- **Material docent de l'assignatura PRO1 de la FIB**. `http://www.cs.upc.edu/~pro1/` fet per en *Jordi Cortadella*, *Ricard Gavaldà* i *Fernando Orejas*.

Tema 11

Funcions i accions

Fonaments de Programació

Funcions i accions

Bernardino Casas

bcasas@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

- ✓ Conèixer els elements bàsics de les funcions i les accions.
- ✓ Saber transformar un programa a funció.
- ✓ Poder crear una funció i una acció des de zero.
- ✓ Conèixer quan cal crear una acció i quan una funció.

Índex

Índex

Índex

Què és una funció?

Definició 1: Funció

Una *funció* (anglès: function) és un grup d'instruccions que realitzen un càlcul determinat, i que tenen un nom que ens permet utilitzar-les.

- Els llenguatges de programació, com el C++, no només posen a la nostra disposició un conjunt bàsic d'operacions, sinó que també disposen d'una manera de definir les nostres pròpies operacions.
- Aquestes noves operacions són les *funcions*.

Avantatges de l'ús de funcions

L'ús apropiat de les funcions permet:

- **Incrementar la llegibilitat**. Els programes estan millor estructurats i són més fàcils de comprendre.
- Permet l'**ús de l'abstracció** en el disseny de programes.
- **Facilita la reutilització de codi** en diferents llocs del programa i en d'altres aplicacions.

Sintaxi i semàntica

- Sintaxis:

```
// Explicació del que fa la funció
// Pre: Precondició
// Post: Postcondició
tipus_retorn nom_funcio ( param1, param2, ... ) {
    instruccions
}
```

- Semàntica:
  - *tipus_retorn* és el tipus del valor que retorna la funció.
  - *nom_funcio* és el nom amb el qual s'identifica la funció.
  - *param1*, *param2*, ... són els paràmetres que ens permeten passar/intercanviar informació a la funció.

```
// Explicació ...
// Pre:  ...
// Post:  ...

tipus_retorn nom_funcio(paràmetres) {

    instruccions

    return ...;
}
```

Documentació

Capçalera

Cos de la funció


```
// Explicació ...
// Pre:  ...
// Post:  ...

tipus_retorn nom_funcio(paràmetres) {

    instruccions

    return ...;

}
```

Documentació

Capçalera

Cos de la funció

Documentació de la funció

- Per documentar el comportament d'una funció usarem els contractes Pre/Post i una explicació del que fa la funció.
- L'**explicació** indica el que fa la funció i està escrita en llenguatge natural.
- La **precondició** (Pre) és una condició que s'ha de satisfer abans de l'execució de la funció. Generalment farà referència als paràmetres i al conjunt de valors possibles que pot rebre la funció.
- La **postcondició** (Post) és una condició que sempre s'ha de complir després de l'execució de la funció. Generalment farà referència al que retorna la funció, és a dir, el conjunt de valors possibles que pot generar la funció..

Documentació en les funcions SEMPRE

Explicació Pre i Post SEMPRE

Per cadascuna de les funcions que creem nosaltres SEMPRE escriurem l'**explicació**, la **precondició** i la **postcondició**.

```
// Explicació ...
// Pre:  ...
// Post:  ...
```

Documentació

```
tipus_retorn nom_funcio(paràmetres) {
```

Capçalera

```
    instruccions

    return ...;

}
```

Cos de la funció

Tipus de retorn

- El primer que s'ha d'indicar en la capçalera d'una funció és el **tipus de retorn**, és a dir, el tipus del valor que torna la funció.
- La funció pot tornar:
  - un tipus bàsic: `int`, `double`, `bool`, `char`, ...
  - un tipus complex que hem inclòs en el nostre programa: vector, string, ...
  - un tipus creat per nosaltres.
  - res: `void`. Si no torna res ja no estaríem parlant de funció (veure acció).

Sols es pot retornar un valor

COMPTE!!

Una funció ha de retornar **UN ÚNIC** valor. En cas que no torni res o volguem tornar més d'un valor hem d'usar accions.

Nom de la funció

- El **nom de la funció** és com s'identifica a aquesta funció en tot el programa, i és el nom que posarem quan volguem usar-la.
- Per posar el nom a una funció es segueixen les mateixes regles que amb els altres identificadors.
- Cal "batejar" a la funció amb un nom adequat, que doni la informació de quina tasca realitza.
- El nom de la funció no necessàriament ha de ser únic. Pot haver dues funcions amb el mateix nom, però han de tenir tipus i/o nombre de paràmetres diferent.

Paràmetres

- Els **paràmetres** són el mecanisme que permet a una funció comunicar-se amb l'entorn (el programa principal o una altra funció) que l'ha invocat.
- Els paràmetres es declaren entre parèntesis després del nom de la funció, indicant primer el tipus i després el nom: tipus1 nom1, tipus2 nom2, ...
- Pot ser que una funció no tingui paràmetres, però així i tot després del nom han d'aparèixer **( )** sense res a dins.

Definició 2: Paràmetre formal

S'anomenen *paràmetres formals* els paràmetres que apareixen en la capçalera de la definició d'una funció.

Exemples de capçaleres

```
int suma2(int a, int b) { ... }

double mitjana(double a1, double a2, double a3) { ... }

bool es_capicua(int a) { ... }

double elevat(double base, int exponent) { ... }

bool es_primer(unsigned int x) { ... }
```

```
// Explicació ...
// Pre:  ...          Documentació
// Post:  ...

tipus_retorn nom_funcio(paràmetres) {  Capçalera

    instruccions
                       Cos de la funció
    return ...;

}
```


Cos de la funció

- El conjunt d'instruccions que estan associades a una funció conforma el Cos de la funció. Aquestes instruccions aniran entre **{** i **}**.
- Dins d'una funció podem escriure les instruccions que vulguem unides amb les estructures de control (seqüencial, condicional i/o repetitiva) i també fer crides a d'altres funcions.

Return

- El `return` és la instrucció que ens permet indicar quin és el valor que torna la funció.
- El `return` és l'ÚLTIMA instrucció d'una funció.
- Sintaxi:

```
return expressio;
```

- **IMPORTANT**: Per norma general, només hi haurà un únic `return` en les funcions. Recordeu que les funcions només poden tornar un únic valor.

Índex

Crida a una funció (I)

- El fet d'utilitzar una funció s'anomena col·loquialment *cridar* a la funció.
- **La crida a una funció altera l'execució normal d'un programa**. Es salta del punt en que s'està a executar el subprograma corresponent amb els paràmetres concrets.

Definició 3: Paràmetre actual

Els paràmetres concrets amb els que es crida una funció s'anomenen *paràmetres actuals* o *arguments*.

Crida a una funció (II)

- Sintaxis:

```
var = nom_funcio ( param1, param2, ... );
```

- Semàntica:
  - *var* és la variable on es recull el resultat de la funció.
  - *nom_funcio* és el nom amb el qual s'identifica una funció.
  - *param1, param2, ...* són els paràmetres que passen informació a la funció.

Funcionament de la funció

1. S'avalua cadascun dels paràmetres actuals de la funció.
2. Se salta a la capçalera de la funció corresponent. Cada paràmetre formal pren el valor que li correspon de l'avaluació del paràmetre actual.
3. S'executen les instruccions de la funció.
4. Es torna a la línia on hi havia la crida inicial de la funció amb el valor de retorn.
5. S'assigna el valor de retorn a la variable de recollida.
6. Es continua l'execució del programa.

Nombre i tipus de paràmetres ha de coincidir

COMPTE!!

El nombre de paràmetres que apareixen en la crida de la funció ha de coincidir en nombre i tipus amb els paràmetres formals de la capçalera de la funció.

Índex

Exemple 1: Suma

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

// Suma a i b.
// Pre: a i b són enters
// Post: retorna la suma d'a i b.
int suma(int a, int b) {
    int res;
    res = a + b;
    return res;
}

int main() {
    int z;
    z = suma(4 + 1, 3);
    cout << "El resultat de la suma es " << z;
}
```

Exemple 1: Suma

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

// Suma a i b.
// Pre: a i b són enters
// Post: retorna la suma d'a i b.
int suma(int a, int b) {
    int res;
    res = a + b;
    return res;
}

int main() {
    int z;      5    3
    z = suma(4 + 1, 3);
    cout << "El resultat de la suma es " << z;
}
```


Exemple 1: Suma

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

// Suma a i b.
// Pre: a i b són enters
// Post: retorna la suma d'a i b.
int suma(int a, int b) {
    int res;
    res = a + b;
    return res;
}

int main() {
    int z;      5    3
    z = suma(4 + 1, 3);
    cout << "El resultat de la suma es " << z;
}
```

Es poden cridar més d'un cop

Nombre de paràmetres

Una funció pot ser cridada diverses vegades en un programa, i els seus arguments no es limiten únicament a literals.

Exemple 2: Cridant la funció vàries vegades

```
[...]
// Suma a i b.
// Pre: a i b són enters
// Post: retorna la suma d'a i b.
int suma(int a, int b) {
    int res;
    res = a + b;
    return res;
}

int main() {
    int x, y;
    cin >> x >> y;
    int z1 = suma(x, y - 1);
    int z2 = suma(x*2, y - 2);
    int z3 = suma(x*3, y - 3);
    cout << z1 << ' ' << z2 << ' ' << z3 << endl;
}
```

Al canal d'entrada hi ha: 7 11

Exemple 3: Crida dins de crida

```
[...]
// Suma a i b.
// Pre: a i b són enters
// Post: retorna la suma d'a i b.
int suma(int a, int b) {
    int res;
    res = a + b;
    return res;
}

int main() {
    int x, y;
    cin >> x >> y;
    int z = suma(x, suma(x*2, y - 2));
    cout << z << endl;
}
```

Al canal d'entrada hi ha: 7 11

Índex

Índex

Què és una acció?

Definició 4: Acció

Una *acció* (anglès: procedure) és una funció però que no retorna res. El tipus de retorn és `void`.

- Si hem de retornar un valor usarem funcions i en qualsevol altre cas usarem *accions*.
- Si es vol tornar més d'un valor cal usar accions i pas de **paràmetres per referència** (veure més endavant pas de paràmetres).
- Atès que les accions no retornen res, les accions no tenen `return`.

Acció que mostra els factors d'un nombre

```
// Mostra els factors de n.
// Pre: n és un enter
// Post: Escriu per pantalla tots els factors de n.
void factors(int n) {
  int f = 2;
  int x = n;
  while (x != 1) {
    // Inv: Ha escrit els factors entre 2 i f
    if (x%f == 0) {
      cout << f << endl;
      x = x/f;
    }
    else ++f;
  }
}
```

⟵ **NO HI HA RETURN**

Índex

Tipus de paràmetres

- Una variable està associada amb un valor i amb una localització de memòria on està guardat aquest valor.
- En C++, hi ha dues formes per passar informació a una funció mitjançant els paràmetres:
  - **Pas per valor**. Aquesta és la manera normal de treballar. Això es fa simplement declarant el paràmetre de manera normal (amb tipus i nom).
  - **Pas per referència**. Es passa la localització de memòria. Això s'indica posant el símbol **&** entre el tipus i el nom del paràmetre.
- Per exemple:

```
void prova (double a, double &b)
```


Pas per valor

- El pas per valor fa un còpia de la informació que ens passen en la crida de la funció.
- És equivalent a tenir una assignació entre cada paràmetre actual de la crida i el paràmetre formal corresponent de la capçalera de la funció.
- Per exemple:

```
int x = 4;
s = suma(1, 3 + x);        // params actuals
```

implica que els paràmetres formals en la funció valdran:

```
int suma(int a, int b) {     // params formals
    // a = 1; b = 7
}
```

Pas per referència

- En el pas per referència el paràmetre actual (argument) i el paràmetre formal de la funció són el mateix.
- És a dir, el paràmetre formal és un alias del paràmetre actual (el paràmetre formal fa **referència** al mateix lloc de memòria de l'argument).

COMPTE!!

El paràmetre actual en un pas per referència SEMPRE ha de ser una **variable**. MAI pot ser una expressió.

Quan usar cada pas de paràmetre?

- Usar **pas per valor** quan els paràmetres no es modifiquin dins de la funció o acció.
- Usar **pas per referència** només en **accions** quan volguem:
  - tornar més d'un valor i per tant es vol que els canvis que es facin en el paràmetre formal modifiquin el paràmetre actual.
  - estalviar-nos de fer còpies d'objectes grans (encara que no es modifiqui el paràmetre formal).
- Habitualment usarem el *pas per valor* per tipus bàsics i el *pas per referència* per tipus complexos (que utilitzen molta memòria).

Índex

Exemple 1

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

void exemple(int a) {
  a *= 2;
  cout << a << endl;
}

int main() {
  int x = 11;
  exemple(x);
  cout << x << endl;
}
```

Exemple 2

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

void exemple(int &a) {
  a *= 2;
  cout << a << endl;
}

int main() {
  int x = 11;
  exemple(x);
  cout << x << endl;
}
```


Exemple 3

```
// Intercanvia els valors de x i y.
// Pre: x, y: enter
// Post: Intercanvia els valors de x i y.
void intercanvia(int &x, int &y) {   // params formals
    int z = x;
    x = y;
    y = z;
}

int main() {
    int a = 8, b = 11;
    intercanvia(a, b);   // params actuals
    cout << a << ' ' << b << endl;
    int x = 11, y = 121;
    intercanvia(x, y);   // params actuals
    cout << x << ' ' << y << endl;
}
```

Índex

Exercicis de repàs

1. Feu una funció que donat un nombre enter positiu calculi el seu factorial.
2. Feu una funció que calculi el màxim comú divisor de dos nombres enters.
3. Feu una funció que donat un nombre enter positiu indiqui si és un nombre quadrat perfecte. Un nombre n és quadrat perfecte si existeix un altre nombre que elevat al quadrat és igual a n.

Nota: No podeu usar `cmath` per resoldre cap dels exercicis.

Reconeixement i llicència

Aquesta sessió ha utilitzat el següent material:

- **Portal minidosis**. `http://www.minidosis.org/` fet per en *Pau Fernández*.
- **Material docent de l'assignatura PRO1 de la FIB**. `http://www.cs.upc.edu/~pro1/` fet per en *Jordi Cortadella*, *Ricard Gavaldà* i *Fernando Orejas*.

Tema 12

Visibilitat

Fonaments de Programació

Visibilitat

Bernardino Casas

bcasas@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

- ✓ Conèixer els diferents tipus de visibilitat de les variables: locals i globals.
- ✓ Donada una variable ja declarada saber on és visible.
- ✓ Ser conscient que estan completament prohibides l'ús de variables globals.

Índex

Índex

Índex

Què és la visibilitat d'una variable?

Definició 1: Visibilitat d'una variable

El concepte de *visibilitat* (anglès: name visibility) fa referència a si una variable es pot veure (utilitzar) en un cert lloc del nostre programa. Les variables només es poden utilitzar després de la seva creació i en el bloc (**{ }**) on han estat creades.

- Els blocs poden incloure altres blocs. Amb la qual cosa les variables de blocs externs són visibles en els blocs interns però no a la inversa.

Restriccions de visibilitat

- Dues variables que estiguin creades en el mateix bloc no poden tenir el mateix nom.
- És a dir, totes les variables creades en un bloc han de tenir un nom diferent.
- Una variable declarada en un bloc intern **emmascara** les variables que tinguin el mateix nom en blocs externs.

Índex

Exemple 1

```
int main() {
  int x;
  x = 0;

  double x;   // error: nom de variable ja usat en aquest bloc
  x = 1.0;
}
```


Exemple 2 (I)

```
int main() {
        // a??, b??
        int a = 1, b = 20;
        // a??, b??
        cout << a;                          // escriu ??
        {
                // c??, a??, b??
                cout << a + b;              // escriu ??
                int b = 3, c = 4;
                // b??, c??
                // b externa??
                cout << b;                  // escriu ??
                cout << c;                  // escriu ??
        }
        // c??
        cout << b;                          // escriu ??
}
```

Exemple 2 (II)

```
int main() {
        // a i b no són visibles
        int a = 1, b = 20;
        // a i b són visibles
        cout << a;                          // escriu 1
        {
                // c no és visible, a i b són visibles
                cout << a + b;              // escriu 21
                int b = 3, c = 4;
                // b i c són visibles,
                // però la b externa no és visible.
                cout << b;                  // escriu 3
                cout << c;                  // escriu 4
        }
        // c no és visible
        cout << b;                          // escriu 20
}
```


Índex

Variables locals vs globals

- Les variables les podem classificar com a **variables locals** i **variables globals**.
- Una variable local és una variable que està declarada dins d'un bloc i per tant només és visible *localment* dins del bloc.
- Una variable global és una variable declarada fora de qualsevol bloc (fora del main i de les funcions).
  Una variable global és visible des de tot arreu: tant des del programa principal com des de les funcions.

Exemple variable global

```
int foo;              // variable global

int una_funcio() {
    int bar;          // variable local
    bar = 0;
}

int una_altra_funcio() {
    foo = 1;   // ok: foo és global
    bar = 2;   // incorrecte: bar no és visible en aquesta funció
}

int main() {
   foo = 0;   // ok: foo és global
   ...
}
```

No es poden usar variables globals

COMPTE!!

Està completament prohibit usar variables globals en els vostres programes.

Índex

Exercicis de repàs

1. Indica que apareixerà pel canal de sortida en executar el següent programa:

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

int main() {
  int a = 1, b = 2;
  {
    cout << a << b << endl;
    int a = 10, b = 3;
    {
      int a = 100, b = 5;
      cout << a + b << endl;
    }
    cout << a << b << endl;
  }
  cout << a << b << endl;
}
```


Reconeixement i llicència

Aquesta sessió ha utilitzat el següent material:

- **C++ Language - C++ Tutorials**.
  `http://www.cplusplus.com/doc/tutorial/`.

Tema 13

Taules i vectors

Fonaments de Programació

Taules i vectors

Bernardino Casas

bcasas@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

- ✓ Conèixer el funcionament de les taules.
- ✓ Conèixer el funcionament dels vectors.
- ✓ Saber crear i modificar vectors.
- ✓ Saber afegir i eliminar caselles d'un vector.
- ✓ Saber com demanar el nombre d'elements que té un vector en un moment donat.

Índex

Índex

Índex

Què és una taula?

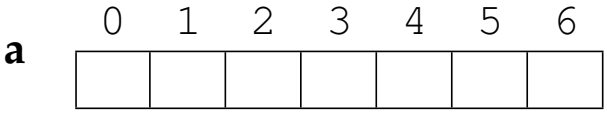
Definició 1: Taula

Una *taula* (anglès: array) és una estructura complexa que permet emmagatzemar una seqüència finita sota un mateix nom de variable. Els diferents elements de la seqüència es diferencien pel seu ordre (índex), la qual cosa permet l'accés diferenciat als elements.

- Una taula es pot veure com una agrupació de variables del mateix tipus, accessibles a través d'un índex.
- Els elements d'una taula també se'ls anomena **casella**.
- Quan es crea una taula indiquem quants elements tindrà, i aquest nombre no es pot canviar (no es poden afegir caselles a una taula).

Numeració de les caselles

- Una taula s'acostuma a representar gràficament de la següent manera:



- En aquest exemple la taula s'anomena **a** i té 7 caselles. Cadascuna d'aquestes caselles està numerada amb un nombre del `0` al `6`.

Primer índex de les caselles

COMPTE!!

Com en altres llenguatges de programació, i a diferència del que estem acostumats, l'índex del primer element d'una taula és el 0.

Declaració d'una taula

- Sintaxis:

```
tipus nom[num_elems];
```

- Semàntica:
  - *tipus* és el tipus de dades de totes les caselles de la taula.
  - *nom* és l'identificador amb el qual ens referim a la taula.
  - *num_elems* el nombre total de caselles de la taula.
- Exemples:

```
int a[7];
double taula[100];
```


Índex

Accés a les caselles

- Per accedir al contingut d'una casella d'una taula es posa el nom de la taula i a continuació entre corxets l'índex de la casella a la qual es vol accedir (per consultar o modificar el seu contingut).
- Atès que les caselles no deixen de ser variables, qualsevol cosa que es pot fer amb una variable es pot fer amb una casella d'una taula.
- Per exemple:

```
char t[3];
t[0] = 'a';
t[1] = 'b';
t[2] = t[1] + 1;
cout << t[0] << t[1] << t[2] << endl;
```


Variables com a índexos

- Els índexos que identifiquen les caselles no és necessari que siguin literals, poden ser variables enteres.
- Per exemple:

```
int t[3];
int i = 0;
t[i] = 1;
t[i + 1] = t[0] + 1;
t[t[0] + 1] = 10;
```

Índex fora de rang

COMPTE!!

Cal tenir present el nombre de caselles d'una taula per tal de no intentar accedir a una casella que no existeix. Accedir a una casella inexistent és un error greu ja que provoca que el programa acabi amb error.

Exemple taules

```
[...]
int main() {
  int t[100];

  int i = 0;
  while (i <= 100) {
    t[i] = i*2;
    ++i;
  }

  // Què contindrà la taula t en aquest punt?
}
```

Exemple taules

```
[...]
int main() {
  int t[100];

  int i = 0;
  while (i <= 100) {
    t[i] = i*2;
    ++i;
  }

  // Què contindrà la taula t en aquest punt?
}
```

Compte!! No arribaria al punt indicat ja que el programa acaba abans amb error. S'intenta accedir a una posició incorrecta de la taula.

Índex

Assignacions entre taules

- Si es vol fer una còpia d'una variable que és una taula NO es pot fer l'assignació normal. Per exemple: `a = b`
- Per copiar una taula cal recòrrer totes les caselles i copiar casella a casella.
- Exemple:

```
int a[100];
...
int b[100];
int i = 0;
while (i < 100) {
  b[i] = a[i];
  ++i;
}
```


Índex

Índex

Què és un vector?

- La classe vector forma parte de la biblioteca estàndar de C++, i en particular de la STL: la **Standard Template Library**.
- La paraula "*template*" significa plantilla en anglès, és a dir, la STL és una biblioteca de plantilles.
- Els vectors es comporten com les taules de C++, però són molt més potents:
  - A una taula **NO** es poden afegir ni eliminar caselles.
  - En canvi, a un vector sí que se li poden afegir o eliminar caselles en qualsevol moment (en funció de les nostres necessitats).

Include `vector`

Cal afegir #include

Sempre que volguem utilitzar vectors, haurem d'afegir `#include <vector>` al principi del programa.

