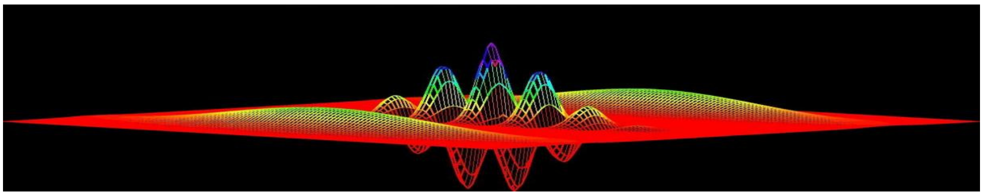
Computational Physics

numerical methods with C++ (and UNIX)



Fernando Barao
Instituto Superior Tecnico, Dep. Fisica
email: barao@lip.pt

Computational Physics

Classes and Objects

OOP programming

Fernando Barao, Phys Department IST (Lisbon)

Creating class objects

- ✔ Now that we understood the constructor role we can build objects and refer to the public available functions

locally

local object point

```
// make a point
point P(1.,2.);
P.Print(); //print point
P.X(); // look to x coo
P.Y(); // look to x coo
```

dynamically

local object point

```
// make a pointer to a new object
// constructor called
point *p = new point(1.,2.);
//print point (note the ->)
p->Print();
p->X(); //look to x coord
p->Y(); //look to y coord
```

class point

```
class point {

public:

 //methods publically visible

 point(double fx=0, fy=0):x(fx), y(fy){;} //constr
 point(const point& p):x(p.x),y(p.y){;} //copy constr
 point& operator=(const point& p); //assignment
 point& point::operator+=(const point& p); //+=
 point& point::operator-(const point& p); //-
 point point::operator+(const point& p); //+

 double X() const {return x;} // access the x coord
 double Y() const {return y;} // access the y coord
 void SetX (double); // set the x coord
 void SetY (double); // set the y coord
 void Print(); // print point

private:

  double x; //X coordinate
  double y; //Y coordinate

};
```

Removing the object : destructor

- ✔ The *destructor* of a class its the function called for releasing the memory that the class object allocated

point class destructor

```
class point {
 public:
  ~point(); //destructor
};
```

- ✔ if no destructor is defined in the class block, the compiler will invoke its own default destructor

  data is removed from memory in reversed order with respect to the order they appear in the class block

- ✔ the compiler default destructor is good enough for objects without data members pointers

  the default destructor would remove only the addresses variables and not the pointed objects !

C++ Classes : an example

Class header (IST.h)

```
#ifndef __IST__
#define __IST__
class IST {
public:
 IST(); // constructor
 ~IST() {;} //destructor
 void SetName(string); // set name
 string GetName() {return name;} // accessor
private:
  string name; float mark;
};
#endif
```

Class implementation (IST.C)

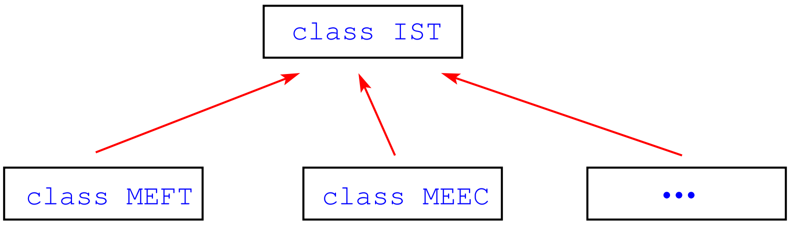
```
#include "IST.h"
IST::IST() { ////// default constructor
  name ="";
  mark=0.0;
}
void IST::SetName(string fname) {
  name = fname;
}
```

using class (test.C)

```
#include "IST.h" //class header

int main() {
  // mem allocated
  IST* pIST = new IST();
  pIST->SetName("Joao N.");
  pIST->SetMark(15.5);

  // vector of pointer objects
  vector<IST*> vIST;
  vIST.push_back(new IST("JJ",15,5));

  //free memory
  delete pIST;
  delete vIST[0];
}
```

C++ classes inheritance

- ✔ **MEFT** and **MEEC** are *derived classes* of the *base class* **IST**
- ✔ Derived classes inherit all the accessible members of the base class



- ✔ The inheritance relationship of two classes is declared in the derived class

```
  class MEFT : public IST {
    public:
            ... //public members
    private:
            ... //private members
};
```

- ✔ The keyword **public** specifies the most accessible level for the members inherited from the base class - **all inherited members keep their levels**

  the members of the derived class can access the protected members inherited from the base class but not its private members (invisible members)

C++ classes inheritance (cont.)

