

Laboratori 1. Pràctica 1

Assignatura: Estructura de Dades
Grau d'Enginyeria Informàtica
Facultat de Matemàtiques i Informàtica,
Universitat de Barcelona

Funcionament de les pràctiques

- Treball individualment
- Lliuraments setmanals
 - Part d'una pràctica o la totalitat de la pràctica (depenent de la data)
- Format de l'entrega (mirar instruccions al final de cada enunciat)
- Cada exercici en un projecte NetBeans 8.2 diferent i codificat amb C++11, amb el nom “exerciciN”.
 - Agrupeu tots els exercicis en una carpeta amb el nom i cognoms de l'alumne
- Si s'usen fitxers externs, incloure'ls als recursos del projecte (add existing item)
- Si un exercici usa classes d'un exercici anterior, fer-ne una còpia al nou exercici.

Material per començar

Consultes de C++:

- <http://www.tutorialspoint.com/cplusplus/>
- <http://wwwcplusplus.com/reference/>

Tutorials bàsics (no ho tenen tot):

- <http://www4.tecnun.es/asignaturas/Informat1/AyudaInf/aprendainf/cp/p/basico/cppbasico.pdf>
- <http://www4.tecnun.es/asignaturas/Informat1/AyudaInf/aprendainf/cp/avanzado/cppavan.pdf>

A la biblioteca teniu disponibles per lloguer els llibres referenciats a la presentació del curs

Exercici 1

- Heu de crear un projecte nou, anomenat **Exercici1**
 - Al crear el projecte, ja es genera un `main.cpp`
- L'exercici 1 només requereix un programa **main.cpp**
- Aquest exercici serveix per practicar amb `includes`, `namespace`, `cin`, `cout`, `if...`
- En aquesta pràctica ens caldran alguns tipus de dades de C++
 - `string`,
 - `int`
 - `char`
 - `float`
 - `bool`

Exercici 2

- Heu de crear un projecte nou, anomenat **Exercici2**
 - Al crear el projecte, ja es genera un **main.cpp**
- L'exercici 2 només requereix un programa **main.cpp**
- Serveix per practicar amb:
 - **vector** i les **seves** built-in functions
 - **switch**
 - també practiqueu com definir funcions i passar paràmetres

Exercici 3

- Heu de crear un projecte nou, anomenat **Exercici3**
 - Al crear el projecte, ja es genera un `main.cpp`
 - Afegiu un nou fitxer. Per fer-ho: aneu a l'arrel del projecte, botó dret → new → C++ class. Poseu el nom i us crearà el `circle.cpp` i `circle.h`
- En aquest exercici necessiteu crear una classe **Circle** i controlar excepcions.
- NetBeans ens ajuda a definir classes (ja crea **fitxer .h** amb constructors i **fitxer .cpp**)
 - Al **fitxer .h** s'inclou la definició de la classe amb els atributs i la signatura dels mètodes
 - Al **fitxer .cpp** s'implementen els mètodes de la classe

Exercici 3

- **Exemple .h**

```
Class Exemple{
```

```
    private:
```

```
        int a;
```

```
        int edat;
```

```
    public:
```

```
        Exemple();
```

```
        int getEdat();
```

```
        void calculaVolum(int,int,int);
```

```
};
```

- **Exemple .cpp**

```
Exemple::Exemple(){
```

```
    a=0;
```

```
    edat=0;
```

```
}
```

```
int Exemple::getEdat(){
```

```
    return this->edat;
```

```
}
```

Els atributs solen ser privats

No cal posar els noms de variables

Davant del mètode nomClasse::

Els atributs privats amb this->

Exercici 3: Excepcions

- Una excepció és un fet extraordinari que pot provocar que el programa s'aturi. És un error.
- El programador pot controlar els fets extraordinaris i activar una excepció per a que el programa no s'aturi.
 - Crear excepcions i controlar-les fa que els programes siguin més clars, més robustos i més tolerants a fallades.
- Les excepcions es creen dins el mètode corresponent, i es comuniquen a qui l'hagi cridat amb un **“throw”**
- El programa que crida un mètode que pot provocar excepció, fa un intent (**try**) i recull la possible excepció (**catch**).

Esquelet del codi d'una excepció

- en el mètode:

```
if (condicio_errònia) {  
    throw 10;  
}
```

Si funciona bé
segueix als ...