Declaració d'un vector (I)

- Sintaxis:

```
vector<tipus> nom;
```

- Semàntica:
  - *tipus* és el tipus de dades de totes les caselles del vector.
  - *nom* és l'identificador amb el qual ens referim al vector.
- Exemples:

```
vector<int> a; // crea un vector d'enters buit de nom a
vector<char> b; // crea un vector de caràcters buit de nom b
```

Declaració d'un vector (II)

- Sintaxis:

```
vector<tipus> nom(num_elems);
```

- Semàntica:
  - *tipus* és el tipus de dades de totes les caselles del vector.
  - *nom* és l'identificador amb el qual ens referim al vector.
  - *num_elems* és el nombre inicial d'elements del vector.
- Exemples:

```
vector<int> a(7);  // crea un vector d'enters de nom a amb
                   // inicialment 7 caselles.
vector<char> b(100); // crea un vector de caràcters de nom b
                     // amb inicialment 100 caselles.
```


Declaració d'un vector (III)

- Sintaxis:

```
vector<tipus> nom(num_elems, val);
```

- Semàntica:
  - *tipus* és el tipus de dades de totes les caselles del vector.
  - *nom* és l'identificador amb el qual ens referim al vector.
  - *num_elems* és el nombre inicial d'elements del vector.
  - *val* és el valor que inicialment tindrà cada casella. Cal que sigui del tipus de dades *tipus*.
- Exemples:

```
vector<int> a(7, 0);  // crea un vector d'enters de nom a amb
                      // 7 caselles amb valor 0.
vector<char> b(100, 'a'); // crea un vector de caràcters de nom
                          // b amb 100 caselles amb valor 'a'.
```

Índex

Accés: consulta i modificació

- Encara que la declaració entre una taula i un vector és ben diferent, a l'hora d'accedir al contingut de les caselles d'un vector es fa de la mateixa manera com amb una taula.
- Exemple:

```
vector<char> v(3);
v[0] = 'a';
v[1] = 'b';
v[2] = v[1] + 1;
cout << v[0] << v[1] << v[2] << endl;
```

Afegir i eliminar elements a un vector

- Per afegir noves caselles al final de vector utilitzem el mètode push_back(*nou_element*) on *nou_element* és el nou element que es vol afegir.
- Per eliminar l'última casella del vector es fa amb el mètode pop_back().

```
vector<int> a;        // mida a és 0

a.push_back(3);
a.push_back(5);
a.push_back(8);
// a = [3, 5, 8]; mida a és 3

a.pop_back();
// a = [3, 5]; mida a és 2
```


Nombre d'elements d'un vector

- Atès que un vector té una mida variable, en un instant de temps ens pot interessar saber el nombre d'elements que té exactament el vector.
- Això es pot saber mitjançant el mètode size().
- Cal tenir present que la mida del vector és un **unsigned int** (enter sense signe o natural).
- Exemple:

```
vector<double> v(50);
unsigned int m = v.size();
cout << m << endl; // escriuria 50
```

Índex

Assignacions entre vectors

- Com a exemple de la potència dels vectors, copiar dos vectors es pot fer amb l'assignació: `a = b`
- A diferència de les taules l'assignació **SÍ** que funciona correctament entre vectors (ja que aquesta operació està definida en la classe vector).
- Cal pensar que l'assignació entre vectors és una operació costosa i s'hauria d'intentar d'evitar en la mesura del possible.
- Per aquest motiu en cas que es passi un vector com a paràmetre a una funció es passarà per **referència constant**: `const tipus &nom` o per **referència** per evitar les còpies.

Taules vs vectors

COMPTE!!

Donada la major versatilitat i potència dels vectors respecte les taules, en els exercicis que haguem d'utilitzar taules en el seu lloc farem servir **vector** de STL.

Índex

Exercicis de repàs

1. Feu un programa que creï un vector amb els nombres enters del 1 al 10.
2. Feu un programa que creï un vector amb les lletres minúscules de l'alfabet anglès ordenades de més petit a més gran.

Reconeixement i llicència

Aquesta sessió ha utilitzat el següent material:

- **Portal minidosis**. `http://www.minidosis.org/` fet per en *Pau Fernández*.
- **Material docent de l'assignatura PRO1 de la FIB**. `http://www.cs.upc.edu/~pro1/` fet per en *Jordi Cortadella*, *Ricard Gavaldà* i *Fernando Orejas*.

Tema 14

Cadenes de caràcters

Fonaments de Programació

Cadenes de caràcters

Bernardino Casas

bcasas@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

- ✓ Saber crear i utilitzar strings per mostrar missatges pel canal de sortida.
- ✓ Saber modificar strings.
- ✓ Saber consultar una part d'un string.
- ✓ Ser capaç de llegir un string format per diverses paraules des del canal d'entrada.

Índex

Índex

Cadenes de caràcters (I)

Definició 1: Cadena de caràcters

Una *cadena de caràcters* (anglès: string) és una successió de caràcters (lletres, nombres o altres signes o símbols) que permet emmagatzemar un text.

- La biblioteca estàndard de C++ ofereix la classe `string` que ens permet gestionar les cadenes de caràcter amb tota comoditat.
- Els string es poden tractar com a una taula de caràcters amb certes operacions addicionals.
- La classe `string` ofereix les operacions habituals de construcció, assignació, etc. i podem llegir-los o imprimir-los d'igual forma que els tipus elementals.

Cadenes de caràcters (II)

- Si no es posen restriccions respecte l'alfabet, una cadena podrà estar formada per qualsevol combinació finita dels caràcters disponibles (les lletres de la 'a' a la 'z' i de la 'A' a la 'Z', els nombres del '0' al '9', l'espai en blanc ' ', símbols diversos '!', '@', '%', etc.)
- Els literals de les cadenes de caràcters sempre estan definits entre cometes dobles.
- Per exemple:

```
"El gat menja peix"
"Hello world!!"
```

Declaració d'strings

Cal afegir #include

Sempre que volguem utilitzar string's, haurem d'afegir `#include` <string> al principi del programa. Això no és necessari si usem la biblioteca <iostream> ja que aquesta ja fa un include d'string.

Exemples d'strings (I)

```
string s1 = "";
string s2 = "abc";
string s3(10, 'x');

cout << s1 << s2 << s3;

cout << s2 << "123" << s2;

string s4;
cout << "Escriu el teu nom: ";
cin >> s4;
```

Exemples d'strings (II)

```
string s1 = "";   // s1 = cadena buida
string s2 = "abc";   // s2 = "abc"
string s3(10, 'x'); // s3 = "xxxxxxxxxx"

cout << s1 << s2 << s3; // Escriu abcxxxxxxxxxx

cout << s2 << "123" << s2; // Escriu abc123abc

string s4;
cout << "Escriu el teu nom: ";
cin >> s4;
```


Índex

Índex

Accedir a un caràcter de l'string

- Es pot accedir els caràcters d'un string escrivint entre corxets l'índex del caràcter que es vol consultar.
- Igual que en les taules i els vectors l'índex del primer caràcter d'una cadena de caràcters és el 0.
- Exemple:

```
string s = "Hola";
cout << s[0];
// Escriu H
```

Mida d'un string

- Per obtenir la mida d'un string cal usar la funció `length()`.
- Com es fa amb els vectors també es pot obtenir la mida amb la funció `size()`.
- Exemple:

```
string txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
cout << "La mida de l'string txt es: " << txt.length();
cout << "La mida de l'string txt es: " << txt.size();
```


Índex

Canviar el valor d'un caràcter

- Canviar un caràcter d'un string es fa a través de l'índex del caràcter que es vol canviar.
- Es posa el nom de la variable string seguit d'aquest índex entre corxets i se li assigna un nou caràcter.
- Exemple:

```
string s = "Hola";
s[0] = 'J';
cout << s;
// Escriu Jola en comptes de Hola
```


Concatenació

- Els operadors + i += serveixen per concatenar.

```
string s, t ; // crea dos strings buits
s = "Hola" ;
t = s + "␣mon!" ; // concatena a s l'string "mon!"
cout << t << endl ; // imprimeix Hola mon!
t += "␣i␣adeu!";
cout << t << endl ; // imprimeix Hola mon! i adeu!
```

- També es pot concatenar un caràcter a un string.

```
string t("cola") ;
t = t + '*';  // t = "cola*"
```

Índex

Llegint strings

- Es pot usar l'operador >> en el canal d'entrada per demanar una cadena de caràcters a l'usuari.
- Exemple:

```
string nom;
cout << "Escriu el teu nom: ";
cin >> nom;
cout << "El teu nom es: " << nom;

// Escriu el teu nom: Joan
// El teu nom es: Joan
```

Problemes amb els espais

- IMPORTANT: L'operador >> considera els espais (espai en blanc, tabulacions, etc.) com un caràcter final. Això significa que cada lectura només llegirà una sola paraula encara que l'usuari n'hagi escrit moltes.
- Exemple:

```
string nomComplet;
cout << "Escriu els teus noms i cognoms: ";
cin >> nomComplet;
cout << "El teu nom i cognoms es: " << nomComplet;

// Escriu els teus noms i cognoms: John Doe
// El teu nom i cognoms es: John
```

- A l'exemple anterior hom podria esperar que el programa mostrés *John Doe*, però només mostra *John*.

`getline()`

- Hi ha d'altres operacions que permeten llegir de manera diferent. Per exemple, si es vol llegir tota una línia de text es pot usar la funció `getline()`.
- `getline` rep el canal d'on s'ha de llegir la informació i la variable on es vol guardar la informació.
- Exemple:

```
string nomComplet;
cout << "Escriu els teus noms i cognoms: ";
getline(cin, nomComplet);
cout << "El teu nom i cognoms es: " << nomComplet;

// Escriu els teus noms i cognoms: John Doe
// El teu nom i cognoms es: John Doe
```

Índex

Exercicis de repàs

1. Feu una funció que donat un string que conté una paraula (que està formada per lletres minúscules de l'alfabet anglès) retorni la versió PigLatin d'aquesta paraula. Per convertir una paraula a PigLatin només cal seguir aquestes regles:
   1. Si la paraula comença amb una vocal, la versió PigLatin afegeix a la paraula el sufix "way".
   2. Si la paraula comença amb una consonant, la versió PigLatin transfereix totes les consonants fins arribar a la primera vocal al final i després afegeix el sufix "ay".
   3. La lletra 'y' es tractarà com a consonant si és la primera lletra d'una paraula, i vocal en cas contrari.

   Exemples:

   flower $\longrightarrow$ owerflay

   yellow $\longrightarrow$ ellowyay

   apple $\longrightarrow$ appleway

Reconeixement i llicència

Aquesta sessió ha utilitzat el següent material:

- **Portal minidosis**. `http://www.minidosis.org/` fet per en *Pau Fernández*.
- **Material docent de l'assignatura PRO1 de la FIB**. `http://www.cs.upc.edu/~pro1/` fet per en *Jordi Cortadella*, *Ricard Gavaldà* i *Fernando Orejas*.

Tema 15

Bucle For

Fonaments de Programació

Bucle for

Bernardino Casas

bcasas@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

- ✓ Conèixer el funcionament de l'estructura iterativa **for**.
- ✓ Saber transformar un **while** a un **for**.
- ✓ Saber transformar un **for** a un **while**.

Índex

Índex

Índex

while vs for

- El **for** és un forma alternativa de *while* on tota la informació sobre la iteració queda resumida en una línia.
- Per exemple donat el següent bucle while:

```
i = 1;
while ( i <= 100 ) {
  cout << i << endl;
  ++i;
}
```

es pot transformar en el següent bucle for:

```
for ( i = 1; i <= 100; ++i ) {
  cout << i << endl;
}
```

Ús del bucle for

- Sintaxis:

```
for ( inicial; condicio_cont; increment ) {
    instruccions
}
```

- Semàntica:
  1. S'executa un sol cop la instrucció *inicial*.
  2. S'avalua la *condicio_cont*. Si avalua és falsa sortim del bucle. Si és certa continuem en el bucle.
  3. S'executen les instruccions dins del cos del bucle.
  4. S'executa l'*increment* i es torna al pas **2**.

Quan usar el bucle for

COMPTE!!

L'estructura iterativa **for** NOMÉS es pot usar quan es coneix el nombre d'iteracions abans del seu ús.

Per exemple, alguns casos on es coneix el nombre d'iteracions abans del seu ús són els següents:

- l'usuari entra el nombre d'elements a tractar
- aquest nombre es pot calcular a priori

.

Índex

Exemple: Enunciat

Problema: Feu un programa que donat un nombre enter positiu *n*, mostri la suma dels primers *n* nombres senars.

Exemple: Solució usant for

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

// Calcula la suma dels n primers nombres senars.
int main() {
  int n, s = 0;
  cin >> n;
  for (int i = 0; i < n; ++i) {
    // Inv: s conté la suma dels i primers nombres senars.
    s = s + i*2 + 1;
  }
  cout << s << endl;
}
```


Exemple: Solució usant while

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

// Calcula la suma dels n primers nombres senars.
int main() {
  int n, s = 0;
  cin >> n;
  int i = 0;
  while (i < n) {
    // Inv: s conté la suma dels i primers nombres senars.
    s = s + i*2 + 1;
    ++i;
  }
  cout << s << endl;
}
```

Índex

Exercicis de repàs

1. Transforma el següent programa per tal que enlloc d'un bucle while hi hagi un bucle for:

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

int main() {
  int a = 10;
  while (a <= 100) {
    cout << "a val: " << a << endl;
    a += 10;
  }
}
```

2. Feu un programa que mostri un compte enrere d'un en un des del número indicat per l'usuari fins el 0.

Reconeixement i llicència

Aquesta sessió ha utilitzat el següent material:

- **Portal minidosis**. `http://www.minidosis.org/` fet per en *Pau Fernández*.

Tema 16

Esquemes iteratius sobre vectors

Fonaments de Programació

Esquemes iteratius sobre vectors

Bernardino Casas

bcasas@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

- ✓ Saber identificar els elements bàsics d'un problema amb vectors: primer, següent i últim element.
- ✓ Conèixer els esquemes iteratius de recorregut i cerca sobre vectors.
- ✓ Saber reconèixer quin esquema iteratiu caldria aplicar per resoldre un problema de vectors.
- ✓ Saber aplicar la metodologia de disseny d'algorismes iteratius per resoldre problemes de vectors.

Índex

Índex

Índex

Recordatori

L'esquema de recorregut s'aplica en els problemes on s'han de tractar tots els elements d'un vector.

```
obtenir_primer_element
while ( not ultim_element ) {
    tractar_element
    obtenir_seguent_element
}
tractar_ultim_element
[tractament_final]
```

Adaptació de l'esquema de recorregut (I)

- Observeu que pels vectors habitualment farem recorreguts començant pel primer element del vector fins arribar a l'últim. Per tant:
  - El primer element del vector a tractar és aquell que té índex 0.
  - El darrer element és el que té la mida del vector menys 1 (size() - 1).
  - Obtenir el següent element del vector a tractar és obtenir l'element amb l'índex incrementat en una unitat.
- Tenint en compte aquestes consideracions es pot refinar l'esquema general de recorregut.

Adaptació de l'esquema de recorregut (II)

COMPTE!!

Si aquestes consideracions sobre com recórrer el vector no s'adeqüen al problema que estem resolent, no es podrà aplicar l'esquema de recorregut adaptat per vectors. Si es vol visitar les caselles del vector en un altre ordre cal canviar el primer/últim/següent de l'esquema.

Esquema de recorregut per vectors

L'esquema de recorregut per vectors queda de la següent manera.

```
unsigned int i = 0, mida = v.size();
while ( i < mida ) {
    tractar_element v[i]
    ++i;
}
[tractament_final]
```

El tractament de l'últim element s'ha inclòs dins el bucle.

Esquema de recorregut per vectors amb for

L'esquema de recorregut també pot implementar amb el bucle for.

```
unsigned int mida = v.size();
for(unsigned int i = 0; i < mida; ++i) {
    tractar_element v[i]
}
[tractament_final]
```

El tractament de l'últim element s'ha inclòs dins el bucle.

Índex

Exemple 1: Enunciat

1. Feu una funció que donat un vector de reals, calculi la mitjana dels elements del vector.

Exemple 1: Enunciat

1. Feu una funció que donat un vector de reals, calculi la mitjana dels elements del vector.

Hem de recórrer **tot** el vector ja que necessitem sumar tots els elements que conté el vector. Per tant es tracta d'un **recorregut**.

Exemple 1: Tractament

- L'ordre en que hem de recórrer el vector és l'estàndar amb la qual cosa podem aplicar l'esquema de recorregut per vectors.
- En el tractament del recorregut anirem calculant la suma, per tant necessitem una variable on acumular la suma.
- D'aquesta manera en cada iteració afegirem a aquesta variable el valor de l'element que estiguem tractant:

```
s += v[i];
```

Exemple 1: Solució

```
// Calcula la mitjana dels elements del vector.
// Pre: v és un vector de reals
// Post: retorna la mitjana del elements del vector v.
double mitjana(const vector<double> &v) {
  unsigned int mida = v.size(), s = 0;
  for (unsigned int i = 0; i < mida; ++i) {
    // Inv: s conté la suma dels elements del subvector v[0..i-1]
    s += v[i];
  }
  double mitj = 0.0;
  if (mida > 0) {      // COMPTE!! el vector no pot estar buit per
    mitj = double(s)/mida;      // calcular la mitjana.
  }
  return mitj;
}
```


Exemple 2: Enunciat

2. Feu una funció que donat un vector d'enters, calculi la suma dels elements del vector que estan en posicions parells.

Exemple 2: Enunciat

2. Feu una funció que donat un vector d'enters, calculi la suma dels elements del vector que estan en posicions parells.

Hem de recórrer el vector ja que necessitem sumar els elements del vector que estan en posicions parells. Per tant es tracta d'un **recorregut**.

Exemple 2: Tractament

- L'ordre en que hem de recórrer el vector és l'estàndar amb la qual cosa podem aplicar l'esquema de recorregut per vectors.
- En el tractament del recorregut anirem calculant la suma per tant necessitem una variable on anar acumulant la suma.
- D'aquesta manera en cada iteració en cas que la posició sigui parell afegirem a aquesta variable el valor de l'element que estiguem tractant:

```
if (i%2 == 0) {
   s += v[i];
}
```

Exemple 2: Solució

```
// Calcula la suma dels elements del vector que estan en posicions parells.
// Pre: v és un vector d'enters
// Post: retorna la suma del elements del vector v que estan en posicions
//    parells.
double suma_pos_parells(const vector<int> &v) {
  unsigned int mida = v.size(), s = 0;
  for (unsigned int i = 0; i < mida; ++i) {
    // Inv: s conté la suma dels elements del subvector v[0..i-1] que estan
    //    en posicions parells.
    if (i%2 == 0) {
      s += v[i];
    }
  }
  return s;
}
```


Exemple 2: Millorant la solució

- Una possible millora seria visitar només el elements necessaris. És a dir, en comptes que el següent del vector sigui `++i` hauria de ser `i += 2`.
- Amb aquest canvi visitaríem la meitat dels elements del vector (i faríem la meitat d'iteracions). El tractament seria el següent:

```
s += v[i];
```

Exemple 2: Solució

```
// Calcula la suma dels elements del vector que estan en posicions parells.
// Pre: v és un vector d'enters
// Post: retorna la suma del elements del vector v que estan en posicions
//    parells.
double suma_pos_parells(const vector<int> &v) {
  unsigned int mida = v.size(), s = 0;
  for (unsigned int i = 0; i < mida; i += 2) {
    // Inv: s conté la suma dels elements del subvector v[0..i-1] que estan
    //    en posicions parells.
    s += v[i];
  }
  return s;
}
```


Índex

Índex

Recordatori

L'esquema de cerca s'aplica en els problemes on s'ha de determinar si existeix un element en un vector que compleixi una propietat donada.

```
bool trobat = false;
obtenir_primer_element
while (not trobat and not ultim_element) {
    if (elem_compleix_cond) trobat = true;
    else {
        obtenir_seguent_element
    }
}
if (trobat) tractament_trobat
else tractament_no_trobat
```

Aplicació de l'esquema de cerca

- Observeu que pels vectors habitualment farem recorreguts començant pel primer element del vector fins arribar a l'últim. Per tant:
  - El primer element a tractar del vector habitualment és aquell que té índex 0.
  - El darrer element és el que té la mida del vector menys 1 (size() - 1).
  - Obtenir el següent element a tractar del vector habitualment és obtenir l'element amb l'índex incrementat en una unitat.
- Tenint en compte aquestes consideracions es pot refinar l'esquema general de cerca.

Recordatori

L'esquema de cerca refinat per vectors queda de la següent forma:

```
bool trobat = false;
unsigned int i = 0, mida = v.size();
while (not trobat and i < mida) {
   if (elem_compleix_cond) trobat = true;
   else ++i;
}
if (trobat) tractament_trobat
else tractament_no_trobat
```

No usar for en les cerques

COMPTE!!

Per resoldre un problema de cerca **NO** es pot fer servir l'estructura iterativa **for**.

Índex

Exemple 3: Enunciat

3. Feu una funció que donat un vector de naturals, indiqui si tots els elements del vector són parells.

Esquema de recorregut
Esquema de cerca
Exercicis de repàs
Esquema
Exemples

Exemple 3: Enunciat

3. Feu una funció que donat un vector de naturals, indiqui si tots els elements del vector són parells.

Hem de buscar si tots els elements del vector són parells. En concret, hem de buscar si algun element del vector és senar. Per tant es tracta d'una **cerca**.

Exemple 3: Condició cerca

- L'ordre en que hem de recórrer el vector és l'estàndar amb la qual cosa podem aplicar l'esquema de cerca per vectors.
- Estem buscant si algun element del vector és senar. Per tant la condició de cerca seria la següent:

```
v[i]%2 != 0
```


Exemple 3: Solució

```
// Indica si tots els elements del vector són parells.
// Pre: v és un vector de naturals
// Post: retorna true si tots els elements del vector v són parells;
//   false en cas contrari.
bool tots_parells(const vector<int> &v) {
  bool trobat = false;
  unsigned int  i = 0, mida = v.size();
  while (not trobat and i < mida) {
    // Inv: tots els elements del subvector v[0..i-1] són parells.
    if (v[i]%2 != 0) trobat = true;
    else ++i;
  }
  return not trobat;
}
```

Exemple 4: Enunciat

4. Feu una funció que donat un vector de naturals amb com a mínim un element, indiqui si hi ha algun element del vector (llevat del primer element) que té el dígit de les unitats igual que el dígit de les unitats del primer element del vector.

Esquema de recorregut
Esquema de cerca
Exercicis de repàs
Esquema
Exemples

Exemple 4: Enunciat

4. Feu una funció que donat un vector de naturals amb com a mínim un element, indiqui si hi ha algun element del vector (llevat del primer element) que té el dígit de les unitats igual que el dígit de les unitats del primer element del vector.

Hem de buscar si el dígit de les unitats d'algun element del vector coincideix amb el dígit de les unitats del primer element del vector. Per tant es tracta d'una **cerca**.

Exemple 4: Condició cerca

- L'ordre en que hem de recórrer el vector és l'estàndar llevat que el primer element a tractar no és el que té índex 0 sinó amb índex 1. Per tant podem aplicar l'esquema de cerca per vectors canviant el primer element del vector a visitar.
- Per obtenir el dígit de les unitats d'un nombre només cal aplicar el mòdul 10 (`%10`) sobre aquest número. Tenint en compte això la condició de cerca seria la següent:

```
v[i]%10 == v[0]%10;
```


Exemple 4: Solució

```
// Indica si el dígit de les unitats d'algun dels elements del vector és igual al
// dígit de les unitats del primer element del vector.
// Pre: v és un vector de naturals no buit
// Post: retorna true si el dígit de les unitats d'algun dels elements de v és
//    igual al dígit de les unitats del primer element de v; false en cas
//    contrari.
bool igual_unitats0(const vector<int> &v) {
  bool trobat = false;
  unsigned int  i = 1, mida = v.size();
  while (not trobat and i < mida) {
    // Inv: cap dels elements del subvector v[1..i-1] té igual el dígit de les
    //    unitats que el dígit de les unitats de l'element en la posició 0.
    if (v[i]%10 == v[0]%10) trobat = true;
    else ++i;
  }
  return trobat;
}
```

Índex

Exercicis de repàs

1. Feu un programa que donat un nombre enter n, creï un vector i guardi en aquest vector n enters llegits de teclat.
2. Feu una funció que donat un vector de caràcters indiqui si el vector és capicua.
3. Feu un programa que llegeixi des de teclat dos vectors de reals de la mateixa mida $v1$ i $v2$ (que corresponen a una sèrie de dades i els seus pesos) i calculi i mostri la mitjana ponderada. La fòrmula de la mitjana ponderada és la següent:

$$\tilde{x} = \frac{\sum_{i=0}^{n-1} v1[i] \cdot v2[i]}{\sum_{i=0}^{n-1} v2[i]}$$

Reconeixement i llicència

Aquesta sessió ha utilitzat el següent material:

- **Llistes**. Apunts d'Informàtica fet per na *Neus Català* i en *Mario Martín*.
- **Portal minidosis**. `http://www.minidosis.org/` fet per en *Pau Fernández*.

Tema 17

Matrius

Fonaments de Programació

Matrius

Bernardino Casas

bcasas@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

- ✓ Conèixer com crear matrius estàtiques i dinàmiques.
- ✓ Saber accedir al valor d'una casella en concret d'una matriu.
- ✓ Poder afegir i eliminar files i columnes en matrius dinàmiques.
- ✓ Saber com demanar el nombre de files i columnes que té una matriu en un moment donat.

Índex

Índex

Què és una matriu?

Definició 1: Matriu

Una *matriu* es una estructura de dades que emmagatzema de manera contínua elements del mateix tipus. Les matrius com a mínim tenen dues dimensions.

- Una matriu es pot considerar com un vector de dues dimensions, és a dir, un vector de vectors.
- Totes les caselles de la matriu són del mateix tipus de dades.
- Gràficament una matriu la representarem:

5	8	1	0
4	3	0	7
2	6	9	3

- Igual com passava amb les taules, les matrius es poden declarar de dues maneres diferents: **estàticament** o **dinàmicament**.
- La diferència principal entre les matrius dinàmiques i les estàtiques és que:
  - Es pot modificar el nombre de files i columnes de les *matrius dinàmiques*.
  - Les *matrius estàtiques* sempre tenen la mida amb la que es van crear.
- Per crear matrius dinàmiques farem servir la classe `vector`.
- L'ús d'un tipus o un altre de matriu dependrà del problema (de si cal afegir/esborrar files i/o columnes a la matriu).

Índex

Declaració d'una matriu estàtica

- Sintaxis:

```
tipus nom[files][cols];
```

- Semàntica:
  - *tipus* indica el tipus de dades de les caselles de la matriu.
  - *nom* és l'identificador amb el qual ens referim a la matriu.
  - *files* és el nombre de files de la matriu.
  - *cols* és el nombre de columnes de la matriu.
- Exemples:

```
int mat1[3][4];
double mat2[7][15];
```

Índex

Declaració d'una matriu dinàmica (I)

- Sintaxis:

```
vector< vector<tipus> > nom;
```

- Semàntica:
  - *tipus* indica el tipus de dades de les caselles de la matriu.
  - *nom* és l'identificador amb el qual ens referim a la matriu.
- Exemples:

```
vector< vector<int> > mat1; // mat1 no té mida
vector< vector<double> > mat2; // mat2 no té mida
```

Declaració d'una matriu dinàmica (II)

- Sintaxis:

```
vector< vector<tipus> > nom
            (files, vector<tipus>(cols));
```

- Semàntica:
  - *tipus* indica el tipus de dades de les caselles de la matriu.
  - *nom* és l'identificador amb el qual ens referim a la matriu.
  - *files* és el nombre de files de la matriu.
  - *cols* és el nombre de columnes de la matriu.
- Exemples:

```
vector< vector<int> > mat1(3, vector<int>(4));
vector< vector<double> > mat2(7, vector<double>(15));
```


Declaració d'una matriu dinàmica (III)

- Sintaxis:

```
vector< vector<tipus> > nom
     (files, vector<tipus>(cols, valor));
```

- Semàntica:
  - *tipus* indica el tipus de dades de les caselles de la matriu.
  - *nom* és l'identificador amb el qual ens referim a la matriu.
  - *files* és el nombre de files de la matriu.
  - *cols* és el nombre de columnes de la matriu.
  - *valor* és el valor inicial del contingut de les caselles de la matriu.
- Exemples:

```
vector< vector<char> > mat(3, vector<char>(4, ' '));
```

Declaració del tipus matriu dinàmica (IV)

- Sintaxis:

```
typedef vector<tipus> Fila;
typedef vector<Fila> Matriu;
Matriu nom(files, Fila(cols));

// o directament
typedef vector< vector<tipus> > Matriu;
Matriu nom(files, vector<tipus>(cols));
```

- Semàntica:
  - *tipus* indica el tipus de dades de les caselles de la matriu.
  - *nom* és l'identificador amb el qual ens referim a la matriu.
  - *files* és el nombre de files de la matriu.
  - *cols* és el nombre de columnes de la matriu.

Exemple de declaració amb el tipus matriu

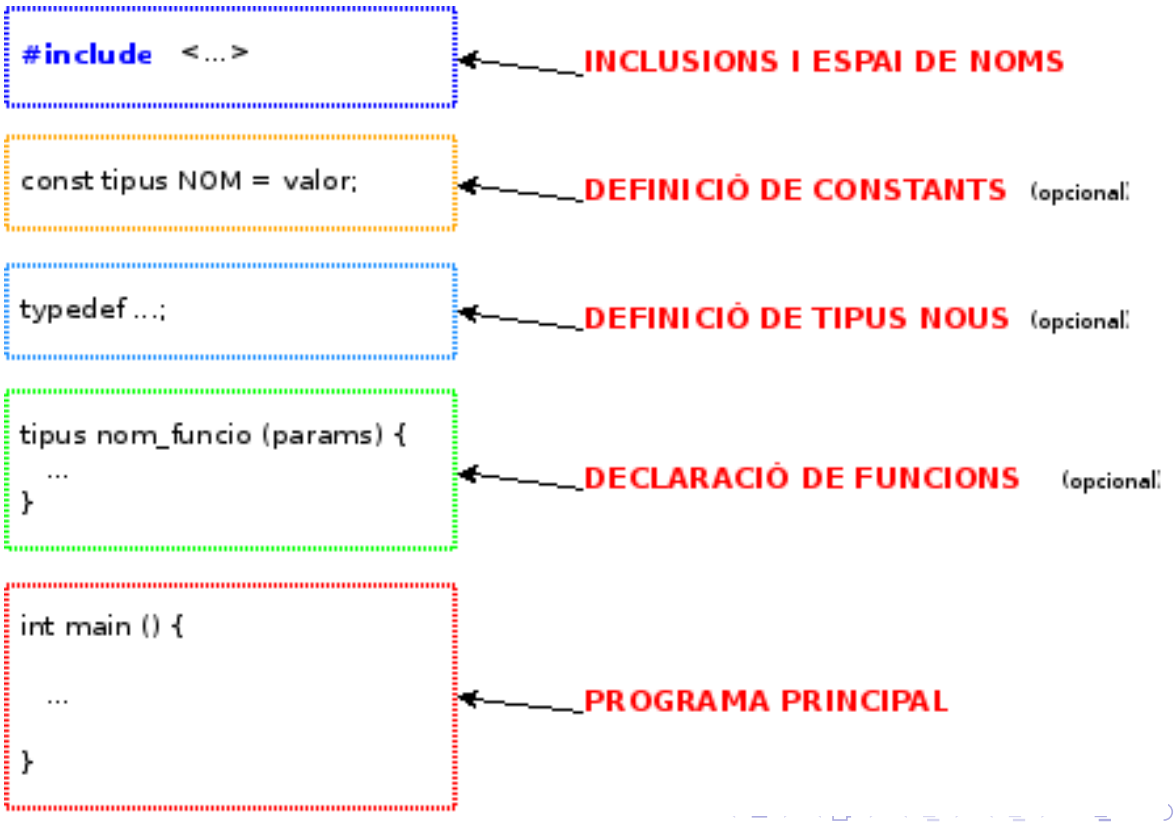
- Exemple 1:

```
typedef vector<char> Fila;
typedef vector<Fila> Matriu;
Matriu mc(6, Fila(5));
```

- Exemple 2:

```
typedef vector< vector<bool> > Matriu;
Matriu mb(9, vector<bool>(3));
```

Recordatori: Estructura d'un programa



Índex

Índex

Accés: consulta/modificació

- Accedir al contingut d'una casella d'una matriu funciona de manera similar com amb els vectors però en comptes d'un índex cal usar-ne dos:

```
nom_matriu[index fila][index columna]
```

- Si una matriu té n files, els números de fila vàlids estan a l'interval $0...n-1$, de manera que la primera fila és la 0.
- Si una matriu té m columnes, els números de columna vàlids estan a l'interval $0...m-1$, de manera que la primera columna és la 0.

Cal indicar sempre fila i columna

- El compilador no pot comprovar que el valor d'un índex estigui en el rang corresponent (ja que la mida de les matrius pot variar durant l'execució del programa).
- Només comprova que hi hagi dos índexos i que cada índex sigui un enter.

COMPTE!!

Per accedir a una casella d'una matriu bidimensional SEMPRE cal indicar l'índex de les files i l'índex de les columnes (cada índex entre corxets).

Exemple 1

```
typedef vector<char> Fila;
typedef vector<char> Matriu;

Matriu ma(4, Fila(5, '␣'));
ma[1][2] = 'A';
ma[3][4] = 'T';
ma[6][2] = 'A';
ma[1][12] = 'A';
```

Exemple 1: Solució

```
typedef vector<char> Fila;
typedef vector<char> Matriu;    // error en el tipus de la Fila
typedef vector<Fila> Matriu;

Matriu ma(4, Fila(5, ' '));
ma[1][2] = 'A';
ma[3][4] = 'T';
ma[6][2] = 'A';   // error l'índex de les files va de 0..3
ma[1][12] = 'A';   // error l'índex de les columnes va de 0..4
```

	0	1	2	3	4
0	' '	' '	' '	' '	' '
1	' '	' '	'A'	' '	' '
2	' '	' '	' '	' '	' '
3	' '	' '	' '	' '	'T'

Índex

Afegir i eliminar files a una matriu dinàmica

- Per afegir o esborrar una fila a una **matriu dinàmica** s'ha de fer servir el mètode push_back() o pop_back() respectivament.
- Exemple:

```
typedef vector<int> Fila;
typedef vector<Fila> Matriu;
Matriu m(11, Fila(7, 0));

// afegeix una fila al final de la matriu on cada casella d'aquesta
// nova fila val -1
m.push_back(Fila(7, -1));
...
m.pop_back(); // esborra l'ultima fila de la matriu
```


Afegir i eliminar columnes a una matriu dinàmica

COMPTE!!

En les matrius dinàmiques es poden afegir i eliminar files i columnes. Però, per simplicitat un cop creada la matriu en aquest curs NO s'afegiran ni s'esborraran columnes.

Exemple 2

```
typedef vector<int> Fila;
typedef vector<Fila> Matriu;

Matriu ma(2, Fila(3, 0));
ma[1][2] = 1;
ma[1][2] += 3;
ma[ma[0][2]][1] = ma[1][2];
ma.push_back(Fila(3, 1));
ma[2][2] = 2;
```


Definició i declaració
Operacions bàsiques amb matrius
Exercicis de repàs
Consulta/modificació de caselles de la matriu
Afegir/eliminar files i columnes
Nombre de files i columnes

Exemple 2

```
typedef vector<int> Fila;
typedef vector<Fila> Matriu;

Matriu ma(2, Fila(3, 0));
ma[1][2] = 1;
ma[1][2] += 3;
ma[ma[0][2]][1] = ma[1][2];
ma.push_back(Fila(3, 1));
ma[2][2] = 2;
```

	0	1	2
0	0	4	0
1	0	0	4
2	1	1	2

Índex

Nombre de files i columnes d'una matriu dinàmica

- Per saber el nombre de files i columnes de la matriu dinàmica s'ha de fer servir el mètode `size()` (que té la classe vector).
- Exemple:

```
Matriu m(11, Fila(7));

// Escriu el nombre de files
cout << m.size() << endl;

// Escriu el nombre de columnes
cout << m[0].size() << endl;
```

Índex

Exercicis de repàs (I)

1. Fes una funció que donada una matriu d'enters retorni el resultat de sumar el nombre de files més el nombre de columnes de la matriu.
2. Fes una funció que donades tres matrius de reals indiqui quina matriu té més caselles. La funció ha de retornar 1, 2 o 3 depenent de quina sigui la matriu amb més caselles. En cas que hi hagi empat en el nombre més gran de caselles ha de retornar el nombre més baix.

Exercicis de repàs (II)

3. Fes un programa que creï una matriu de la mida i els valors que es veu en aquesta representació:

X		O	O
		X	O
O	O		
O	O	X	X

   Fes dues solucions, una usant una matriu estàtica i una altra usant una matriu dinàmica.
   No facis servir cap bucle per omplir la matriu.

4. Fes un programa que creï una matriu d'enters de 11x20 i ompli la matriu per files amb nombres negatius consecutius començant pel -1.

Reconeixement i llicència

Aquesta sessió ha utilitzat el següent material:

- **Matrius**. Apunts d'Informàtica fet per na *Neus Català* i en *Mario Martín*.
- **Portal minidosis**. `http://www.minidosis.org/` fet per en *Pau Fernández*.

Tema 18

Esquemes iteratius sobre matrius

Fonaments de Programació

Esquemes iteratius sobre matrius

Bernardino Casas

bcasas@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

- ✓ Saber identificar els elements bàsics d'un problema amb matrius: primer, següent i últim element.
- ✓ Conèixer els esquemes iteratius de recorregut i cerca sobre matrius.
- ✓ Saber reconèixer quin esquema iteratiu cal aplicar per resoldre un problema de matrius.
- ✓ Saber aplicar la metodologia de disseny d'algorismes iteratius per resoldre problemes de matrius.

Índex

Recordatori esquemes

- A l'hora de dissenyar algorismes que treballin amb matrius també farem servir esquemes.
- L'**esquema de recorregut** en matrius es fa servir quan volem tractar tots els elements de la matriu.
- L'**esquema de cerca** en matrius es fa servir per buscar un element que compleixi una determinada condició dins la matriu.

Índex

Esquema de recorregut
Esquema de cerca
Exercicis de repàs
Esquema
Exemples

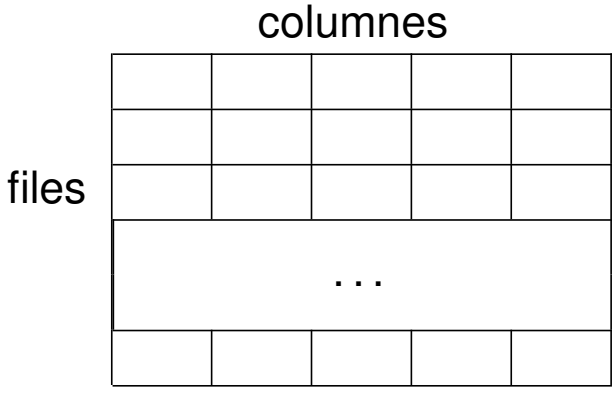
Índex

Esquema algorísmic de recorregut

```
unsigned int numf = mat.size(), numc = mat[0].size();
for (unsigned int i = 0; i < numf; ++i) {
  for (unsigned int j = 0; j < numc; ++j) {
    tractar mat[i][j]
  }
}
tractament_final
```

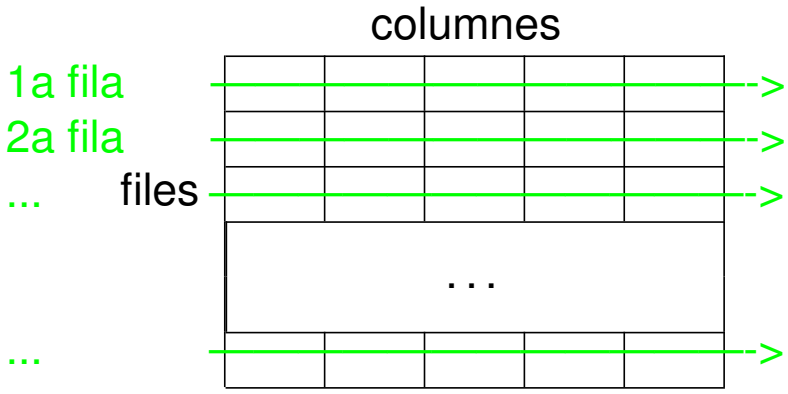

Quin ordre segueix l'esquema?

- L'esquema anterior recorre la matriu per files. Primer recorre totes les columnes de la primera fila, després recorre les columnes de la segona fila, etc.
- Gràficament ho representaríem:



Quin ordre segueix l'esquema?

- L'esquema anterior recorre la matriu per files. Primer recorre totes les columnes de la primera fila, després recorre les columnes de la segona fila, etc.
- Gràficament ho representaríem:



L'ordre de visita dels elements

COMPTE!!

Si es vol visitar les caselles de la matriu en un altre ordre cal canviar el primer/últim/següent fila i/o columna de l'esquema.

Índex

Exemples de recorregut

1. Feu una acció que donada una matriu de reals, l'ompli amb valors que s'obtenen pel teclat.
2. Feu una funció que donada una matriu d'enters, retorni la suma de tots els elements parells de la matriu.
3. Feu una funció que donada una matriu d'enters, retorni la suma de tots els elements de la matriu que estan a caselles on la seva fila i columna sigui parell.

Exemple 1: Enunciat

1. Feu una acció que donada una matriu de reals, l'ompli amb valors que s'obtenen pel teclat.

Esquema de recorregut
Esquema de cerca
Exercicis de repàs
Esquema
Exemples

Exemple 1: Enunciat

1. Feu una acció que donada una matriu de reals, l'ompli amb valors que s'obtenen pel teclat.

Hem de recórrer **tota** la matriu i a cada casella de la matriu guardar un real llegit del canal d'entrada.

Exemple 1: Solució

```
typedef vector< vector<double> > Matriu;

// Omple la matriu que es passa per paràmetre amb valors reals llegits
// del canal d'entrada.
// Pre: mat és una matriu de reals.
// Post: omple mat amb valors reals llegits de teclat.
void omple_matriu(Matriu &mat) {
  unsigned int numf = mat.size(), numc = mat[0].size();
  for (unsigned int i = 0; i < numf; ++i) {
    // Inv: totes les caselles amb índex de fila menor a i estan omplertes
    //    amb reals llegits de teclat.
    for (unsigned int j = 0; j < numc; ++j) {
      // Inv: totes les caselles amb índex de fila menor que i i les caselles
      //    amb índex de fila igual a i i índex de columna menor que j estan
      //    omplertes amb reals llegits de teclat.
      cin >> mat[i][j];
    }
  }
}
```


Exemple 2: Enunciat

2. Feu una funció que donada una matriu d'enters, retorni la suma de tots els elements parells de la matriu.

Exemple 2: Enunciat

2. Feu una funció que donada una matriu d'enters, retorni la suma de tots els elements parells de la matriu.

Hem de recórrer **tota** la matriu i suma només els elements parells. Per tant necessitem una variable on acumular la suma de cada casella que tingui un valor parell.

Exemple 2: Solució

```
typedef vector< vector<int> > Matriu;

// Retorna la suma de tots els elements parells de la matriu.
// Pre: mat és una matriu d'enters.
// Post: retorna la suma de les caselles de mat amb valors parells.
int suma_parells(const Matriu &mat) {
  int s = 0;
  unsigned int numf = mat.size(), numc = mat[0].size();
  for (unsigned int i = 0; i < numf; ++i) {
    // Inv: s acumula la suma de totes les caselles amb valor parell
    //    amb índex de fila menor a i.
    for (unsigned int j = 0; j < numc; ++j) {
      // Inv: s acumula la suma de totes les caselles amb valor parell
      //    amb índex de fila menor a i i també les caselles
      //    amb índex de fila igual a i i índex de columna menor que j.
      if (mat[i][j] % 2 == 0) s += mat[i][j];
    }
  }
  return s;
}
```

Exemple 3: Enunciat

3. Feu una funció que donada una matriu d'enters, retorni la suma de tots els elements de la matriu que estan a caselles on la seva fila i columna sigui parell.

Esquema de recorregut
Esquema de cerca
Exercicis de repàs
Esquema
Exemples

Exemple 3: Enunciat

3. Feu una funció que donada una matriu d'enters, retorni la suma de tots els elements de la matriu que estan a caselles on la seva fila i columna sigui parell.

Hi ha dues maneres d'abordar el problema:

(a) recórrer **tota** la matriu i descartar aquelles caselles els índexs de les quals no siguin parells.

(b) recórrer només les caselles amb índex de fila i columna parell.

Exemple 3: Enunciat

3. Feu una funció que donada una matriu d'enters, retorni la suma de tots els elements de la matriu que estan a caselles on la seva fila i columna sigui parell.

Hi ha dues maneres d'abordar el problema:

(a) recórrer **tota** la matriu i descartar aquelles caselles els índexs de les quals no siguin parells.

(b) recórrer només les caselles amb índex de fila i columna parell.

Quina és la millor estratègia?

Esquema de recorregut
Esquema de cerca
Exercicis de repàs
Esquema
Exemples

Exemple 3: Enunciat

3. Feu una funció que donada una matriu d'enters, retorni la suma de tots els elements de la matriu que estan a caselles on la seva fila i columna sigui parell.

Hi ha dues maneres d'abordar el problema:

(a) recórrer **tota** la matriu i descartar aquelles caselles els índexs de les quals no siguin parells.

(b) recórrer només les caselles amb índex de fila i columna parell.

Quina és la millor estratègia? **(b)**
A més a més, en qualsevol d'aquestes estratègies necessitem una variable on acumular la suma.

Exemple 3: Solució

```
typedef vector< vector<int> > Matriu;

// Calcula la suma de tots els elements de la matriu que estan a caselles on l'índex
// de la seva fila i columna és parell.
// Pre: mat és una matriu d'enters.
// Post: retorna la suma de les caselles de mat amb índexos parells.
int suma_indexos_parells(const Matriu &mat) {
  int s = 0;
  unsigned int numf = mat.size(), numc = mat[0].size();
  for (unsigned int i = 0; i < numf; i += 2) {
    // Inv: s acumula la suma de totes les caselles amb índexos parells i la
    //    fila menor a i.
    for (unsigned int j = 0; j < numc; j += 2) {
      // Inv: s acumula la suma de totes les caselles amb índexos parells
      //    i la fila menor a i i també les caselles amb índex parell de fila
      //    igual a i i índex parell de columna menor que j.
      s += mat[i][j];
    }
  }
  return s;
}
```


Índex

Índex

Esquema algorísmic de cerca

```
bool trobat = false;
unsigned int i = 0;
unsigned int numf = mat.size(), numc = mat[0].size();
while (not trobat and i < numf) {
  unsigned int j = 0;
  while (not trobat and j < numc) {
    if (compleix_condicio_mat[i][j]) trobat = true;
    else ++j;
  }
  if (not trobat) ++i;
}
if (trobat) tractament_trobat;
else tractament_no_trobat;
```

L'ordre de visita dels elements

COMPTE!!

L'esquema de cerca segueix el mateix ordre de visita que l'esquema de recorregut. Com passa amb l'esquema de recorregut si es vol canviar aquest ordre de visita cal canviar el primer/següent/últim de la fila i/o columna.

Índex

Exemples de cerca

4. Feu una funció que donada una matriu d'enters, determini si té un nombre múltiple de 5.
5. Feu una funció que donada una matriu quadrada de caràcters, indiqui si la matriu és simètrica.
6. Feu una funció que donada una matriu de caràcters, indiqui si alguna fila de la matriu té més de 3 espais en blanc.

Exemple 4: Enunciat

4. Feu una funció que donada una matriu d'enters, determini si té un nombre múltiple de 5.

Exemple 4: Enunciat

4. Feu una funció que donada una matriu d'enters, determini si té un nombre múltiple de 5.

Hem de buscar si la matriu conté algun element múltiple de 5.

Exemple 4: Solució

```
[...]
// Indica si la matriu conté un nombre múltiple de 5.
// Pre: mat és una matriu d'enters.
// Post: retorna true si mat té un nombre múltiple de 5; false en cas contrari.
bool multiple5(const Matriu &mat) {
  bool trobat = false;
  unsigned int i = 0, numf = mat.size(), numc = mat[0].size();
  while (not trobat and i < numf) {
    // Inv: cap de les caselles amb índex de fila menor que i té un múltiple de 5.
    unsigned int j = 0;
    while (not trobat and j < numc) {
      // Inv: cap de les caselles amb índex de fila menor que i i les caselles amb índex
      //    de fila igual a i i índex de columna menor que j té un múltiple de 5.
      if (mat[i][j]%5 == 0) trobat = true;
      else ++j;
    }
    if (not trobat) ++i;
  }
  return trobat;
}
```

Exemple 5: Enunciat

5. Feu una funció que donada una matriu quadrada de caràcters, indiqui si la matriu és simètrica.

Esquema de recorregut
Esquema de cerca
Exercicis de repàs
Esquema
Exemples

Exemple 5: Enunciat

5. Feu una funció que donada una matriu quadrada de caràcters, indiqui si la matriu és simètrica.

Hem de buscar (al contrari del que podria semblar) que la matriu sigui no simètrica. Si trobem alguna casella de la matriu no simètrica llavors no cal mirar més ja que la matriu ja no pot ser simètrica.

Atès que cada casella es comprova amb el seu simètric (aquella que té els índexos intercanviats) només cal recórrer els elements per sobre o per sota de la diagonal principal.

Exemple 5: Solució