- With the keyword **protected**, all *public members of the base class* are inherited as *protected in the derived class*
- the **private** keyword will not give access to the base class members from the derived class

```
class MEFT : protected IST {...};
class MEFT : private IST {...};
```

- If no access level is specified for the inheritance, the compiler assumes *private* for classes declared with keyword *class* and *public* for those declared as *struct*
- A derived class (public access keyword) inherits every member of a base class except :
  - ➜ its constructors and destructor
  - ➜ its assignment operator members (=)
  - ➜ its friends
  - ➜ its private members
- Nevertheless, the derived class constructor call the default constructor of the base class (the one without arguments) which must exist
  - ➜ calling a different constructor is possible :

```
Derived_Construtor(parameters) : Base_Constructor(parameters) {...};
```

class inheritance : virtual functions

- *Virtual functions* can be declared in a base class with the keyword *virtual* and may be redefined (overriden) in each derived class when necessary
- *Virtual functions* will have the **same name and same set of argument types in both base class and derived class**, but they will perform different actions

```
class Base {
  public:
   //virtual function declaration
   virtual void Function(double);
};

class Derived: public Base {
  public:
   //objects Derived will use this function
   void Function (double);
};
```

class inheritance : abstract classes

✔ A virtual function declared in a base class can eventually stay undefined due to lack of information - it will be called a *pure virtual function*

pure virtual function

```
class Base {
  public:
   //pure virtual function
   virtual void Function(double) = 0;
};
```

✔ A class with one or more pure virtual functions is called an *abstract class*

✔ **No objects of an abstract class can be created**

✔ A pure virtual function that is not defined in a derived class remains a pure virtual function and the derived class is also an abstract class

C++ classes inheritance (cont.)

Base class header (IST.h)

```
#ifndef __IST__
#define __IST__
class IST {
public:
 IST(); // NEEDED default constructor
 IST(string, float); // constructor
 ~IST() {;} //destructor
 void SetName(string);
 string GetName();
 virtual void SetBranch(string)=0;
protected:
  string name;
  float mark;
};
#endif
```

Derived class header (MEFT.h)

```
#ifndef __MEFT__
#define __MEFT__
class MEFT : public IST {
public:
 MEFT(string, float, string); //constr
 ~MEFT() {;} //destructor
 void SetBranch(string);
 string GetBranch();
protected:
  string branch; //curso
};
#endif
```

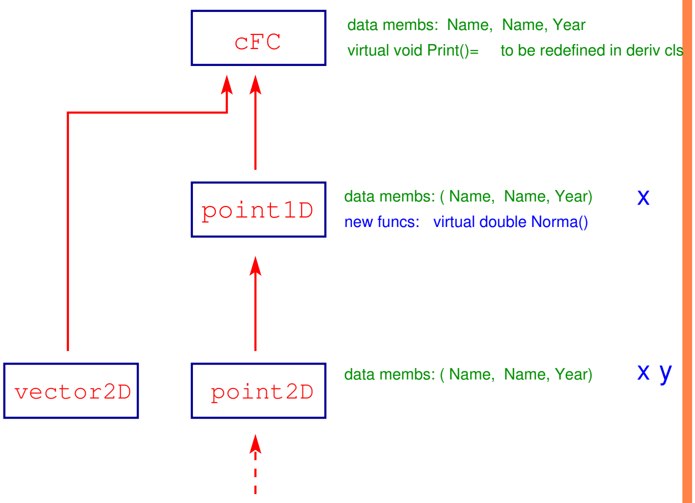
Base class code (IST.C)

```
#include "IST.h"
IST::IST(string fname, float fmark) : name(fname), mark(fmark) {;} // ... code
```

Derived class code (MEFT.C)

```
#include "MEFT.h"
MEFT::MEFT(string fname, float fmark) : IST(fname, fmark) {;}
void MEFT::SetBranch(string fbranch) {branch = fbranch;} // ... code
```