Poden haver-hi molts
blocs catch per
diferents tipus
d'excepcions

- a la crida:

```
try {  
  
    metode (parametres) :  
    ...  
} catch (int e) {  
    cout << "Problemes amb ..." << endl;  
}
```

Si es produeix aquest
tipus d'excepció
s'executa aquest codi

Funcionament del try and catch

```
void Func3()  
{  
  
    try  
{  
  
        Func4();  
  
    }  
    catch ( ErrType )  
{  
  
    }  
}
```

Crida
mètode

Retorn
normal

```
void Func4()  
{
```

```
    if ( error )  
        throw ErrType();
```

Retorn amb
excepció

Classe exception

- **Exception** és la classe base estandard de C++ per a totes les excepcions
 - Per usar-la hem de fer:
#include <stdexcept>
- Proveeix a les classes derivades amb un mètode **what**, que permet retornar un missatge associat a l'excepció
- Més informació i exemples a:
 - <http://www.cplusplus.com/doc/tutorial/exceptions/>
 - https://www.tutorialspoint.com/cplusplus/cpp_exceptions_handling.htm

Llibreria estàndard d'exceptions

- Excepcions lògiques: Class `logic_error`
 - Errors a la lògica del programa (es poden veure al codi)
 - Classes derivades:
 - `invalid_argument` - l'argument enviat al mètode és invàlid
 - `length_error` - hem sobrepassat la mesura permesa
 - `out_of_range` - el valor no està dins el rang permès
- Excepcions d'execució: Class `runtime_error`
 - Errors que es detecten en temps d'execució (no es veuen al codi)
 - Classes derivades:
 - `overflow_error` - overflow aritmètic
 - `underflow_error` - underflow aritmètic

Llibreria estàndard d'excepcions

- Altres excepcions: deriven directament d'`exception`
 - Excepcions llençades per les característiques del llenguatge de C++
 - `new - bad_alloc` – no s'ha pogut assignar memòria
 - `dynamic_cast - bad_cast` – no s'ha pogut fer el canvi de tipus
 - `typeid - bad_typeid` – l'identificador de tipus no és correcte
 - Posa `std::bad_exception` a `throw` list
 - `unexpected()` farà `throw bad_exception` instead of calling function set by `set_unexpected`

Exercici 4

- En aquest exercici s'ha de definir la classe **Ellipse**
 - Igual que anteriorment, creeu el projecte Exercici4
 - Feu un new → C++ class per crear el .h i .cpp
- A més a més, l'exercici és una preparació per als exercicis posteriors
- Es repassen conceptes vistos als exercicis anteriors
- La lectura de l'input
 - No cal fer-la processant un string
 - Es pot fer amb múltiples cin:
`cin >> dada1 >> dada2 >> ... ;`

Exercici 5. Lectura de fitxers

```
#include <fstream>
...
int main() {
ifstream meu_fitxer("figures.txt");
char tipus;
float dada1,dada2;

while (!meu_fitxer.eof()) {
    meu_fitxer >> tipus;
    meu_fitxer >> dada1;
    meu_fitxer >> dada2;
    // el seu codi aquí
}
meu_fitxer.close();
return 0;
}
```

fstream és la classe dels streams de fitxer. Semblant a iostream

ifstream és un fitxer d'input. En el constructor cal indicar el nom del fitxer a disc.

eof (end of file) és un mètode que avisa si hem arribat al final del fitxer

IMPORTANT!
Sempre cal tancar
(com a Python i a Java)

Exercici 6

- En aquest exercici s'ha de definir l'herència publica de la classe **Circle**
- Aprendreu a:
 - Definir l'herencia, la classe mare serà **Ellipse** i la filla **Circle**
 - Definir constructors i destructors amb classes heretades i veure com és la sortida per pantalla

Exercici 6: Herència

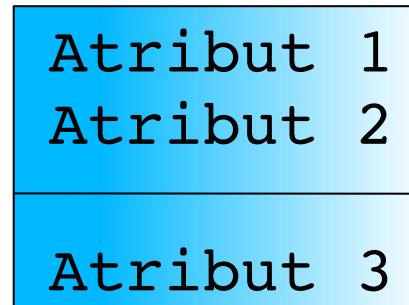
```
class className: memberAccessSpecifier baseClassName{  
    members list  
};
```

Classe Base



Atributs de
la classe base

Classe Derivada



(herència de la classe base)

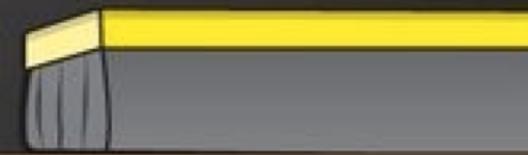
Atributs de la classe base
accessibles per l'herència

Atributs definits a la
classe derivada

Visibilitat amb herència pública

```
class circle: public shape {  
    . . .  
};
```

Visibilitat Classe Base	Visibilitat Classe Derivada	Accés derivada?	Accés extern
public	public	SI	SI
private	private	NO	NO
protected	protected	SI	NO



Exercici 6: Herència amb mètode abstracte

A la superclasse els mètodes només es defineixen.