```
[...]
// Indica si la matriu és simètrica.
// Pre: mat és una matriu quadrada de caràcters.
// Post: retorna true si mat és simètrica; false en cas contrari.
bool simetrica(const Matriu &mat) {
  bool trobat = false;
  unsigned int i = 0, numf = mat.size(), numc = mat[0].size();
  while (not trobat and i < numf - 1) {
    // Inv: totes les caselles amb índex de fila menor que i són simètriques.
    unsigned int j = i + 1;
    while (not trobat and j < numc) {
      // Inv: totes les caselles amb índex de fila menor que i i les caselles amb índex
      //    de fila igual a i i índex de columna menor que j són simètriques.
      if (mat[i][j] != mat[j][i]) trobat = true;
      else ++j;
    }
    if (not trobat) ++i;
  }
  return not trobat;
}
```


Exemple 6: Enunciat

6. Feu una funció que donada una matriu de caràcters, indiqui si alguna fila de la matriu té més de 3 espais en blanc.

Exemple 6: Enunciat

6. Feu una funció que donada una matriu de caràcters, indiqui si alguna fila de la matriu té més de 3 espais en blanc.

Hem de buscar si alguna fila conté més de tres espais en blanc. Això implica que hem de comptar el nombre d'espais en blanc en cada fila i comprovar si hi ha alguna fila que en tingui més de 3.

Exemple 6: Solució

```
[...]
// Indica si alguna fila de la matriu té més de 3 espais en blanc.
// Pre: mat és una matriu de caràcters.
// Post: retorna true si alguna fila de mat té més de 3 espais en blanc; false en cas
//    contrari.
bool tres_espais(const Matriu &mat) {
  bool trobat = false;
  unsigned int i = 0, numf = mat.size(), numc = mat[0].size();
  while (not trobat and i < numf) {
    // Inv: cap de files menors que i conté més de tres caselles amb espais en blanc.
    unsigned int j = 0, espais = 0;
    while (not trobat and j < numc) {
      // Inv: [...]
      if (mat[i][j] == ' ') ++espais;
      if (espais > 3) trobat = true;
      else ++j;
    }
    if (not trobat) ++i;
  }
  return trobat;
}
```

Índex

Exercicis de repàs

1. Feu una funció que donada una matriu, retorni un vector amb la suma de cadascuna de les columnes de la matriu.
2. Feu una funció que donada una matriu d'enters, determini si és la matriu identitat (té 1 a les posicions de la diagonal i 0 a la resta).
3. Feu un programa que donat un enter k i una matriu d'enters positius, determini si la suma dels nombres parells que conté la matriu és superior a k.
4. Feu una funció que donades dues matrius de reals, torni la suma matemàtica de les mateixes.

Reconeixement i llicència

Aquesta sessió ha utilitzat el següent material:

- **Portal minidosis**. `http://www.minidosis.org/` fet per en *Pau Fernández*.

Tema 19

Recursivitat

Fonaments de Programació

Recursivitat

Bernardino Casas

bcasas@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

- ✓ Saber interpretar un algorisme recursiu, és a dir, poder escriure el resultat d'aplicar un algorisme recursiu.
- ✓ Saber identificar els casos directe/s i els casos recursius d'un problema recursiu.
- ✓ Saber dissenyar algorismes recursius en C++ a través de la seva identificació de casos.

Índex

Índex

Índex

Exemple 1: Què apareixerà per pantalla?

```
1  void proc(int a) {
2    int res = a + 1;
3    cout << a << endl;
4  }
5
6  int main() {
7    proc(3);
8    proc(11);
9  }
```

Exemple 2: Què apareixerà per pantalla?

```
1  void proc(int a) {
2    int res = a + 1;
3    cout << a << endl;
4    proc(a+1);
5  }
6
7  int main() {
8    proc(3);
9    proc(11);
10 }
```


Què és un algorisme recursiu?

Definició 1: Algorisme recursiu

Un *algorisme recursiu* és aquell que fa ús de la recursivitat. En la pràctica un subprograma (acció o funció) és recursiu quan dins del seu codi es crida a sí mateix (incorpora una crida a ell mateix).

- La recursivitat pot substituir un bucle en el disseny d'un programa. Alguns algorismes recursius es poden reescriure com algorismes iteratius.
- Generalment les solucions recursives són més simples que (o tan simples com) les solucions iteratives.

Altres consideracions

- Habitualment, les solucions recursives són lleugerament menys eficients que les solucions iteratives (si el compilador no intenta optimitzar les crides recursives).
- Hi ha solucions recursives que surten de manera molt natural que poden ser extremadament ineficients.

COMPTE!!

Un algorisme recursiu li cal una condició de parada per a no entrar en un bucle infinit.

Índex

Passos del disseny recursiu

El disseny d'un algorisme recursiu segueix els següents passos:

1. Identificar el **cas bàsic** (també anomenat *cas directe*) i determinar com s'ha de resoldre. El cas bàsic és aquell cas en el qual el programa es pot resoldre directament sense necessitat de fer una crida recursiva. De casos bàsics pot haver-ne més d'un.
2. Determinar com es resol el **cas recursiu** en termes del casos bàsics (els quals ja hem resolt en el pas anterior). De casos recursius pot haver-ne més d'un.

Compte amb les crides infinites

COMPTE!!

Cal fixar-se en que els paràmetres de la crida recursiva es van apropant de mica en mica a algun dels casos bàsics. Això garanteix que hi ha una seqüència finita de crides recursives (no entrem en una crida infinita).

Exercici: Què apareixerà per pantalla?

```
1  int func(int a, int b) {
2    int res;
3    cout << "Abans " << a << " " << b << endl;
4    if (a > b) res = a;
5    else res = a + func(2*a, b);
6    cout << "Despres " << a << ' ' << b << ' ' << res << endl;
7    return res;
8    cout << "Fi " << res << endl;
9  }
10
11 int main() {
12   cout << func(3, 15) << endl;
13 }
```


Solució: Què apareixerà per pantalla?

```
Abans 3 15
Abans 6 15
Abans 12 15
Abans 24 15
Despres 24 15 24
Despres 12 15 36
Despres 6 15 42
Despres 3 15 45
45
```

La recursivitat per dins

- Cada cop que una funció es crida, es crea una nova instància de la funció. Cada cop que una funció acaba (fem el return), aquesta instància també s'acaba.
- La creació d'una nova instància només requereix la reserva de memòria per les dades (els paràmetres i les variables locals).
- Les instàncies d'una funció es destrueixen en l'ordre invers a la seva creació. La primera instància a ser creada serà l'última a ser destruïda.

Índex

Índex

Enunciat

Exercici:
Fes una funció que donat un nombre enter positiu retorni el factorial d'aquest nombre.

Solució iterativa

```
// Calcula el factorial del nombre donat de forma iterativa.
// Pre: n és un nombre enter positiu
// Post: retorna n!.
int factorial(int n) {
  int f = 1, i = 0;
  while (i < n) {
    // Inv: f = i!
    i = i + 1;
    f = f*i;
  }
  return f;
}
```


Definició de factorial

- Definició de factorial:

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots 2 \cdot 1$$

- Definició recursiva:

$$n! = \begin{cases} n \cdot (n-1)!, & si\ n > 0 \\ 1, & si\ n = 0 \end{cases}$$

Disseny de l'algorisme recursiu

1. **Cas bàsic**. L'únic cas bàsic segons la definició anterior és aquell en que $n = 0$. En aquest cas la funció ha de retornar 1.
2. **Cas recursiu**. Sabem que el factorial d'un nombre n més gran que 0 és igual a:

$$n * factorial(n - 1)$$

Nota: Assegurem que no entrem en crides infinitives ja que $n - 1$ està més aprop de 0 que n. Per tant, es pot assegurar que la funció acaba.

Solució recursiva

```
// Calcula el factorial del nombre donat de manera recursiva.
// Pre: n és un nombre enter positiu
// Post: retorna n!.
int factorial(int n) {
  int res;
  if (n == 0) res = 1;
  else res = n*factorial(n - 1);
  return res;
}
```

Índex

Enunciat

Exercici:
Fes una acció que donat un nombre enter positiu escrigui per pantalla la seva representació binària.

Disseny de l'algorisme recursiu

1. **Cas bàsic**. Tenim dos casos bàsics:
   - $n = 0$ llavors escrivim 0.
   - $n = 1$ llavors escrivim 1.
2. **Cas recursiu**. Cal escriure la representació binària de $n/2$ i a continuació escriure $n\%2$.

Nota: Assegurem que no entrem en crides infinitives ja que $n/2$ està més aprop de 0 i 1 que n. Per tant, es pot assegurar que la funció acaba.

Solució recursiva

```
// Escriu pel canal de sortida la representació binària del nombre donat.
// Pre: n és un nombre enter positiu
// Post: escriu la representació binària de n.
void base2(int n) {
  int res;
  if (n == 0) res = 0;
  else if (n == 1) res = 1;
  else {
    base2(n/2);
    res = n%2;
  }
  cout << res;
}
```

Índex

Enunciat

Exercici:
Fes una funció que donat un nombre enter positiu n retorni el nombre de Fibonacci d'ordre n.
Un nombre de fibonacci d'ordre n és igual a la suma dels dos nombres de fibonacci anteriors, exceptuant per $n = 0$ i $n = 1$ que és 1.
Per exemple:

n	0	1	2	3	4	5	6	7	8	9	...
fib	1	1	2	3	5	8	13	21	34	55	...

Disseny de l'algorisme recursiu

1. **Cas bàsic**. Tenim dos casos bàsics:
   - $n = 0$ llavors el resultat és 1.
   - $n = 1$ llavors el resultat és 1.
2. **Cas recursiu**. Sabem que el nombre fibonacci d'un nombre n és:

$$fibonacci(n-1) + fibonacci(n-2)$$

Nota: Assegurem que no entrem en crides infinitives ja que $n-1$ i $n-2$ està més aprop de 0 i 1 que n. Per tant, es pot assegurar que la funció acaba.

Solució recursiva

```
// Calcula el nombre de fibonacci de l'ordre indicat.
// Pre: n és un nombre enter positiu
// Post: retorna el nombre de fibonacci d'ordre n.
unsigned int fib(unsigned int n) {
  unsigned int res;
  if (n == 0 or n == 1) res = 1;
  else res =  fib(n - 1) + fib(n - 2);
  return res;
}
```

COMPTE!! La implementació recursiva (aquesta versió) és força més costosa que la iterativa.

Índex

Exercicis de repàs

1. Feu una funció recursiva que donats un enter x i un natural n, calculi x^n.
2. Feu una funció que donat un nombre enter positiu retorni l'*arrel digital* d'aquest nombre. L'arrel digital d'un nombre s'obté sumant tots els dígits d'aquest nombre, i aplicant el mateix procés per cada resultat fins que el resultat tingui un sol dígit. Per exemple, l'arrel digital de 7423 és 7 ja que $7 + 4 + 2 + 3 = 16$ i $1 + 6 = 7$.
3. Feu una acció recursiva que donat un vector de reals mostri per pantalla el vector en ordre invers (des de l'últim fins al primer element del vector).

Reconeixement i llicència

Aquesta sessió ha utilitzat el següent material:

- **Material docent de l'assignatura PRO1 de la FIB**. `http://www.cs.upc.edu/~pro1/` fet per en *Jordi Cortadella*, *Ricard Gavaldà* i *Fernando Orejas*.

Tema 20

Tuples i disseny de l'estructura de dades

Fonaments de Programació

Tuples i disseny de l'estructura de dades

Bernardino Casas

bcasas@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

- ✓ Saber com declarar i usar una tupla.
- ✓ Saber usar les tuples per retornar més d'un valor en una funció.
- ✓ Saber dissenyar estructures de dades per algorismes més complexos.

Índex

Índex

Índex

Què és una tupla?

Definició 1: Tupla

Una *tupla* (`struct` en C++) és un conjunt d'elements agrupats sota un mateix nom. Aquests elements (també anomenats *camps*) poden ser de diferents tipus i de diferents mides.

- Tots els elements d'una taula o d'un vector han de ser del mateix tipus, i a més, cadascun d'aquests elements s'identifica amb un número.
- En canvi, els elements d'una tupla poden ser de tipus diferent, i a més, cada element de la tupla s'identifica amb un nom.

Declaració de tuples

- Sintaxis:

```
struct nom_tupla {
  tipus1 nom1;
  tipus2 nom2;
  ...
};    // recordeu-vos de posar ; !!!
```

- Semàntica:
  - *nom_tupla* és l'identificador amb el qual ens referim a aquesta tupla.
  - *tipus1, tipus2, ...* indiquen els tipus de dades de cadascun dels camps de la tupla.
  - *nom1, nom2, ...* són els identificadors amb els que ens referim a cadascun dels camps de la tupla.

El nom dels camps d'una tupla

COMPTE!!

No pot haver dos o més camps en una tupla amb el mateix nom. Sinó no podríem identificar de manera inequívoca cadascun dels camps d'una tupla.

Exemple de tuples

```
struct Adreca {
  string tipus, nom;
  int numero;
};

struct Persona {
  string nom, cognoms;
  unsigned int any_naix;
  string telefon, email;
  Adreca adreca;    // la tupla Adreca ha d'estar declarada abans
};
```

L'ordre en que es declaren les tuples importa!!

Índex

Accedir a un camp d'una tupla

- Per accedir a un camp d'una tupla cal posar el nom de la **variable** de tipus tupla, l'operador de selecció (un punt) i el nom del camp al que es vol accedir:

```
nom_variable . nom_camp;
```

- Per exemple:

```
Persona p;
p.any_naix = 1980;
p.nom = "John";
p.cognoms = "Doe";
cout << p.nom << ' ' << p.cognoms << endl;
```


Copiar tuples

- Per copiar els valors d'una variable tupla a una altra variable del mateix tipus es pot fer de dues maneres: copiar camp a camp o bé usar l'assignació.
- Per exemple:

```
struct Article {
  string nom;
  double preu;
};

Article a = {"poma", 1.00}, b;
b.nom = a.nom;
b.preu = a.preu;
```

El problema és que si la tupla té molts camps, això ens obliga a copiar cadascun per separat. Millor:

```
b = a;
```

Tuples per retornar més d'un valor

- Les tuples es poden usar en les funcions per retornar més d'un valor.
- Per exemple:

```
struct Resultat {
  int lloc;
  bool trobat;
};

Resultat cercar(int x, const vector<int> &v);
```


Índex

Invariants

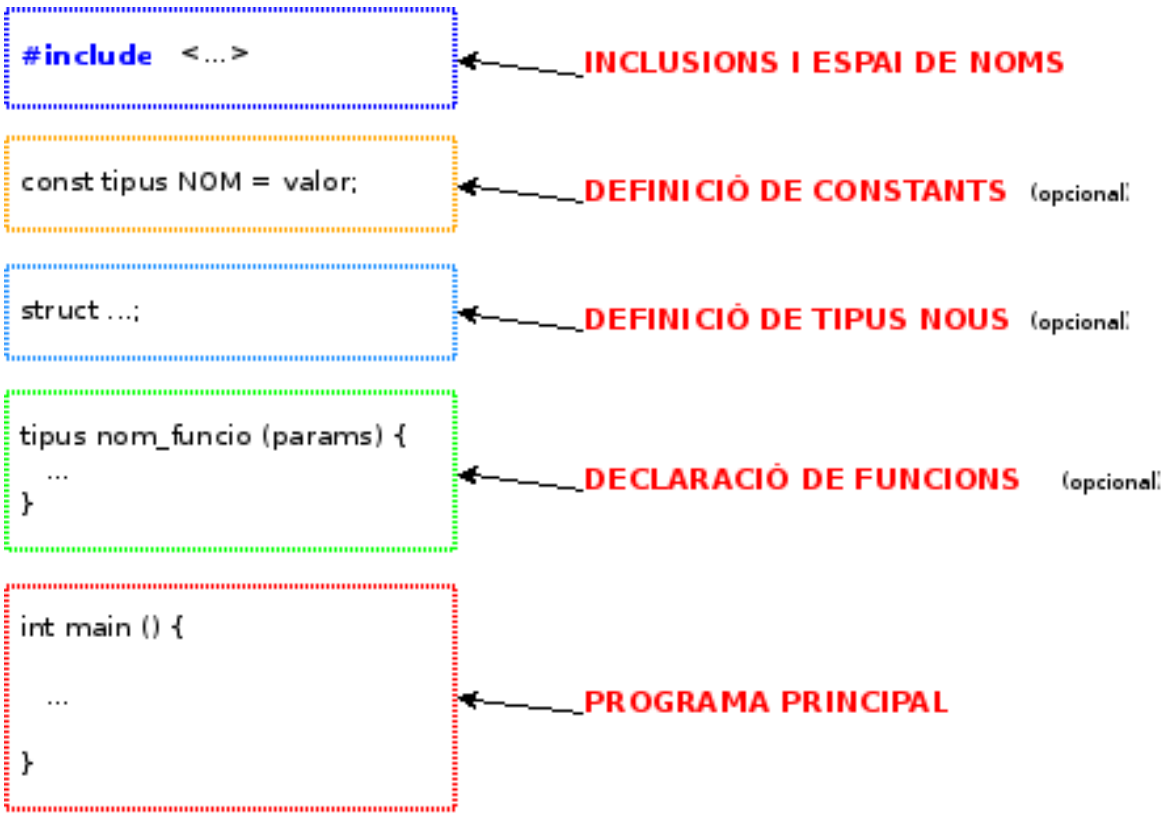
- Les tuples sovint tenen algunes propietats que han de ser mantingudes pels algorismes que les manipulen. Per tant, les tuples sovint tenen definits invariants de la representació.
- Per exemple:

```
struct Temps {
  int hores;  // Inv: 0 <= hores < 24
  int minuts;  // Inv: 0 <= minuts < 60
  int segons;  // Inv: 0 <= segons < 60
};
```


Índex

Índex

Estructura d'un programa



Disseny d'estructures de dades

- Fins ara, dissenyar un programa (o una acció o una funció) significava dissenyar un algorisme. L'estructura de les dades era prou senzilla com per no tenir-la en consideració (no era rellevant).
- Ara quan creem un programa, de vegades necessitarem dissenyar estructures per emmagatzemar les dades i resultats intermitjos.
- El disseny d'una bona estructura de dades sovint és crític:
  - per resoldre el problema.
  - per donar una solució més eficient.
- Cal pensar que aquest curs, no deixa de ser un curs introductori a la programació. En propers cursos veurem estructures de dades més complexes.

Índex

Índex

Exemple 1: Racional

Enunciat:

a) Declareu una estructura de dades per emmagatzemar un nombre racional. En cas que sigui necessari indica l'invariant.

b) Feu una funció que donats dos racionals retorni el racional simplificat resultat de la suma d'aquests dos racionals.

Exemple 1: Estructura racional

- La informació que s'ha d'emmagatzemar del nombre racional és el numerador i el denominador. Per tant, un racional es pot declarar com:

```
struct Racional {
  int num;
  int den;  // Inv: den > 0
};
```

- El denominador d'un nombre racional mai pot ser negatiu ni zero per tant, aquesta condició és el que seria l'Invariant.

Exemple 1: Solució suma (I)

```
// Retorna la suma de a i b.
// Pre: a i b són racionals.
// Post: retorna a + b.
Racional suma(Racional a, Racional b) {
  Racional c;
  c.num = a.num*b.den + b.num*a.den;
  c.den = a.den*b.den;
  // c.den > 0 ja que a.den > 0 i b.den > 0

  // caldria simplificar el racional !!!
  [...]
  return c;
}
```

Exemple 1: Solució suma (II)

```
// Retorna la suma de a i b.
// Pre: a i b són racionals.
// Post: retorna a + b.
Racional suma(Racional a, Racional b) {
  Racional c;
  c.num = a.num*b.den + b.num*a.den;
  c.den = a.den*b.den;
  // c.den > 0 ja que a.den > 0 i b.den > 0

  // simplificacio
  simplifica(c);
  return c;
}
```


Exemple 1: Solució simplifica

```
// Simplifica el racional a de manera que cap dels divisors >1 del numerador
// d'a està en el denominador d'a i a la inversa.
// Pre: a és un racional
// Post: simplifica el racional a.
void simplifica(Racional &a) {
  unsigned int div = mcd(a.num, a.den);
  a.num /= div;
  a.den /= div;
}
```

Exemple 1: Solució mcd

```
// Calcula el màxim comú divisor entre a i b.
// Pre: a i b són enters
// Post: retorna el màxim comú divisor de a i b.
unsigned int mcd(int a, int b) {
  if (a < 0) a *= -1;
  if (b < 0) b *= -1;
  unsigned int num;
  if (a < b) num = a;
  else num = b;
  while (a%num != 0 or b%num != 0) {
    --num;
  }
  return num;
}
```


Índex

Exemple 2: Incrementa temps

Enunciat:
Usant aquesta tupla

```
struct Temps {
  int hores; // Inv: 0 <= hores < 24
  int minuts; // Inv: 0 <= minuts < 60
  int segons; // Inv: 0 <= segons < 60
};
```

feu una funció que retorni una tupla *Temps* després d'haver-la incrementat un segon.