An inheritance scheme for Fis Comp



cFC

data membs: Name, Name, Year

virtual void Print()= to be redefined in deriv cls

point1D

data membs: (Name, Name, Year) x

new funcs: virtual double Norma()

vector2D

point2D

data membs: (Name, Name, Year) x y

- ✔ Let's define a base class that should define basic information common to all classes to be developed - **cFC class**
  - ➜ the group name *(string)*
  - ➜ the scholar year *(string)*
  - ➜ the class name *(string)*
  - ➜ virtual functions supposed to be redefined in derived classes
- ✔ The classes that derive from *cFC class* will inherit all members of base class and will :
  - ➜ provide replacements for virtual's funcs
  - ➜ add new data members
  - ➜ add new functions
- ✔ A derived class can be a base of another derived class

cFC class : header file

Class header (cFC.h)

```
#ifndef __cFC__
#define __cFC__
#include <string>
#include <iostream>
using namespace std;
class cFC {
 public:
  cFC() {groupName="; Year="; ClassName="; }
  cFC(string fg, string fy) : groupName(fg), Year(fy) {};
  string GetGroupName();
  string GetYear();
  void PrintGroupId();
  virtual void Print() = 0; //generic print to be implemented in every derived class
  void SetClassName(string fc) {ClassName = fc;}
  string GetClassName() {return ClassName;}
  void PrintClassName() {cout << ``Class Name = `` << ClassName << endl;}
 private:
  string groupName;
  string Year;
  string ClassName; //+...(nome do trabalho, ...)
};
#endif
```

cFC class : code

Class implementation (cFC.C)

```
#include <iostream>
using namespace std;
#include "cFC.h"

string cFC::GetGroupName() {
  return groupName;
}
string cFC::GetYear() {
  return Year;
}
void cFC::PrintGroupId() {
  cout << "group Name   = " << groupName << endl;
  cout << "Scholar year = " << Year << endl;
}
```

point1D class : header file

Let’s define a class to manipulate one-dimensional points : **Class header (point1D.h)**

```
#ifndef __point1D__
#define __point1D__
#include "cFC.h"
#include "point1D.h"

  class point1D : public cFC { // 1D points

  public:
    point1D(double fx=0.) : cFC("A01","2014-15"), x(fx) {
      SetClassName("point1D"); } // default constructor (inlined)
    void move(double); //move to new position
    void move(point1D); //move to new position
    void Print(); //print
    virtual double Norma(); //calculate modulo

  protected:
    double x; // x coordinate
  };
#endif
```

class : comments

cFC

- ✔ abstract class due to pure virtual function *Print()*
- ✔ reminder : abstract class cannot be instatiated by itself !
- ✔ the virtual function must be defined by the derived classes

point1D

- ✔ class has protected members *x*, which means visible to derived classes members
- ✔ constructor code is implemented inside header file
  - ➜ *inlined* constructor
  - ➜ shows that implementation can follow declaration
- ✔ There is *default constructor (constructor with no arguments)*
- ✔ destructor is not needed because there is no space allocated on *heap* by the class
- ✔ overloading of member functions *move()*

point1D class : code implementation

Class code (point1D.C)

```
#include <iostream>
using namespace std;
#include "point1D.h"

void point1D::move(double fx) {x=fx;}

void point1D::move(point1D p) {x=p.x;}

void point1D::Print() {
 PrintClassName();
 cout << ``[point1D] x='' << x << endl;
}

double point1D::Norma() { return x;}
```

point2D class

point2D.h

```
class point2D : public point1D {
  public:
   point2D(double fx, double fy) : point1D(fx), y(fy) {;}
   ...
  private:
   double y; // y coordinate
};
```

main program (main.C) YOU HAVE TO TRY IT!!!!

```
#include "point2D.h"
int main() {
   point2D a; // try this...! which constructor is being used?
   a.Dump();

   point2D b(0,0);  b.Dump();

   point2D c(5,2);
   b.move(c); //b=(5,2)
   b.Dump();
   double d = Norma(b);
}
```

point2D class (cont.)

- ✔ You are going to have a compiler error due to the fact you are trying to instantiate a *point2D* using the default constructor (NOT IMPLEMENTED !)
- ✔ Implementation of a default constructor

```
point2D() {x=0; y=0;}
```

- ✔ You can define a much more generic constructor that is a default constructor (no arguments needed) and also accepts arguments

```
point2D(double fx=0, double fy=0) : x(fx), y(fy) {;}
```

Example of use of the different constructors

```
point2D a; // (0,0)
point2D b(5); // (5,0)
point2D b(5,2); // (5,2)
```