Els inclourem al fitxer .h:

virtual indica que
és abstracte

```
virtual int elMeuMetode(int, int)=0;
```

En el fitxer .cpp de la superclasse el mètode no apareix.

A les subclasses els mètodes s'implementen i s'inclouen al fitxer .cpp

```
int NomSubClasse::elMeuMetode(int a, int b) {  
    ...  
}
```

En el fitxer .h de la classe el mètode es redefineix
(override) permetent el polimorfisme.

Definim la classe com:

```
class subExemple:public Exemple{  
};
```

Indiquem l'herència i visibilitat

Al fitxer .cpp en els constructors
apareixerà:

```
subExemple::subExemple () :Exemple () {  
}
```

En realitat el constructor està creant també
una instància de la superclasse

Exercici 6: algunes precaucions

- Recordeu que hi ha dues maneres de crear instàncies:
 - **Estàticament**: es guarden a l'stack
`Persona p1;`
 - **Dinàmicament**: es guarden en el heap
`Persona* p1 = new Persona();`

En els exercicis 6 i 7 recomanem la segona opció: **Crear instàncies dinàmicament.**

Exercici 6:

Constructors i destructors

- Tenen el mateix nom que el nom de la classe
- El constructor sempre es crida automàticament quan es crea un objecte. Pot ser de diversos tipus:
 - Constructor **per defecte** (sense paràmetres)
La seva definició i implementació són automàtiques en el moment de compilació
 - Constructors **amb paràmetres**. Serviran per donar valor als atributs.
 - Constructor **còpia** (amb un únic argument, una referència a un altre objecte de la mateixa classe)
- Una classe només pot tenir un destructor

Exercici 6: Implementació de constructors i destructors

- Si els atributs són dades estàtiques
 - El compilador crea un constructor còpia i un destructor de forma automàtica
- Si els atributs són dades dinàmiques
 - **Cal escriure el codi de la implementació del destructor i del constructor còpia**
- En tots dos casos
 - El compilador crea un constructor per defecte sempre

Exercici 6: Exemples

- Per crear una instància d'una classe, cridem el constructor:
 - De forma explícita
`Exemple e1 = Exemple(200,10);`
 - De forma implícita
`Exemple e1(200,10);`
- O a partir d'un altre objecte:
`Exemple e1 (200,10);`
`Exemple e2;`
`e2 = e1;`
`Exemple e3(e1);`
`Exemple e4 = Exemple(e1);`

Constructors amb paràmetres

Constructors còpia

Exercici 6: Constructor còpia

```
Exemple::Exemple(const Exemple& orig)
{
    atr1= orig.atr1;
    atr2 = orig.atr2;
}
```

- Per definició, el compilador també defineix un constructor còpia per defecte que copia bit a bit un objecte.
 - Si l'objecte té memòria dinàmica el constructor còpia per defecte pot donar resultats inesperats
 - En cas de tenir atributs amb memòria dinàmica, és molt millor implementar el constructor còpia

Exercici 7: Contenidor: ús del vector

- L'exercici 7 ha de crear la classe **EllipseContainer**
- **EllipseContainer** és un vector de **Ellipses**
 - **EllipseContainer.h**

```
#include <vector>
```

```
private:
```

```
    std::vector<Ellipse*> v;
```

- **EllipseContainer.cpp**

```
#include <vector>
```

```
vector<Ellipse*> vtemp;
```

```
this->v=vtemp
```

Exercici 7: Abstracció i destructors

- **Problema:** A EllipseContainer tenim "Ellipses"
 - Una Ellipse és una Ellipse o un Circle?
 - No podem cridar el destructor!
- **Solució:** assignació de subclasse dinàmica:
dynamic_cast

```
if (dynamic_cast<Circle*>(Ellipse[i]))  
    delete (Circle *)(Ellipse[i]);
```