```
// Pre: t és temps
// Post: retorna t incrementat en un segon.
Temps increment(Temps t);
```


Exemple 2: Solució incrementa temps (I)

```
// Incrementa el temps t en un segon.
// Pre: t és temps
// Post: retorna el temps incrementat en un segon.
Temps incrementa(Temps t) {
  Temps res = t;
  ++res.segons;
  if (res.segons > 59) {
    res.segons = 0;
    ++res.minuts;
    if (res.minuts > 59) {
      res.minuts = 0;
      res.hores = (res.hores + 1)%24;
    }
  }
  return res;
}
```

Exemple 2: Solució incrementa temps (II)

```
// Incrementa el temps t en un segon.
// Pre: t és temps
// Post: retorna el temps incrementat en un segon.
Temps incrementa(Temps t) {
  Temps res = t;
  ++res.segons;
  res.minuts += res.segons/60;
  res.segons %= 60;
  res.hores += res.minuts/60;
  res.minuts %= 60;
  res.hores %= 24;
  return res;
}
```


Índex

Exemple 3: Lletra més freqüent

Enunciat:
Dissenya una funció que donada una cadena de caràcters no buida retorni la lletra més freqüent en el text i la seva freqüència (sense importar si el caràcter està en majúscula o minúscula). En cas que hi hagi més de d'un caràcter amb freqüència màxima ha de tornar el caràcter menor.
Nota: Per simplicitat pots suposar que les lletres de la cadena de caràcters pertanyen a l'alfabet anglès.

Exemple 3: Definició de la tupla

- Per tal de resoldre l'exercici atès que hem de tornar dos valors usarem la següent estructura de dades:

```
struct Resultat {
  char lletra;
  unsigned int freq;
};
```

Exemple 3: Solució lletra més freqüent (I)

```
// Torna el caràcter més freqüent de s i la freqüència d'aquest caràcter en
// aquesta cadena de caràcters. En cas que hi hagi més de dos caràcters
// amb freqüència màxima retorna el caràcter menor.
// Pre: s és una cadena de caràcters no buida
// Post: Retorna el caràcter de s més freqüent i també la freqüència
//   d'aquest.
Resultat mes_frequent(const string &s) {
  vector <unsigned int> v(26, 0);
  unsigned int mida = s.size(), total = 0;
  for (unsigned int i=0; i < mida; ++i) {
    // Inv: v conté les freqüències dels caràcters de la subcadena s[0..i-1]
    if (es_lletra(s[i])) {
      char c = minuscula(s[i])
      ++v[int(c - 'a')];
      ++total;
    }
  }
                                                    // CONTINUA »
```


Exemple 3: Solució lletra més freqüent (II)

```
  // « VE DE LA PÀG. ANTERIOR

  Resultat res = {'a', v[0]};
  for (unsigned int i=1; i < 26; ++i) {
    // Inv: res conté el caràcter i la freqüència que és més freqüent en el
    //   subvector v[0..i-1]
    if (res.freq < v[i]) {
      res.lletra = char('a' + i);
      res.freq = v[i];
    }
  }
  if (total != 0) res.freq = res.freq*100/total;
  return res;
}
```

Exemple 3: Solució lletra més freqüent (III)

```
// Indica si c és una lletra.
// Pre: -
// Post: Torna true ssi c és una lletra; false en cas contrari.
bool es_lletra(char c) {
  return (c >= 'A' and c <= 'Z') or (c >= 'a' and c <= 'z');
}

// Transforma el caràcter c a minúscula.
// Pre: c és una lletra.
// Post: Torna la transformació de la lletra c a minúscula.
char minuscula(char c) {
  char res = c;
  if (c >= 'A' and c <= 'Z')  res = c - 'A' + 'a';
  return res;
}
```


Índex

Exercicis de repàs (I)

1. Feu un programa que llegeixi dos nombres complexos i els multipliqui. Per això seguiu els següents passos:
   a) Declareu una estructura de dades per emmagatzemar un nombre complex.
   b) Feu una funció que llegeixi un nombre complex.
   c) Feu una funció que donats dos nombres complexos retorni la multiplicació.
   d) Feu un programa que utilitzi les funcions anteriors per llegir dos nombres complexos i calcular la multiplicació.

   Recordeu que:
   $(a + bi)(a' + b'i) = (aa' - bb') + (ab' + ba')i$

Exercicis de repàs (II)

2. Feu una funció que donada una matriu de reals que representa els resultats d'un examen per un grup d'estudiants retorni la nota mitjana global d'aquest grup, el nombre d'aprovats i el nombre de suspesos. Cal tenir present que: les files de la matriu representen els estudiants, les columnes els problemes de l'examen i la suma de totes les columnes d'una matriu és la nota d'aquest estudiant.

Reconeixement i llicència

Aquesta sessió ha utilitzat el següent material:

- **Portal minidosis**. `http://www.minidosis.org/` fet per en *Pau Fernández*.
- **Material docent de l'assignatura PRO1 de la FIB**. `http://www.cs.upc.edu/~pro1/` fet per en *Jordi Cortadella*, *Ricard Gavaldà* i *Fernando Orejas*.

Tema 21

Ordenació 1: usant STL

Fonaments de Programació

Ordenació 1: Usant STL

Bernardino Casas

bcasas@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

- ✓ Conèixer les propietats bàsiques que té l'algorisme d'ordenació ideal.
- ✓ Ser capaç d'ordenar de manera creixent o decreixent un vector d'elements de tipus bàsics (int, double, char, ...) usant l'acció `sort` de la biblioteca STL.
- ✓ Ser capaç d'ordenar un vector de tuples usant l'acció `sort` de la biblioteca STL seguint un criteri establert.

Índex

Índex

Índex

Què és l'ordenació?

Definició 1: Ordenació

Diem que un vector v d'elements està ordenat de manera creixent si per qualsevol i tal que és $0 \leq i < v.size() - 1$, $v[i] \leq v[i+1]$; és a dir, cada element és més gran que el seu element anterior.

- Per tal de poder ordenar un vector amb elements d'un cert tipus, aquest tipus de dades ha de tenir definida l'operació $\leq$. Els tipus de dades bàsics la tenen definida en C++.
- Quan es vol ordenar un vector v d'elements s'acostuma a deixar el resultat en el mateix vector v.
- Una altra problemàtica similar és ordenar una part concreta del vector.

Exemple d'ordenació

Donat el següent vector d'enters:

3	-1	9	0	-3	2	5	7	-2	6	4

el resultat després d'aplicar un algorisme d'ordenació seria el següent:

-3	-2	-1	0	2	3	4	5	6	7	9

Índex

Propietats de l'algorisme d'ordenació ideal (I)

1. **Estable**. Les claus iguals no es reordenen. En cas que hi hagi claus iguals aquestes apareixerien en el mateix ordre que tenien originalment.
   Per exemple, donat el següent vector:

   $$[2_1, 7, 2_2, 5, 3, 2_3, 1]$$

   on els subíndexs indiquen l'ordre dels elements que són iguals, després d'ordenar hauria de quedar de la següent forma:

   $$[1, 2_1, 2_2, 2_3, 3, 5, 7]$$

Propietats de l'algorisme d'ordenació ideal (II)

2. **In situ**. No requereix espai auxiliar. Opera amb el vector, sense requerir l'ús un vector auxiliar.
3. En el pitjor cas fa $O(n \log n)$ comparacions de claus.
4. En el pitjor cas fa $O(n)$ intercanvis.
5. **Adaptatiu**. Accelera a $O(n)$ quan les dades estan gairebé ordenades o quan hi ha unes poques claus úniques.

No existeix l'algorisme d'ordenació ideal

COMPTE!!

No hi ha cap algorisme que compleixi totes aquestes propietats. La tria de l'algorisme d'ordenació dependrà del problema que s'estigui ressolent.

Índex

Índex

L'acció sort de mòdul algorithm

- La biblioteca standard de C++ disposa d'una acció d'ordenació anomenada sort.
- L'algorisme que usa aquesta acció pot variar segons la versió de la biblioteca de STL, però la majoria de vegades és **quicksort**.
- Per usar-lo cal incloure el mòdul algorithm:

```
#include <algorithm>
```

- Per ordenar un vector *v* d'un cert tipus de manera creixent aquest tipus ha de tenir definida l'operació de comparació <=. La crida a sort és la següent:

```
sort(v.begin(), v.end());
```

Índex

sort amb altres criteris d'ordenació (I)

- Per ordenar amb altres criteris cal cridar l'acció `sort` de la següent manera:

```
sort(v.begin(), v.end(), comp);
```

  on `comp` és la funció de comparació entre elements que es farà servir enlloc de $\leq$.
- Aquesta funció ha de tenir dos paràmetres del tipus pel que està format el vector que es vol ordenar i ha de retornar un booleà.

sort amb altres criteris d'ordenació (II)

- Per exemple, per ordenar un vector d'enters de manera decreixent es definiria comp així:

```
bool comp(int a, int b) {
  return a > b;
}
```

- Per exemple, per ordenar un vector de cadenes de caràcter segons la llargada de paraula i en cas que tinguin la mateixa mida segons ordre alfabètic de les paraules es definiria comp així:

```
bool comp(const string &a, const string &b) {
  bool res;
  if (a.size() != b.size()) res = a.size() < b.size();
  else res = a < b;
  return res;
}
```


sort usat en vector de tuples

- sort també permet ordenar tipus de dades creats per nosaltres (p. ex tuples).
- Si el tipus Persona és el següent:

```
struct Persona {
  string nom;
  unsigned int edat;
  string correu;
};
```

- Per ordenar persones per l'edat i després pel nom:

```
bool comp(const Persona &a, const Persona &b) {
  bool res;
  if (a.edat == b.edat) res = a.nom < b.nom;
  else res = a.edat < b.edat;
  return res;
}
```

Índex

Exercicis de repàs

1. Fes una **funció** que donat un vector d'Empleats retorni una còpia d'aquest vector amb les dades ordenades per *Sou* de manera decreixent. En cas que dos empleats tinguin el mateix *Sou* s'hauria d'ordenar pels *Cognoms* en ordre alfabètic. En cas que dos empleats tinguin el mateix *Sou* i *Cognoms* caldria ordenar per *Nom* en ordre alfabètic.

   El tipus Persona és el següent:

```
struct Empleat {
  double sou; // sou > 0
  string nom, cognoms;
};
```

Reconeixement i llicència

Aquesta sessió ha utilitzat el següent material:

- **Portal Sorting Algorithm Animations**. `http://www.sorting-algorithms.com/` fet per en *Sorting-Algorithms.com*.
- **Material docent de l'assignatura PRO1 de la FIB**. `http://www.cs.upc.edu/~pro1/` fet per en *Jordi Cortadella*, *Ricard Gavaldà* i *Fernando Orejas*.

Tema 22

Ordenació 2: algorismes bàsics

Fonaments de Programació

Ordenació 2: Algorismes bàsics

Bernardino Casas

bcasas@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

- ✓ Conèixer el funcionament dels algorismes d'ordenació bàsics: inserció, selecció i bombolla.
- ✓ Ser capaç d'aplicar a un cas concret els algorismes d'ordenació bàsics (ordenar manualment un vector seguint aquests algorismes).
- ✓ Entendre les diferències entre aquests algorismes d'ordenació, i raonar com d'eficients són.

Índex

Índex

Índex

Introducció

Definició 1: Ordenació per selecció

L'**ordenació per selecció** (anglès: *Selection sort*) és un algorisme d'ordenació que va seleccionant cada vegada l'element mínim (en el cas que l'ordre sigui ascendent) de la part no ordenada i el col·loca al final de la part ordenada.

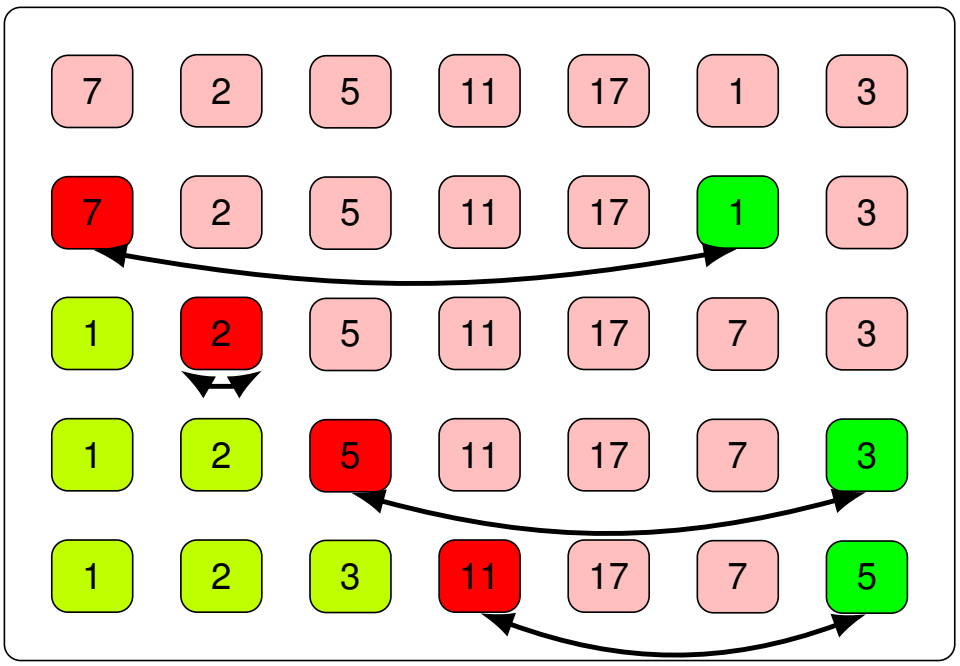
- Aquest algorisme manté dos subvectors:
  - Un subvector on els elements estan ordenats
  - Un subvector que conté la resta dels elements que encara no s'han ordenat.
- L'ordenació per selecció és un algorisme *inestable* (pot canviar l'ordre relatiu dels elements iguals) i *que no necessita espai auxiliar*.

Com funciona el Selection sort?

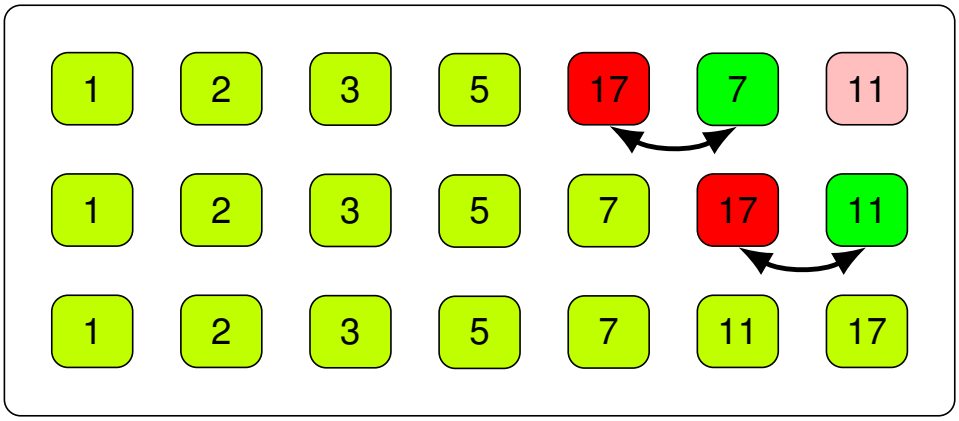
- En la primera iteració es busca l'element més petit i es situa en la primera posició del vector.
- En la segona iteració es busca el segon element més petit i es situa en la segona posició del vector.
- En la tercera, el tercer.
- De manera general, a la iteració i-èssima se selecciona l'element i-èssim més petit i es situa a la posició *i*.

Idea: En cada iteració del selection sort, l'element més petit del subvector sense ordenar es mou al subvector ordenat.

Exemple d'ordenació per selecció (I)



Exemple d'ordenació per selecció (II)



Invariant del Selection sort

L'algorisme compleix aquest invariant:

part ordenada (i-1) | part no tractada

elements ordenats	? ? ?

- La *part ordenada* conté els i-1 elements més petits.
- Tots els elements de la *part no tractada* són més grans o iguals que els elements de la part ordenada.

Info extra selection sort

Fonts extres

- `https://www.youtube.com/watch?v=Ns4TPTC8whw` (Compte la versió del selection sort que apareix en aquest vídeo no és exactament igual que el que veurem a continuació).
- `https://en.wikipedia.org/wiki/Selection_sort`

Algorisme `selection_sort` (I)

```
// Ordena creixentment el vector buscant o seleccionant el següent element
// a ordenar.
// Pre: –
// Post: v està ordenat de manera creixent.
void selection_sort(vector<int> &v) {
  unsigned int ultim = v.size() - 1;
  for (unsigned int i = 0; i < ultim; ++i) {
    // Inv: v[0..i-1] està ordenat de manera creixent, i a més si a < i <= b
    //    llavors v[a] <= v[b]
    unsigned int k = pos_min(v, i, ultim);
    swap(v[k], v[i]);
  }
}
```

Algorisme `selection_sort` (II)

```
// Obté la posició de la casella del subvector v[left..right] que conté
// l'element mínim.
// Pre: 0 <= left <= right < v.size()
// Post: Retorna la posició pos tal que està entre left i right, i a més
//    v[pos] és l'element més petit del vector en el rang v[left..right]
unsigned int pos_min(const vector<int> &v,
                     unsigned int left,
                     unsigned int right) {
  unsigned int pos = left;
  for (unsigned int i = left + 1; i <= right; ++i) {
    // Inv: v[pos] és l'element mínim del vector en el rang v[left..i-1]
    if (v[i] < v[pos]) pos = i;
  }
  return pos;
}
```


Cost del Selection sort (I)

- Aquest algorisme no depèn de com estan distribuits els elements en el vector. Per tant, el cas pitjor, millor i mig són iguals.
- Seleccionar l'element mínim requereix comparar n elements (fent $n-1$ comparacions) i a continuació intercanviar-lo amb el primer element del subvector. Trobar el següent mínim implica comparar amb $n-1$ elements ($n-2$ comparacions), etc.
- El nombre total de comparacions en un vector de mida n és:

$$(n-1)+(n-2)+\ldots+1=\frac{n(n-1)}{2}=\frac{n^2-n}{2}\in\Theta(n^2)$$

Cost del Selection sort (II)

- L'ordenació per selecció té una complexitat de l'ordre de $\Theta(n^2)$ sent n la mida del vector.
- Atès que hem de canviar de lloc l'element mínim $n-1$ vegades (l'últim element ja està ben col·locat), pel vector mida n es faran $n-1$ intercanvis. Per tant, es faran $3(n-1)$ d'assignacions.

Quan usarem l'ordenació per selecció?

Tenint en compte que mai es faran més de $O(n)$ intercanvis, aquest algorisme pot ser útil quan escriure a memòria és una operació costosa.

Índex

Introducció

Definició 2: Ordenació per inserció

L'**ordenació per inserció** (anglès: *Insertion sort*) és un algorisme d'ordenació que recorre el vector seqüencialment i inserta cada element no ordenat en el lloc que li correspon.

- Aquest algorisme és:
  - *Adaptatiu*: eficient per vectors que ja estan gairebé ordenats.
  - *Estable*: no canvia l'ordre relatiu dels elements iguals.
  - *In situ*: no requereix espai auxiliar.

Com funciona l'ordenació per inserció?

- En un moment concret de l'algorisme tenim aquesta situació (x és l'element que estem processant en aquest moment):

part ordenada | part desordenada

$\leq x$	$> x$	x	. . .

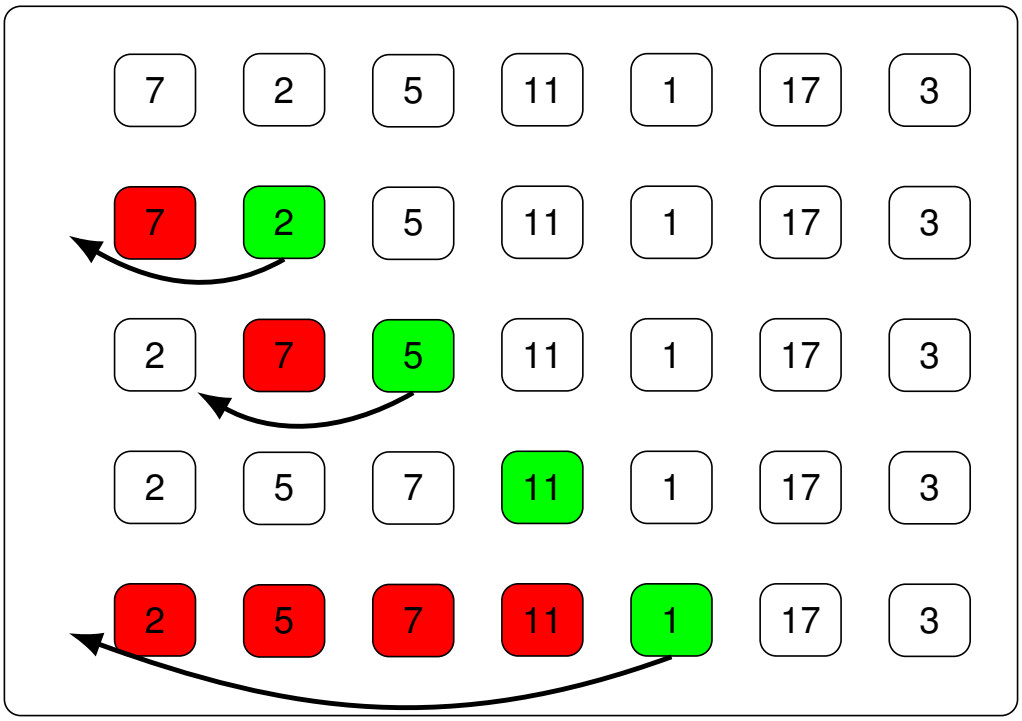
- Després d'executar una iteració de l'algorisme el vector tindria aquesta disposició:

part ordenada | part desordenada

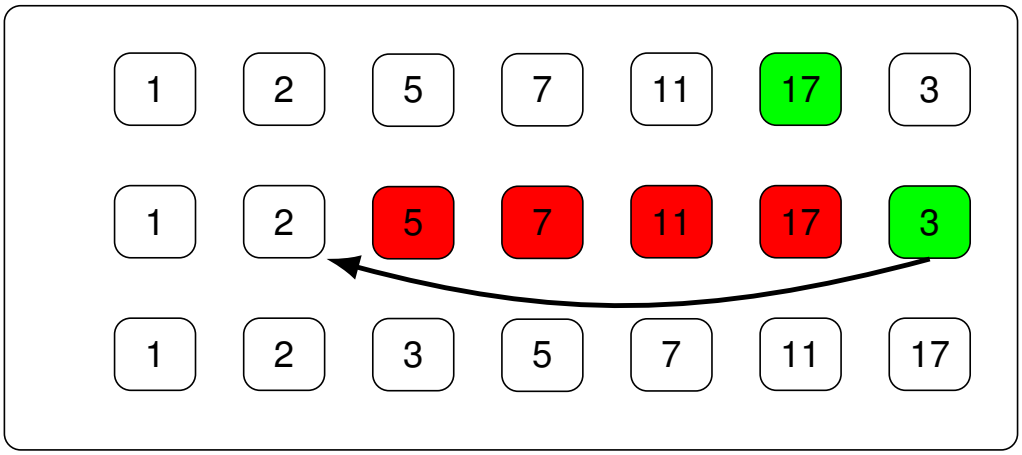
$\leq x$	x	$> x$	. . .

Idea: En cada iteració es va inserint cada element en la posició adequada de la part ordenada del vector.

Exemple d'ordenació per inserció (I)



Exemple d'ordenació per inserció (II)



Invariant de l'Insertion sort

L'algorisme compleix aquest invariant:

part ordenada	part no tractada
i-1	
elements ordenats	? ? ?

- La part ordenada té tots els elements ordenats.
- La part no tractada pot estar sense ordenar, però no sabem res dels elements que hi ha aquí.

Info extra Insertion sort

Fonts extres

- `https://www.youtube.com/watch?v=ROalU379l3U`
- `https://en.wikipedia.org/wiki/Insertion_sort`

Algorisme `insertion_sort`

```
// Ordena creixentment el vector recorrent-lo seqüencialment i insertant
// l'element no ordenat en el seu lloc.
// Pre: –
// Post: v està ordenat de manera creixent.
void insertion_sort(vector<int> &v) {
  unsigned int mida = v.size();
  for(unsigned int i = 1; i < mida; ++i) {
    // Inv: v[0..i-1] està ordenat de manera creixent.
    int x = v[i];
    int j = i;
    while (j > 0 and v[j - 1] > x) {
      // Inv: v[0..j-1] i v[j+1..i] està ordenat de manera creixent.
      v[j] = v[j-1];
      --j;
    }
    v[j] = x;
  }
}
```


Cost de l'ordenació per inserció

- Atès que el cost de l'algorisme d'ordenació per inserció depèn de com estan distribuïdes les dades en el vector, cal fer una anàlisi per casos.
- Analitzarem el cost d'aquest algorisme pels següents casos:
  - cas pitjor: cota superior
  - cas millor: cota inferior
  - cas mig: cota ajustada (comportament mitjà de l'algorisme)

Cost insertion sort: Cas pitjor (I)

- El *cas pitjor* més simple succeeix quan cada element s'ha de moure el màxim possible.
- Això passa, per exemple, quan el vector està ordenat de manera decreixent i es vol ordenar de manera creixent. Per exemple: A = {4, 3, 2, 1}.
- Per inserir l'últim element hem de fer com a màxim $n-1$ comparacions i com a màxim $n-1$ intercanvis. Pel penúltim element, es necessiten com a màxim $n-2$ comparacions i $n-2$ intercanvis, etc.

Cost insertion sort: Cas pitjor (II)

- El nombre total de comparacions que es necessiten en aquest algorisme en el cas pitjor és:

$$1+2+\ldots+(n-2)+(n-1)=\frac{n(n-1)}{2}=\frac{n^2-n}{2}\in\Theta(n^2)$$

- Tenint en compte això la complexitat d'aquest algorisme en el cas pitjor és de l'ordre de $\Theta(n^2)$, on n és la mida del vector.

Cost insertion sort: Cas millor

- El *cas millor* es dona quan el vector ja està ordenat. Per exemple: A = {1, 2, 3, 4 }.
- Es compara l'element a ordenar amb l'element més a la dreta del subvector ordenat. A cada iteració s'acaba fent una única comparació, és a dir, el total de comparacions és:

$$1 + 1 + \ldots + 1 = n - 1 \in \Theta(n)$$

- En el cas millor l'algorisme d'ordenació per inserció té cost de l'ordre $\Theta(n)$, on n és la mida del vector.

Cost insertion sort: Cas mig (I)

- A la iteració i-èssima tindrem la següent situació:

a[0]	a[1]	a[2]	...	a[i-1]	a[i]

on l'element a[i] és el que volem ordenar i per tant col·locar correctament en el subvector $a[0..i-1]$.

- Cal recordar que aquest subvector està ordenat i col·locant el nou element ha de continuar ordenat.
- En el cas mig només hauríem de fer la meitat de les comparacions que fem en el cas pitjor ja que l'element s'hauria de col·locar al mig, és a dir, es faran $\frac{i-1}{2}$ comparacions.

Cost insertion sort: Cas mig (II)

- El nombre total de comparacions que es fan en el cas mig és:

$$\sum_{i=1}^{n} \frac{i-1}{2} = \frac{1}{2}\sum_{i=1}^{n}(i-1) = \frac{1}{2}(1+2+\ldots+(n-1)) =$$

$$= \frac{1}{2}\frac{n(n-1)}{2} = \frac{n^2-n}{4} \in \Theta(n^2)$$

- En el cas mig l'algorisme d'ordenació per inserció té cost de l'ordre $\Theta(n^2)$, on n és la mida del vector.

Índex

Introducció

Definició 3: Ordenació de bombolla

L'**ordenació de bombolla** (anglès: *bubble sort*) és l'algorisme d'ordenació més simple que funciona comparant cada parell adjacent d'elements del vector a ordenar i intercanviant-los en cas que estiguin en l'ordre incorrecte.

- Es considera un dels pitjors algorismes d'ordenació degut a que és dels que triga més en ordenar un vector d'elements.
- Aquest algorisme és *adaptatiu*, *estable* i *no necessita espai auxiliar*.

Com funciona el bubble sort?

- Recorre el vector diverses vegades, intercanviant els elements adjacents si estan en l'ordre incorrecte.
- L'algorisme acaba quan no es produeix cap intercanvi en un dels recorreguts del vector.

Cal tenir present que el *bubble sort* sempre necessita una iteració extra que passi per tot el vector per detectar que el vector ja està ordenat.

Invariant del bubble sort

L'algorisme compleix aquest invariant:

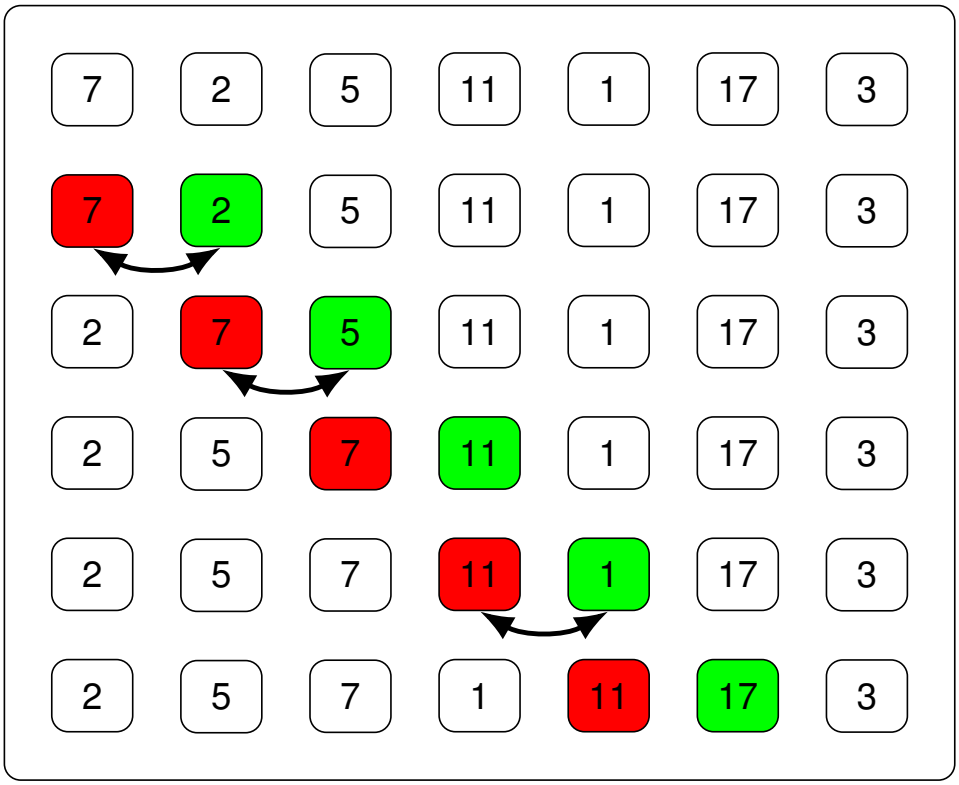
part sense completar | part ordenada

i-1

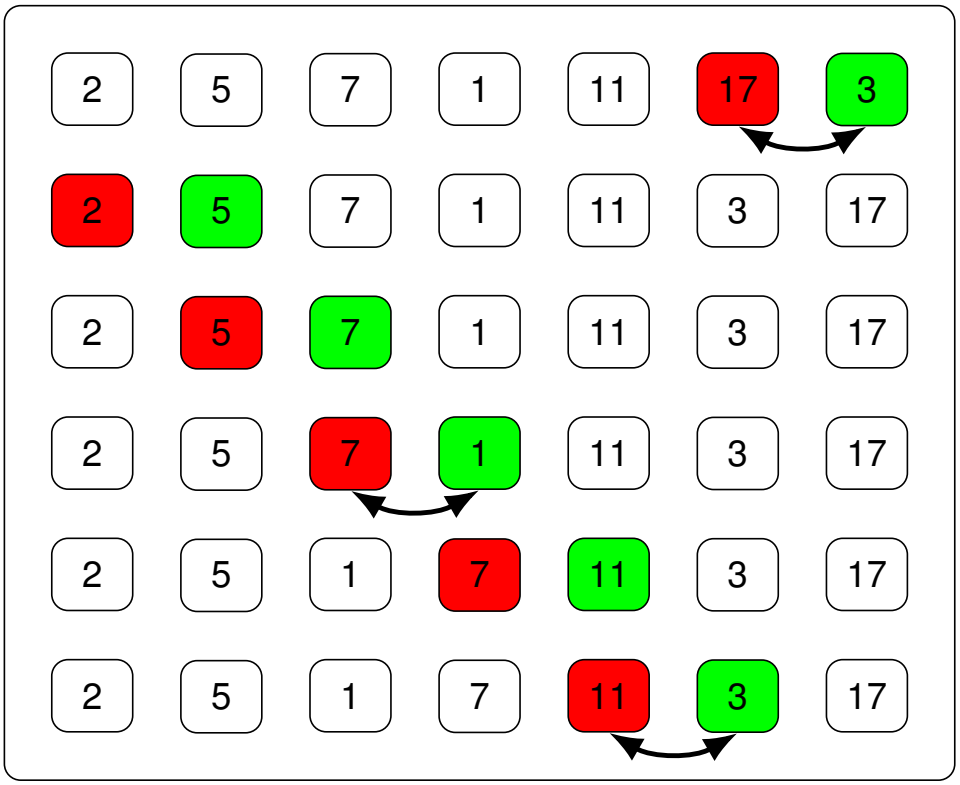
? ? ?	elements ordenats

- La part ordenada està al final del vector i té tots els elements ordenats.
- Tots els elements de la part no completada són menors o iguals que els elements de la part ordenada.

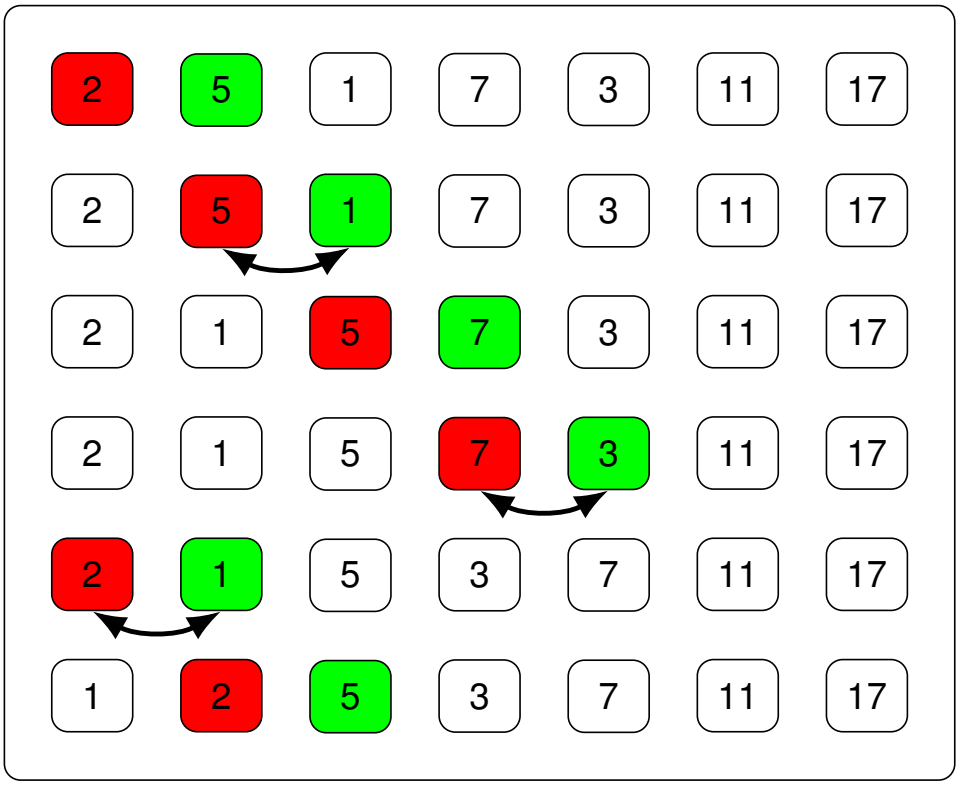
Exemple d'ordenació de bombolla (I)



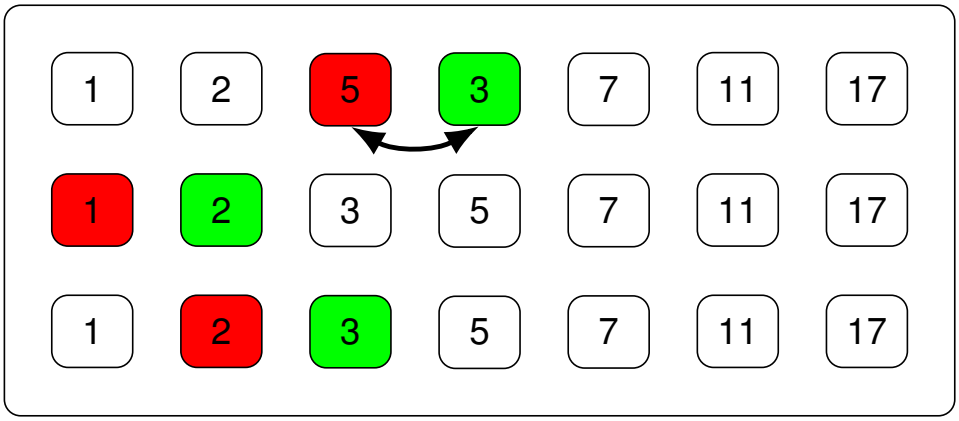
Exemple d'ordenació de bombolla (II)



Exemple d'ordenació de bombolla (III)



Exemple d'ordenació de bombolla (IV)



Info extra bubble sort

Fonts extres

- `https://www.youtube.com/watch?v=lyZQPjUT5B4`
- `https://en.wikipedia.org/wiki/Bubble_sort`

Algorisme `bubble_sort`

```
// Ordena creixentment el vector fent recorreguts (intercanviant elements contigus)
// tantes vegades com sigui necessari.
// Pre: –
// Post: v està ordenat de manera creixent.
void bubble_sort(vector<int> &v) {
  bool ordenat = false;
  int ultim = v.size() - 1;
  while (not ordenat) {
    // Inv: El subvector v[ultim..size()-1] està ordenat però v encara no està ordenat.
    ordenat = true;
    for (int i = 0; i < ultim; ++i) {
      // Inv: ...
      if (v[i] > v[i + 1]) {
        swap(v[i], v[i + 1]);
        ordenat = false;
      }
    }
    --ultim;    // L'element més gran dels no tractats es troba a ultim
  }
}
```


Cost de l'algorisme de la bombolla: Cas pitjor (I)

- El cas pitjor és aquell en que el vector està ordenat al revés del resultat que volem obtenir.
- Això passa, per exemple, el vector està ordenat de manera decreixent i es vol que estigui ordenat de manera creixent. Per exemple: A = {4, 3, 2, 1}.
- En el cas pitjor l'ordenació de la bombolla fa:
  - La primera volta $\longrightarrow n - 1$ intercanvis.
  - La segona volta $\longrightarrow n - 2$ intercanvis.
  - ...
  - L'última volta $\longrightarrow$ 1 intercanvi.

Cost de l'algorisme de la bombolla: Cas pitjor (II)

- El pitjor nombre d'intercanvis és:

$$(n-1)+(n-2)+\ldots+1=\frac{n(n-1)}{2}=\frac{n^2-n}{2}\in\Theta(n^2)$$

- Per tant, el cost d'aquest algorisme en el cas pitjor és de l'ordre de $\Theta(n^2)$.

Cost de l'algorisme de la bombolla: Cas millor

- El cas millor és aquell en que el vector que volem ordenar ja està ordenat. Per exemple: A = {1, 2, 3, 4}.
- Per tant, només recorrerà un sol cop el vector i en no haver fet cap intercanvi s'acaba.
- Atès que només es visiten els elements del vector un sol cop, el cost de l'algorisme pel cas millor és de l'ordre de $\Theta(n)$, on n és la mida del vector que es vol ordenar.
- Aquest cas demostra que aquest algorisme pot ser eficient per **vectors pràcticament ordenats**.

Cost de l'algorisme de la bombolla: Cas mig (I)

- El cas mig de l'ordenació de la bombolla és similar al cas mig de l'ordenació per inserció.
- La idea és que en el cas mig farem la meitat dels intercanvis que faríem en el cas pitjor.
- En el cas pitjor l'ordenació de la bombolla fa:
  - La primera volta $\longrightarrow \frac{n-1}{2}$ intercanvis.
  - La segona volta $\longrightarrow \frac{n-2}{2}$ intercanvis.
  - ...
  - L'última volta $\longrightarrow$ 0 intercanvis.
- A la iteració i-èssima farà $\frac{i-1}{2}$ intercanvis.

Cost de l'algorisme de la bombolla: Cas mig (II)

- El nombre total d'intercanvis que es fan en el cas mig és:

$$\sum_{i=1}^{n} \frac{i-1}{2} = \frac{1}{2}\sum_{i=1}^{n}(i-1) = \frac{1}{2}(1+2+\ldots+(n-1)) =$$

$$= \frac{1}{2}\frac{n(n-1)}{2} = \frac{n^2-n}{4} \in \Theta(n^2)$$

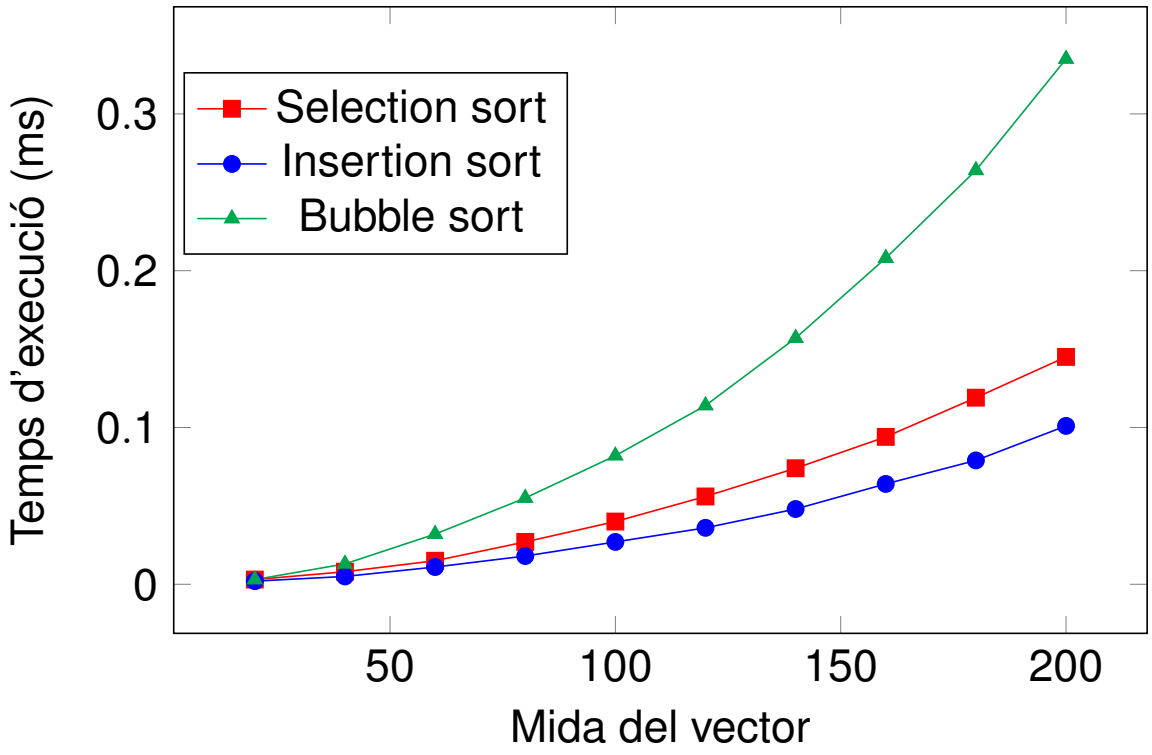
- En el cas mig l'algorisme d'ordenació de la bombolla té cost de l'ordre $\Theta(n^2)$, on n és la mida del vector.

Cost de l'algorisme de la bombolla: Cas mig (III)

- Cal tenir present que un intercanvi implica tres operacions d'assignació.
- Per aquest motiu malgrat els algorismes d'ordenació per inserció i de la bombolla tenen el mateix cost asimptòtic en el cas mig, el de la **bombolla és molt pitjor** ja que fa més intercanvis.
- En general, l'algorisme d'ordenació de la bombolla és un dels algorismes d'ordenació menys eficients. No és gens pràctic per vector molt grans.

Índex

Comparació de l'ordenació en vectors petits



Font: Elaboració pròpia

Resum de les característiques dels algorismes bàsics d'ordenació

Ordenació	In situ	Estable	pitjor	millor	mig
Selecció	✔	✗	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Inserció	✔	✔	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n^2)$
Bombolla	✔	✔	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n^2)$

Conclusions

✔ *Millor algorisme*: Es pot comprovar com el millor algorisme d'ordenació entre aquests tres és l'**ordenació per inserció**.

✗ *Pitjor algorisme*: Per altra banda es veu clarament que l'**ordenació de bombolla** és qui triga més d'aquests algorismes. Per tant aquest algorisme no s'hauria d'utlitzar MAI.

Com ja hem comentat que pot ser interessant utilitzar l'**ordenació per selecció** en cas que escriure a memòria sigui quelcom molt costós i que calgui evitar.

Índex

Exercicis de repàs

1. Representa gràficament pas a pas l'evolució del vector [12, 32, -5, 21, 65, 75, 39, -17, 5] en aplicar l'algorisme d'ordenació per selecció, per inserció i de bombolla.
2. Adapta els algorismes d'ordenació que hem vist de manera que puguin ordenar una part del vector [a...b].
3. Llegeix-te aquest article i vegeu aquest vídeo sobre l'algorisme d'ordenació per fusió (mergesort):
   - `https://www.youtube.com/watch?v=XaqR3G_NVoo`
   - `https://en.wikipedia.org/wiki/Merge_sort`

   Representa gràficament pas a pas com quedaria el vector del primer exercici en aplicar aquest altre algorisme.

Reconeixement i llicència

Aquesta sessió ha utilitzat el següent material:

- **Portal Sorting Algorithm Animations**. `http://www.sorting-algorithms.com/` fet per en *Sorting-Algorithms.com*.
- **Material docent de l'assignatura PRO1 de la FIB**. `http://www.cs.upc.edu/~pro1/` fet per en *Jordi Cortadella*, *Ricard Gavaldà* i *Fernando Orejas*.

Tema 23

Ordenació 3: algorismes avançats

Fonaments de Programació

Ordenació 3: Algorismes avançats

Bernardino Casas

bcasas@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

- ✓ Conèixer el funcionament de l'ordenació per fusió.
- ✓ Ser capaç d'aplicar l'algorisme d'ordenació per fusió a un cas concret (ordenar manualment un vector seguint aquest algorisme).
- ✓ Conèixer el funcionament de l'ordenació ràpida.
- ✓ Ser capaç d'aplicar l'algorisme d'ordenació ràpida a un cas concret (ordenar manualment un vector seguint aquest algorisme).
- ✓ Entendre les diferències entre els diferents algorismes d'ordenació que hem vist i tenir clar com d'eficients són.
- ✓ Ser capaç d'usar l'algorisme d'ordenació més adequat en cada cas.

Índex

Índex

Índex

Introducció

Definició 1: Ordenació per fusió

L'**ordenació per fusió** (anglès: *Merge sort*) és un algorisme d'ordenació que utilitza el principi de divideix i venceràs. Divideix el vector a ordenar en dues meitats, es crida a sí mateix per cadascuna d'aquestes meitats, i després fusiona les dues meitats ordenades.

- Merge sort és un algorisme recursiu. La tasca principal d'aquest algorisme és la fusió dels subvectors.
- La fusió assumeix que els dos subvectors, que es fusionaran en un de sol, estan ordenats.
- Aquest algorisme és *estable* i *necessita espai auxiliar*.

Com funciona l'ordenació per fusió? (I)

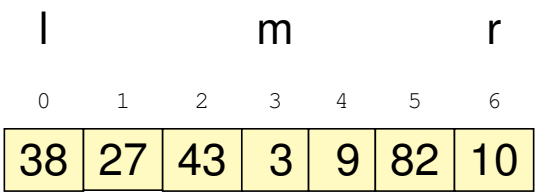
Conceptualment l'ordenació per fusió funciona de la següent manera:

1. Es troba el punt mig del vector per dividir-lo en dues meitats.
2. Es crida a l'acció d'ordenació amb la primera meitat.
3. Es crida a l'acció d'ordenació amb la segona meitat.
4. Es fusionen les dues meitats dels passos 2 i 3 de manera que el subvector resultant està ordenat.

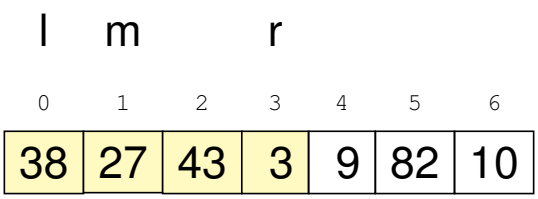
Idea: La divisió de cada subvector en dues meitats es repeteix fins que el subvector només té un element. Un vector amb un sol element es considera ja ordenat.

Exemple de Merge sort (I)

0) Vector inicial

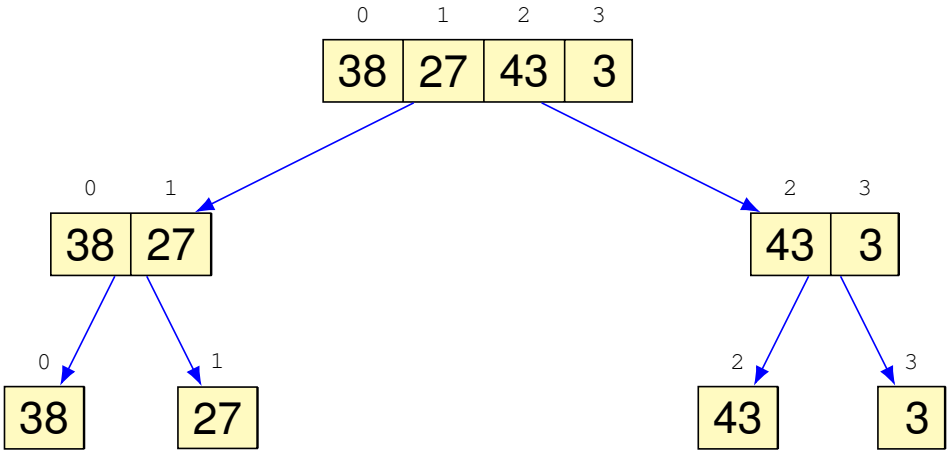


1) Mergesort del subvector



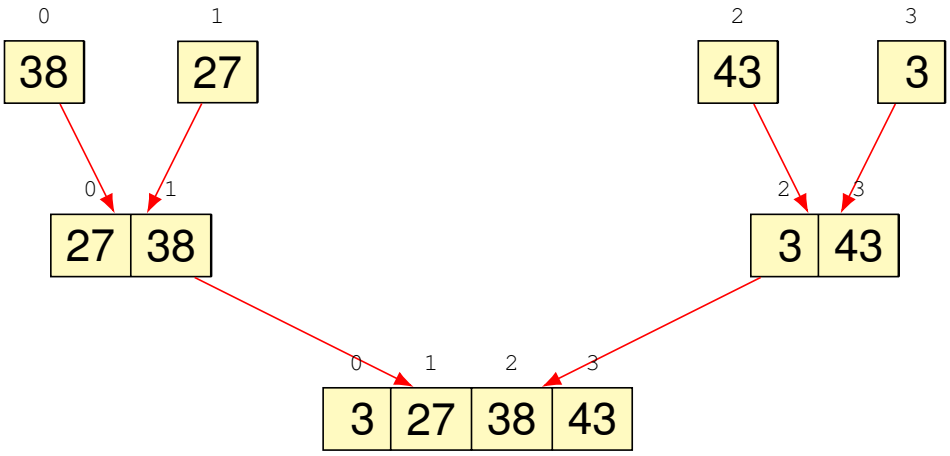
Exemple de Merge sort (II)

2) Mergesort del subvector

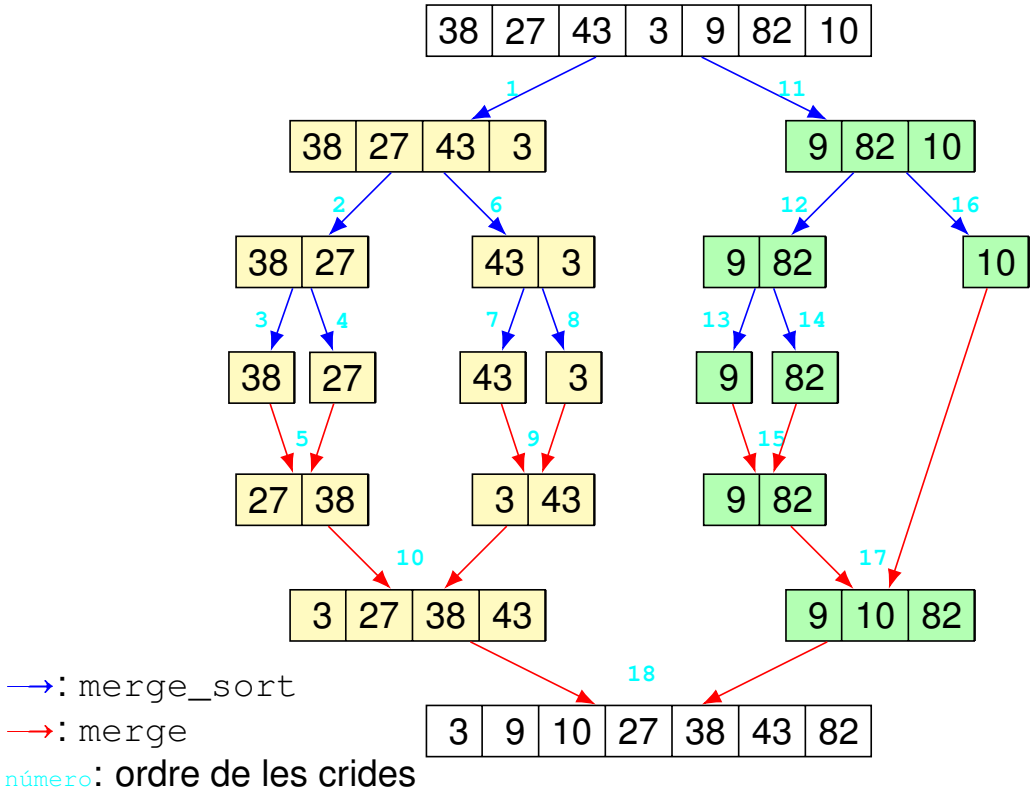


Exemple de Merge sort (III)

3) Merge del subvector



Exemple crides de Merge sort



Info extra merge sort

Fonts d'informació extres

- https://www.youtube.com/watch?v=XaqR3G_NVoo
- https://en.wikipedia.org/wiki/Merge_sort

Algorisme merge_sort

```
// Ordena creixentment el vector dividint-lo en dues parts iguals, fins a
// obtenir vectors d'un element i a continuació els torna a unir de dos
// en dos en l'ordre dels elements.
// Pre: 0 <= left <= right < v.size()
// Post: v[left..right] s'ha ordenat creixentment.
void merge_sort (vector<elem>& v, int left, int right) {
  if (left < right) {
    int m = (left + right)/2;
    merge_sort(v, left, m);
    merge_sort(v, m + 1, right);
    merge(v, left, m, right);
  }
}
```


Algorisme merge (I)

```
// Uneix les dues parts del vector seleccionant iterativament de cada part
// l'element més petit de manera que el subvector v[left..right] queda ordenat.
// Pre: 0 <= left <= mid < right < v.size(), i
//    v[left..mid], v[mid+1..right] estan ordenats creixentment.
// Post: v[left..right] està ordenat
void merge(vector<elem>& v, int left, int mid, int right) {
  int n = right - left + 1, i = left, j = mid + 1, k = 0;
  vector<elem> aux(n);
  while (i <= mid and j <= right) {
    // Inv: el subvector aux[0..k-1] conté els elements ordenats dels
    //    subvectors v[left..i-1] i v[mid+1..j-1]
    if (v[i] <= v[j]) {
      aux[k] = v[i];
      ++i;
    }
    //
```

Continua »

Algorisme `merge` (II)

```
    // « Ve de la pàgina anterior
    else {
      aux[k] = v[j];
      ++j;
    }
    ++k;
  }
  while (i <= mid) {
    // Inv: els elements del subvector v[left..j-1] s'han copiat de manera
    //    ordenada a aux
    aux[k] = v[i];
    ++k;
    ++i;
  }
  //                                              Continua »
```


Algorisme `merge` (III)

```
  // « Ve de la pàgina anterior
  while (j <= right) {
    // Inv: els elements del subvector v[mid+1..j-1] s'han copiat de manera
    //    ordenada a aux
    aux[k] = v[j];
    ++k;
    ++j;
  }
  for (k = 0; k < n; ++k) {
    // Inv: s'ha copiat el subvector aux[0..k-1] en el subvector
    //    v[left..left+k-1].
    v[left+k] = aux[k];
  }
}
```

Cost de l'ordenació per fusió

- Merge sort divideix sempre el vector en dues parts iguals i a continuació ha de recórrer els elements d'aquestes meitats per fusionar-los.
- Per tant, els casos pitjor, millor i mig coincideixen i els tres casos es representen amb la següent equació de recurrència:

$$T(n) = 2T(n/2) + \Theta(n)$$

- La solució de la recurrència anterior és: $n \log n$.
- Per altra banda, l'algorisme necessita $O(n)$ espai addicional on n és igual al nombre d'elements que es volen ordenar.

Índex

Introducció

Definició 2: Ordenació ràpida

L'**ordenació ràpida** (anglès: *Quick sort*) és un algorisme d'ordenació que utilitza el principi de divideix i venceràs. L'algorisme selecciona un element que anomenarem pivot i divideix el vector d'elements al voltant del pivot.

- A diferència d'altres algorismes similars d'ordenació (per ex. Merge sort), no assegura que la divisió dels elements es faci en parts de mida similar.
- Aquest algorisme és *inestable* i *no necessita espai auxiliar*.

Com funciona el Quick sort?

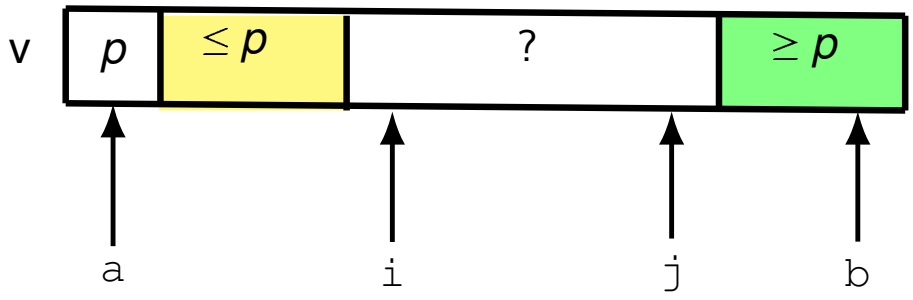
- La tasca clau del Quick sort és la partició.
- L'objectiu d'aquesta tasca és col·locar el pivot p en la posició correcta del vector amb:
  - els elements més petits que p abans (a l'esquerra del pivot),
  - els més grans que p després (a la dreta del pivot).
- Un cop aconseguim tenir el vector amb els elements més petits abans del pivot i els més grans després del pivot, només caldrà ordenar els trossos a l'esquerra i a la dreta del pivot fent dues crides recursives a Quick sort.

Com es tria el pivot?

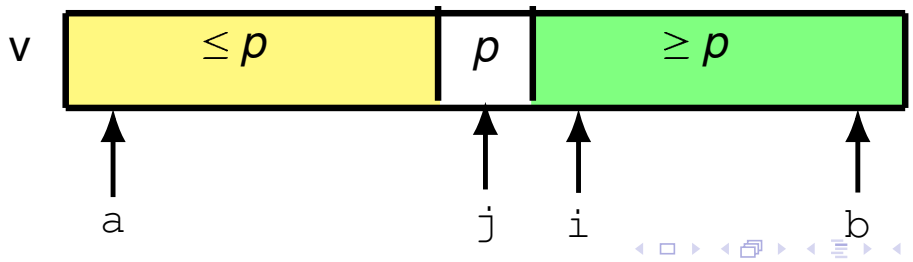
- La tria del pivot es pot fer agafant:
  - el primer element
  - l'últim element
  - un element al·leatori
  - la mediana
- En la implementació que veurem nosaltres el pivot serà el primer element del vector.

Evolució del Quick sort

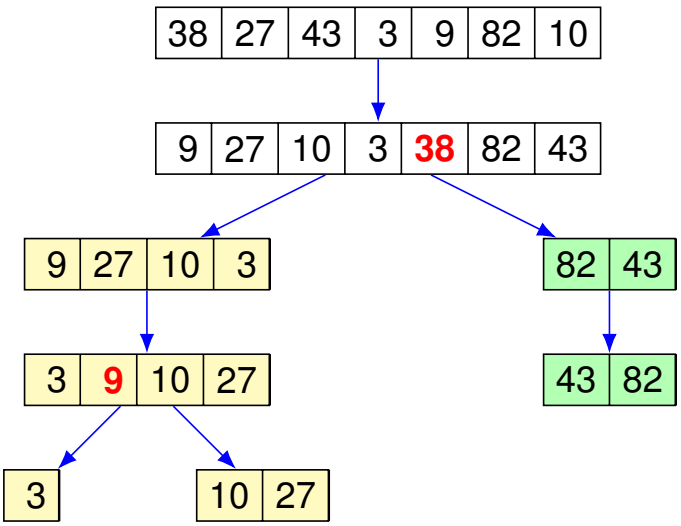
- En un moment concret de l'algorisme tenim aquesta situació (el pivot p és el primer dels elements a ordenar):



- Un cop hem acabat amb la `particio` el vector tindria aquesta situació:



Exemple crides de Quick sort



Quick sort vs Merge sort

- En el Quick sort la fusió no és necessària, doncs els trossos a ordenar ja queden ben repartits a cada banda del pivot.
- Mentre que en el Merge sort la partició és simple i la feina es realitza durant l'etapa de fusió, en el Quick sort és tot el contrari.
- Quick sort no requereix espai addicional d'emmagatzematge, mentre que Merge sort necessita $O(n)$ d'espai extra, on n denota la mida del vector.
- Reservar i alliberar l'espai extra incrementa el temps d'execució d'un algorisme. Per això és millor usar Quick sort quan s'han d'ordenar vectors.

Info extra Quick sort

Fonts d'informació extres

- https://en.wikipedia.org/wiki/quicksort
- https://www.geeksforgeeks.org/quick-sort/

Algorisme `quicksort`

```
// Ordena de manera creixent el subvector v[a..b] segons l'algorisme
// d'ordenació ràpida.
// Pre: a < b < v.size()
// Post: Ordena de manera creixent el subvector v[a..b].
void quicksort(vector<elem> &v,
               unsigned int a, unsigned int b) {
  if (a - b + 1 <= M) {
    // M indica cert llindar a partir del qual és més eficient utilitzar
    // un algorisme d'ordenació simple com Insertion sort.
    // Aquest llindar depèn de la màquina, però habitualment és 10.
  }
  else {
    unsigned int k = particio(v, a, b);
    quicksort(v, a, k - 1);
    quicksort(v, k + 1, b);
  }
}
```

Algorisme particio (I)

```
// Col·loca els elements del subvector v[a..b] de forma que el primer element
// es situa en el lloc que li pertocaria si el subvector estigués ordenat, els
// elements menors que el pivot a l'esquerra i els majors a la dreta.
// Pre: a < b < v.size()
// Post: Retorna la posició on aniria l'element v[a] en cas que el subvector
//    estigués ordenat i el col·loca en aquest lloc. A més col·loca els altres
//    elements davant o darrera d'aquest element depenent de si són
//    menors o majors.
unsigned int particio(vector<elem> &v,
                      unsigned int a, unsigned int b) {
  unsigned int i = a + 1, j = b;
  elem p = v[a];
  while (i < j+1) {
    // Inv: els elements del subvector v[a+1..i-1] són menors que p i
    //    i els elements del subvector v[j+1..b] són majors que p.

    //
```

Continua »

Algorisme particio (II)

```
    // « Ve de la pàgina anterior
    while (i < j+1 and v[i] <= p) {
      // Inv: els elements del subvector v[a+1..i-1] són menors que p
      ++i;
    }
    while (i < j+1 and v[j] >= p) {
      // Inv: els elements del subvector v[j+1..b] són majors que p
      --j;
    }
    if (i < j+1){
      swap(v[i], v[j]);
    }
  }
  swap(v[a], v[j]); // intercanvi entre el pivot i l'últim dels menors
  return j;
}
```

Cost de l'algorisme d'ordenació ràpida

- El temps que triga l'algorisme de Quick sort es pot escriure amb aquesta equació de recurrència:

$$T(n) = T(k) + T(n-k-1) + \Theta(n)$$

- n denota el nombre d'elements del vector que s'ordenaran i k denota el nombre d'elements menors que el pivot.
- Els primers dos termes d'aquesta expressió fan referència a les dues crides recursives.
- L'últim terme fa referència a la partició (cal recórrer tots els elements que es volen ordenar).

Cost Quick sort: Cas pitjor (I)

- El pitjor cas succeeix quan la `particio` sempre agafa els elements més grans o més petits com a pivot. Això passa quan el vector ja està ordenat de manera creixent o decreixent.
- Tenint en compte això es pot escriure com:

$$T(n) = T(0) + T(n-1) + \Theta(n)$$

que és equivalent:

$$T(n) = T(n-1) + \Theta(n)$$

- La solució de la recurrència anterior és: n^2.

Cost Quick sort: Cas pitjor (II)

Com es pot evitar el pitjor cas?

Per tal d'evitar el pitjor cas a l'hora de triar el pivot es pot comparar el primer, l'últim i l'element del mig dels subvector que es vol ordenar. El pivot seria l'element d'aquests tres que no fos ni el major ni el menor.

Cost Quick sort: Cas millor

- El millor cas succeeix quan la `particio` sempre agafa l'element del mig com a pivot. Per tant cada crida recursiva rep la meitat dels elements.
- Tenint en compte això la recurrència es pot escriure com:

$$T(n) = 2T(n/2) + \Theta(n)$$

- La solució de la recurrència anterior és: $n \log n$.

Cost Quick sort: Cas mig (I)

- El cas mig és força més complicat. Per il·lustrar-ho usarem un exemple.
- Si els elements del vector a ordenar fossin [1, 2, 3, 4, 5] per calcular el cas mig hauríem de tenir en compte tots els possibles escenaris (les possibles seleccions del pivot depenent de quin fos el primer nombre del vector):
  - Si el pivot és l'1, llavors els vectors de les crides recursives tindrien mida 0 i 4.
  - Si el pivot és l'2, llavors tindrien mida 1 i 3.
  - Si el pivot fos l'3, llavors tindrien mida 2 i 2.
  - Si el pivot fos l'4, llavors tindrien mida 3 i 1.
  - Si el pivot fos l'5, llavors tindrien mida 4 i 0.

Cost Quick sort: Cas mig (II)

- Atès que cadascun d'aquests escenaris tindrien la mateixa probabilitat (1/5), la recurrència per aquest cas seria:

$$\frac{1}{5}(T(0)+T(4))+\frac{1}{5}(T(1)+T(3))+\frac{1}{5}(T(2)+T(2))+\frac{1}{5}(T(3)+T(1))+\frac{1}{5}(T(4)+T(0))+\Theta(n)$$

- La forma general d'aquesta recurrència seria la suma de cada possible pivot, dividit entre els possibles pivots:

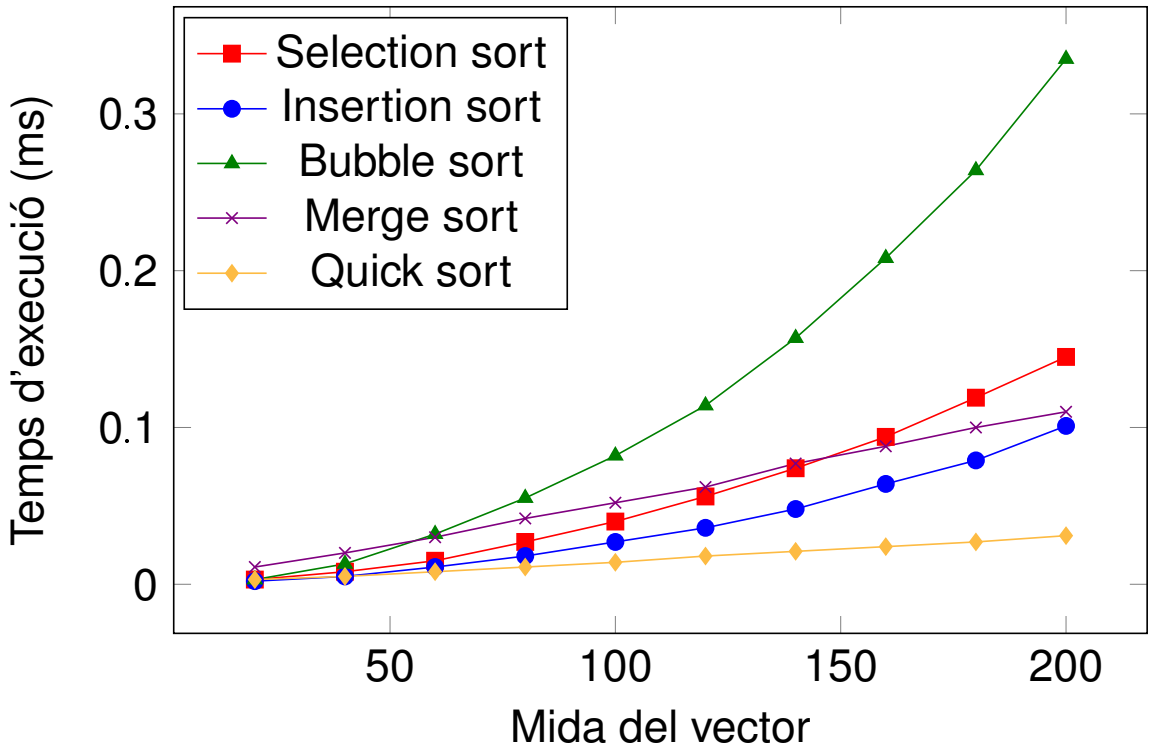
$$T(n)=\left(\frac{1}{n}\sum_{i=0}^{n-1}T(i)+T(n-i-1)\right)+\Theta(n)$$

- La solució d'aquesta recurrència és: $n \log n$.

Índex

Índex

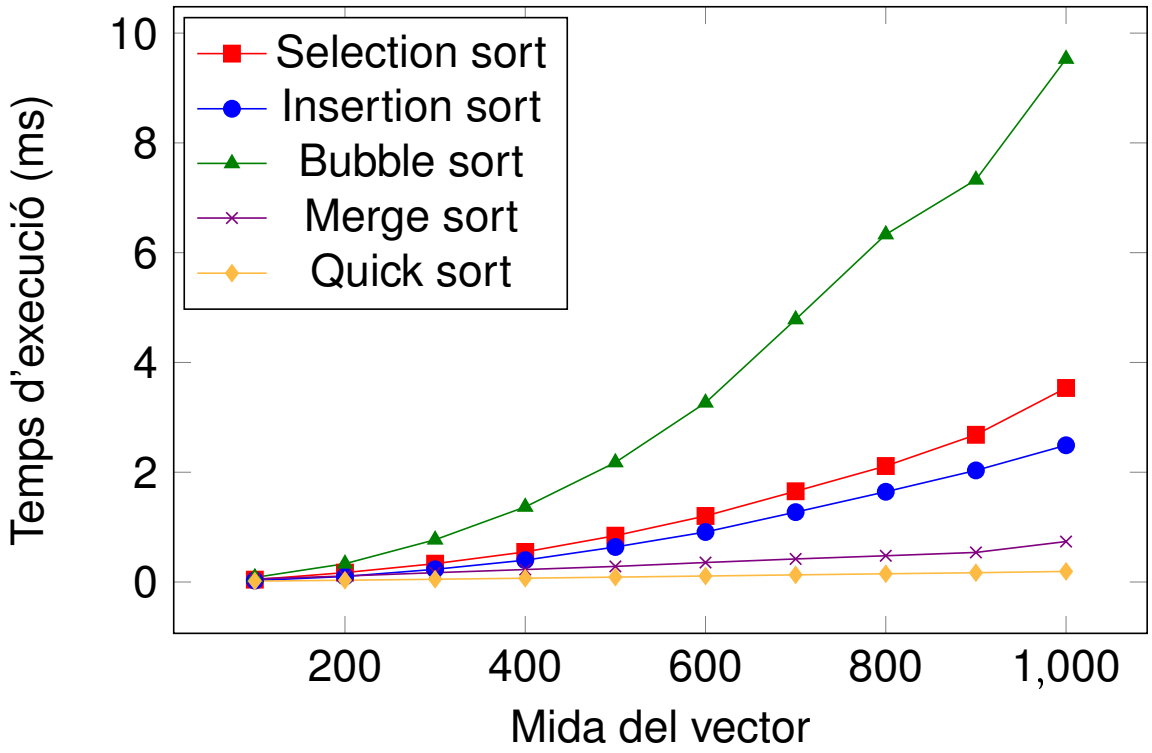
Comparació de l'ordenació en vectors petits



Font: Elaboració pròpia

Índex

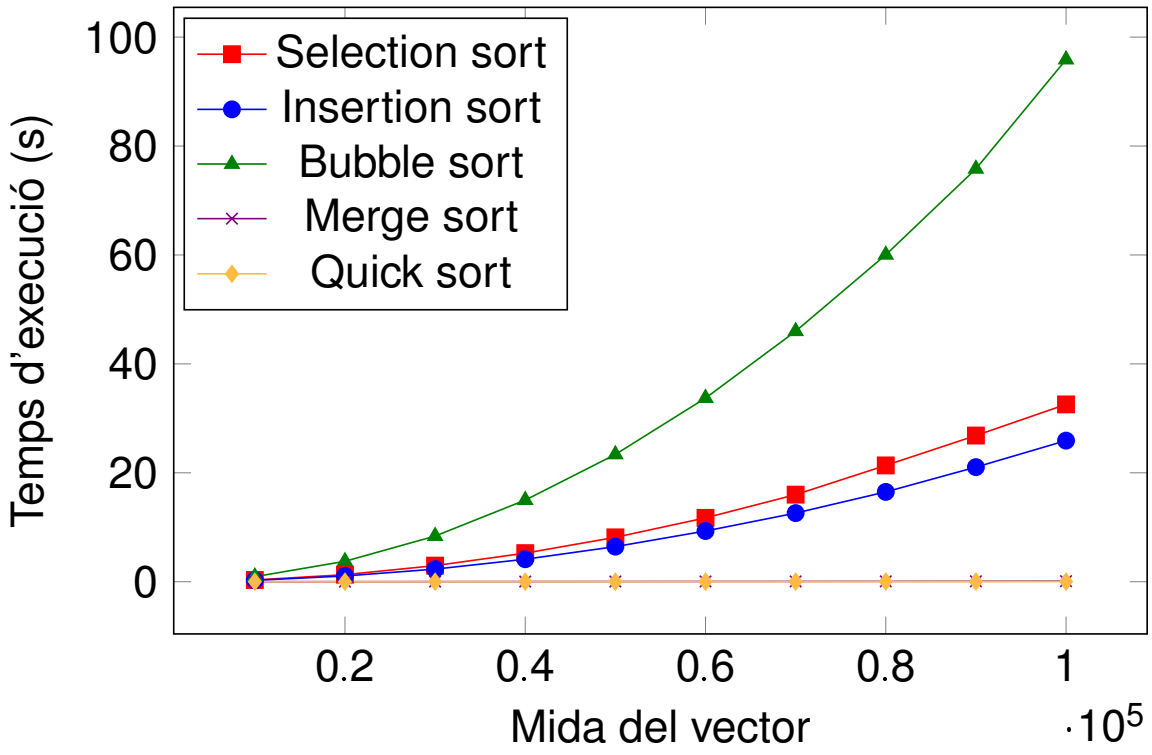
Comparació de l'ordenació en vectors mitjans



Font: Elaboració pròpia

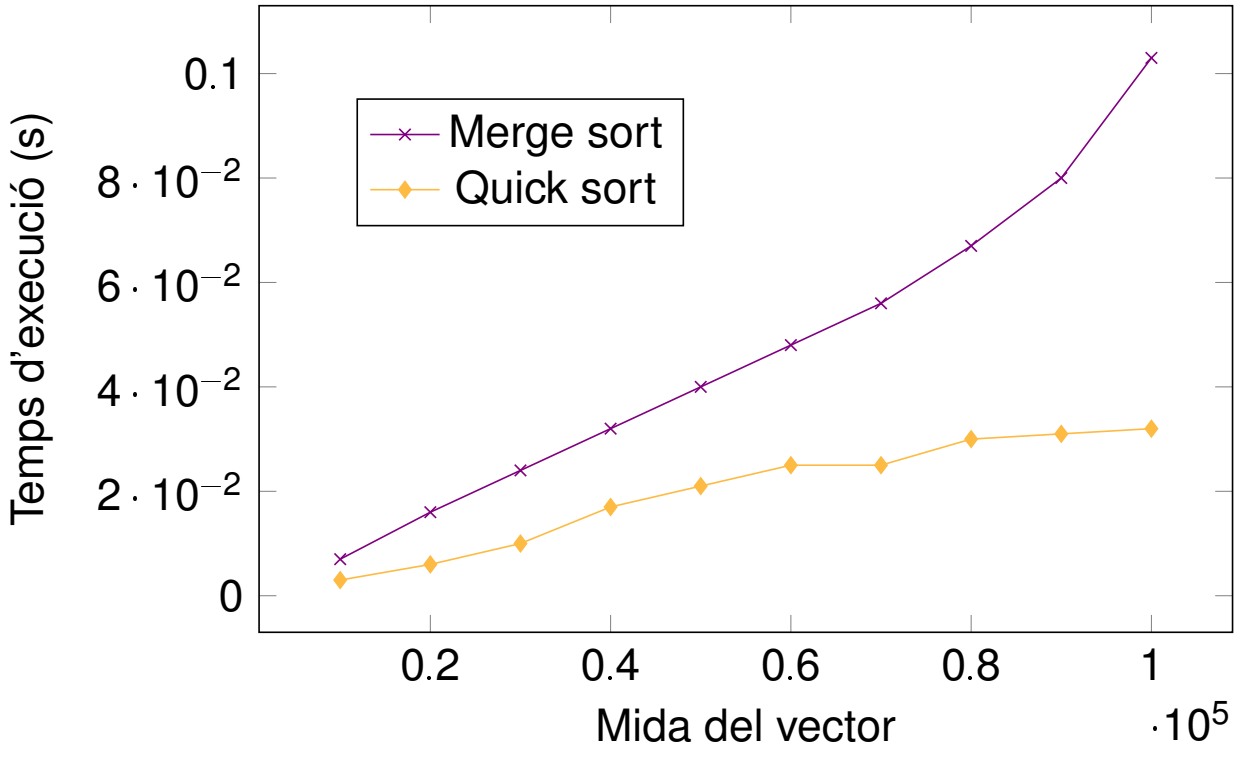
Índex

Comparació de l'ordenació en vectors grans (I)



Font: Elaboració pròpia

Comparació de l'ordenació en vectors grans (II)



Font: Elaboració pròpia

Resum de les característiques dels algorismes d'ordenació

Ordenació	In situ	Estable	Adapt.[1]	pitjor	millor	mig
Selecció	✔	✗	✗	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Inserció	✔	✔	✔	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n^2)$
Bombolla	✔	✔	✔	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n^2)$
Fusió	✗	✔	✗	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$
Ràpida	✔	✗	✗	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n \log n)$

[1] Adaptatiu

Conclusions de l'estudi d'ordenació

- ✔ L'algorisme que ordena un vector més ràpidament és l'algorisme d'**ordenació ràpida**.
- ✔ Només quan el vector té pocs elements l'algorisme més eficient és l'**ordenació per inserció**.

Quan el vector està gairebé ordenat el millor és usar un algorisme d'ordenació *adaptatiu*. En aquest cas la millor opció és usar l'**ordenació per inserció**.

- ✗ Cal destacar que l'algorisme d'~~ordenació de la bombolla~~ és de llarg el que triga més en ordenar. Per tant es desaconsella del tot el seu ús.

Índex

Exercicis de repàs

1. Representa gràficament les crides que es produiran en ordenar el vector [12, 32, -5, 21, 65, 75, 39, -17, 5] usant l'algorisme d'ordenació per fusió. Indica l'ordre en que es realitzaran les diferents crides.
2. Implementa la millora del cas pitjor de l'algorisme d'ordenació ràpida.
3. Mira el següent vídeo:
   - `https://www.youtube.com/watch?v=ywWBy6J5gz8`

   El vídeo mostra una variant del Quick sort. Implementa aquesta variant.

Reconeixement i llicència

Aquesta sessió ha utilitzat el següent material:

- **Material docent de l'assignatura PRO1 de la FIB**. `http://www.cs.upc.edu/~pro1/` fet per en *Jordi Cortadella*, *Ricard Gavaldà* i *Fernando Orejas*.
- **Sorting algorithms - GeeksforGeeks**. `https://www.geeksforgeeks.org/sorting-algorithms`.

Tema 24

Cerca avançada

Fonaments de Programació

Cerca avançada

Bernardino Casas bcasas@cs.upc.edu

Rosa M. Jiménez jimenez@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

- ✓ Conèixer com funciona la cerca dicotòmica.
- ✓ Ser capaç d'aplicar l'algorisme de cerca dicotòmica a un cas concret (cercar manualment un element en un vector seguint aquest algorisme).

Índex

Índex

Índex

Problema 1: Enunciat

Enunciat
Feu una funció que, donat un vector d'enters i un enter, indiqui si aquest enter està present en el vector.

Problema 1: Solució

- Una possible solució seria aplicar simplement l'esquema de cerca sobre vectors.
- Una solució una mica millor seria buscar l'element pels dos cantons del vector alhora.
- Aquesta solució en el cas pitjor (que no es trobi l'element buscat en el vector) encara haurem de visitar tots els elements del vector.

Algorisme cerca per dos costats

```
// Indica si l'element x està en el vector v.
// Pre: v: vector d'enters; x: enter
// Post: retorna true ssi x es troba a v; false en cas contrari.
bool hi_es(const vector<int> &v, int x) {
  bool trobat = false;
  unsigned int i = 0, j, m = v.size();
  if (m%2 == 0) j = m/2;
  else j = m/2 + 1;
  while (not trobat and i < j) {
    // Inv: x no es troba en els subvectors v[0..i - 1] i v[m - i - 1..m - 1]
    //    on m és la mida de v.
    if (x == v[i] or x == v[m-i-1]) trobat = true;
    else ++i;
  }
  return trobat;
}
```

Índex

Problema 2: Enunciat

Enunciat
Feu una funció que, donat un vector d'enters **ordenat** de forma creixent i un enter, indiqui si aquest enter està present en el vector.

Problema 2: Solució

- Una possible solució seria aplicar l'algorisme de cerca per dos costats com ja hem fet.
- Però, atès que el vector està ordenat es pot aplicar l'algorisme de **cerca dicotòmica**.

Índex

Índex

Definició i conceptes generals

Definició 1: Cerca dicotòmica

La **cerca dicotòmica** o binària (anglès: *binary search*) és un algorisme de cerca que troba la posició d'un valor en una taula ordenada. Compara el valor amb l'element en el mig de la taula; si no són iguals, la meitat en la qual el valor no pot estar es descarta i la cerca continua en la meitat restant fins que el valor es trobi o no hi hagi taula on buscar.

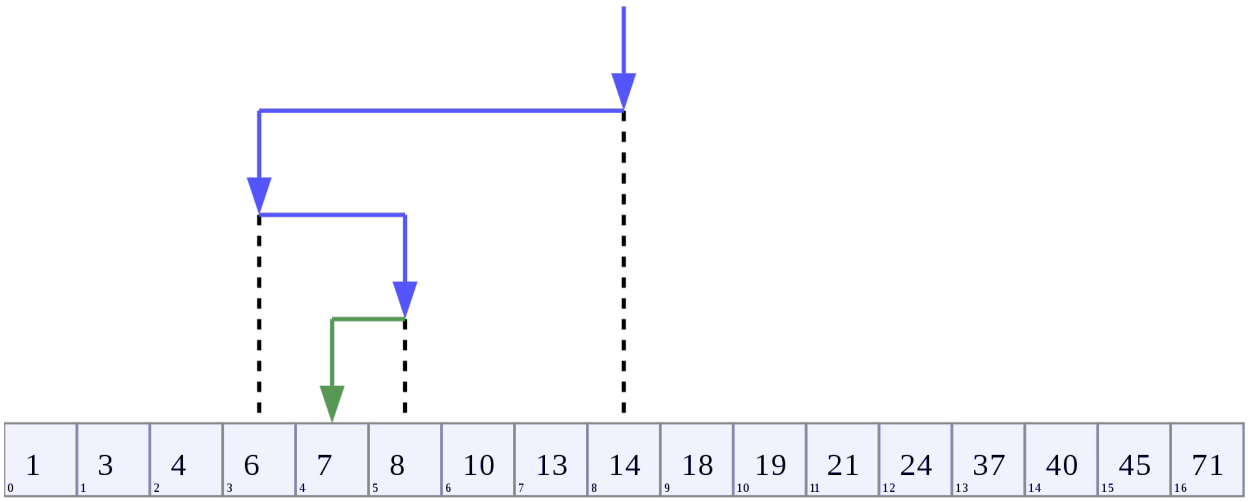
- L'algorisme de **cerca dicotòmica** només es pot aplicar si les dades de la taula estan ordenades.

Funcionament de l'algorisme

1. Es compara l'element a buscar amb l'element central del vector.
2. Si el valor d'aquest és major que el de l'element buscat es repeteix el procediment en la part del subvector que va des de l'inici d'aquest fins a l'element triat.
3. En cas contrari es pren la part del subvector que va des de l'element triat fins al final.
4. S'acaba la cerca quan es troba l'element que estem buscant o el subvector en el que hauríem de buscar no té cap element.

Exemple

Estem buscant el nombre 7 en el següent vector:



Font: https://en.wikipedia.org/wiki/Binary_search_algorithm#/media/File:Binary_Search_Depiction.svg

Índex

Algorisme cerca dicotòmica

```
// Indica si l'element x està en el vector v.
// Pre: v: vector d'enters ordenat creixentment; x: enter
// Post: retorna true ssi x es troba a v; false en cas contrari.
bool hi_es(const vector<int> &v, int x) {
  bool trobat = false;
  unsigned int i = 0, j = v.size() - 1;
  while (not trobat and i <= j) {
    // Inv: x no es troba en els subvectors v[0..i - 1] i v[j + 1..v.size() - 1]
    unsigned int pos = (i + j)/2;
    if (x > v[pos]) i = pos + 1;
    else if (x < v[pos]) j = pos - 1;
    else trobat = true;
  }
  return trobat;
}
```

Índex

Cost de la cerca dicotòmica

- Aquest algorisme redueix el temps de cerca considerablement, ja que disminueix exponencialment el nombre d'iteracions necessàries.
- En el pitjor dels casos el nombre màxim de comparacions és $\lfloor \log_2 n + 1 \rfloor$ on n és el nombre d'elements del vector.
- Per tant el cost d'aquest algorisme és de l'ordre de $\Theta(\log n)$.
- Per exemple, en un vector amb 10.000.000 elements, l'algorisme realitza com a màxim 24 comparacions.

Evolució del logaritme

x	$\log_2(x)$
1	0
10	4
100	7
1.000	10
10.000	14
100.000	17
1.000.000	20
10.000.000	24
100.000.000	27

Índex

Exercicis de repàs

1. Fes una **funció** que donat un vector d'enters ordenat *v* i un enter *x*, indiqui en quina posició del vector es troba *x*. En cas que l'enter no es trobi en el vector ha de tornar -1.
2. Fes una **funció** que donat un vector d'enters ordenat *v* i tres enters *x*, *esq* i *dret*, indiqui si *x* es troba en el subvector *v*[*esq*..*dret*]. Ha de retornar `true` si *x* es troba a *v*[*esq*..*dret*] o `false` en cas contrari.
3. Fes una **funció** que donat un vector d'enters ordenat *v* i tres enters *x*, *esq* i *dret*, implementi de manera recursiva la cerca dicotòmica de *x* en el subvector *v*[*esq*..*dret*]. Ha de retornar `true` si *x* es troba a *v*[*esq*..*dret*] o `false` en cas contrari.

Exercici 1

```
// Indica la posició en que es troba l'element x en el vector v. Si no hi és
// retorna -1.
// Pre: v: vector d'enters ordenat creixentment; x: enter
// Post: retorna la posició on es troba x a v; -1 si x no es troba a v.
int hi_es(const vector<int> &v, int x) {
  bool trobat = false;
  unsigned int i = 0, j = v.size() - 1;
  while (not trobat and i <= j) {
    // Inv: x no es troba en els subvectors v[0..i - 1] i v[j + 1..v.size() - 1]
    unsigned int pos = (i + j)/2;
    if (x > v[pos]) i = pos + 1;
    else if (x < v[pos]) j = pos - 1;
    else trobat = true;
  }
  int res = (i + j)/2 ;
  if (not trobat) res = -1;
  return res;
}
```

Exercici 2

```
// Indica si l'element x està en el subvector v[esq..dret].
// Pre: v: vector d'enters ordenat creixentment; x, esq, dret: enter;
//   0 ≤ esq, dret < v.size()
// Post: retorna la posició on es troba x a v; -1 si x no es troba a v.
int hi_es(const vector<int> &v, int x, int esq, int dret) {
  bool trobat = false;
  unsigned int i = esq, j = dret;
  while (not trobat and i <= j) {
    // Inv: x no es troba en els subvectors v[esq..i − 1] i v[j + 1..dret]
    unsigned int pos = (i + j)/2;
    if (x > v[pos]) i = pos + 1;
    else if (x < v[pos]) j = pos − 1;
    else trobat = true;
  }
  return trobat;
}
```


Exercici 3

```
// Indica si l'element x està en el subvector v[esq..dret].
// Pre: v: vector d'enters ordenat creixentment; x, esq, dret: enter;
//   −1 ≤ esq, dret ≤ v.size()
// Post: retorna true ssi x es troba a v; false en cas contrari.
bool hi_es(const vector<int> &v, int x,
           int esq, int dret) {
  bool res;
  int pos = (esq + dret)/2;
  if (esq > dret) res = false;
  else if (x == v[pos]) res = true;
  else if (x > v[pos]) res = hi_es(v, x, pos + 1, dret);
  else res = hi_es(v, x, esq, pos − 1);
  return res;
}
```

Reconeixement i llicència

Aquesta sessió ha utilitzat el següent material:

- **Material docent de l'assignatura PRO1 de la FIB**. `http://www.cs.upc.edu/~pro1/` fet per en *Jordi Cortadella*, *Ricard Gavaldà* i *Fernando Orejas*.
- **Binary search algorithm**. `https://en.wikipedia.org/wiki/Binary_search_algorithm` fet per col·laboradors del projecte Viquipèdia.

Tema 25

Consells d'estil

Fonaments de Programació

Consells d'estil

Bernardino Casas

bcasas@cs.upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Objectius de la sessió

- ✓ Ser conscient de les dificultats de programar bé.
- ✓ Conèixer els diferents consells d'estil.
- ✓ Aplicar els consells d'estil per obtenir millors programes.

Índex

Per què programar és complicat?

- Part de la dificultat rau en que tenim molts requeriments que complir.
- Els nostres programes han de ser:
  - Útils
  - Correctes
  - Eficients
  - Fàcils de comprendre, modificar i extendre
  - Barats
  - . . . i més.
- Els programes han de resoldre un problema d'un usuari.
- Els usuaris han de saber exactament el que fa un programa.

Índex

Noms
Expressions i instruccions
Comentaris
Com posar noms
Nombres màgics

Índex

Triar un bon nom

- Utilitzar **noms descriptius** per *CONSTANTS* i *Atributs*, i noms curts per variables locals (per ex: índex de taules).

```
? int indexElement;
? for (indexElement = 0; indexElement < nombreElements;
?             ++indexElement)
?   taulaValors[indexElement] = indexElement;

for (int i = 0; i < nelems; ++i) {
  taula[i] = i;
}
```


Usar sempre les mateixes convencions (I)

Constants els noms de les constants han d'estar tot en majúscules.
MAXELEMS, NUMCOLS, ...

Tipus, classes, tuples i enumerats han de començar en majúscula i tenir una lletra majúscula en cada nova paraula.

```
// classes i tuples
class UrlTable { ... }
struct UrlTableProperties { ... }

// typedefs
typedef vector< vector<int> > Matriu;

// enums
enum class UrlTableError { ... }
```

Usar sempre les mateixes convencions (II)

Variables locals han d'estar tot en mínúscules amb un '_' entre paraules.

```
? taulaAuxiliar, comptElems

taula_auxiliar, compt_elems
```

Atributs com les variables locals però han d'acabar amb el caràcter '_'.

```
class TableInfo {
  ...
private:
  string table_name_;   // OK - barrabaixa al final.
  static Pool<TableInfo>* pool_;   // OK.
};
```


Noms de les funcions

- **Utilitzar verbs actius per les funcions**, sovint seguits de substantius. Ha de començar amb minúscula i en canviar de paraula començar amb majúscula.

```
? element(), dibuixa(), ...

obteElement(), mostraTaula(), ...
```

- Els **noms de les funcions i accions** haurien d'ajudar a entendre quin és el seu propòsit. Per exemple, les funcions booleanes haurien de tenir un nom que ajudi a entendre que retornen un booleà.

```
? if (comprovaPrimer(n)) { ... }

if (esPrimer(n)) { ... }
```

Precisió amb els noms

- **Ser precís**. Un nom no només identifica sinó que també ofereix informació al programador que llegeix aquest codi. Un nom confús por provocar errors difícils de resoldre.

```
? bool esNombre (char c) {
?   return (c >= '0') and (c <= '8');
? }

bool esOctal (char c) {
  return (c >= '0') and (c <= '7');
}
```


Índex

Què són els nombres màgics?

- Els **nombres màgics** són les constants, mides de les taules, factors de conversió i d'altres valors numèrics literals que apareixen en els programes.
- Un nombre per si sol dins d'un programa no indica com d'important és, i fa que un algorisme sigui més difícil d'entendre i modificar. Per això cal donar-li nom als *nombres màgics*.

```
? for (int i = 0; i < 10; ++i)
?   taula[i] = 0;
```

```
const int MAXELEMS = 10;
const int BUIT = 0;
const int PLE = 1;

for (int i = 0; i < MAXELEMS; ++i)
  taula[i] = BUIT;
```


Enumerats

- Per declarar els *nombres màgics* d'una manera més abreujada es poden usar enumerats:

```
enum {
  MAXELEMS = 10,
  BUIT = 0,
  PLE = 1
};
```

- IMPORTANT: Només es pot usar un enumerat per englobar constants de tipus enter.

Índex

Índex

Bona indentació

- Indentar adequadament el codi per mostrar l'estructura.
- Utilitzar un mateix estil de sagnat i de posicionament de claus en tot el codi.

```
? for (n=0; n<100; t[n++]='\0');
? t[99] = '␣'; return('\n');

for (n = 0; n < 99; ++n) {
    t[n] = '\0';
}
t[99] = '␣';
return '\n';
```


Índex

Bon ús dels parèntesis

- **Utilitzar parèntesis per resoldre ambigüitats**. En alguns casos per augmentar la lectura del codi.

```
? any_traspas = a % 4 == 0 and a % 100 != 0 or
?               a % 400 == 0;

any_traspas = ((a%4 == 0) and (a%100 != 0)) or
              (a%400 == 0);
```


Índex

Únicament posar els comentaris necessaris

- **No repetir el que ja és obvi**.

```
? i = 0; // inicialitzem i a 0
? i++; /* incrementa la variable i */
```

- Tots els comentaris han d'afegir quelcom que no sigui evident en el codi.
- Cal comentar SEMPRE:
  - *les funcions*: explicació, pre i post
  - *els bucles*: invariant
  - *els atributs d'una classe*: explicar que emmagatzema l'atribut.

Com saber quan un comentari és necessari?

Regla d'or

Només comentar allò que seria una mica més complicat d'entendre en cas que un company de classe hagués de llegir el codi.

Comentaris consistents

- **Els comentaris no han de contradir el codi**. A mesura que passa el temps un codi es modifica corregint errors i ampliant-lo, evoluciona.
- Els comentaris de vegades es poden quedar en la versió original i per tant poden aparèixer discrepàncies entre els comentaris i el codi.
- **Els comentaris han d'aclarir no pas confondre**.

Els comentaris no milloren el codi dolent

- Quan es requereixi més d'unes quantes paraules per explicar el que està succeïnt és una senyal que potser cal reescriure el codi.

Reescriure no pas comentar

No comentar el codi dolent, en el seu lloc reescriure'l.

Reconeixement i llicència

Aquesta sessió ha utilitzat el següent material:

- **La práctica de la programación**. Llibre de l'editorial *Pearson educación* fet per *Brian W. Kernighan* i *Rob Pike*.
- **Google C++ Style Guide**. `https://google.github.io/styleguide/cppguide.html`