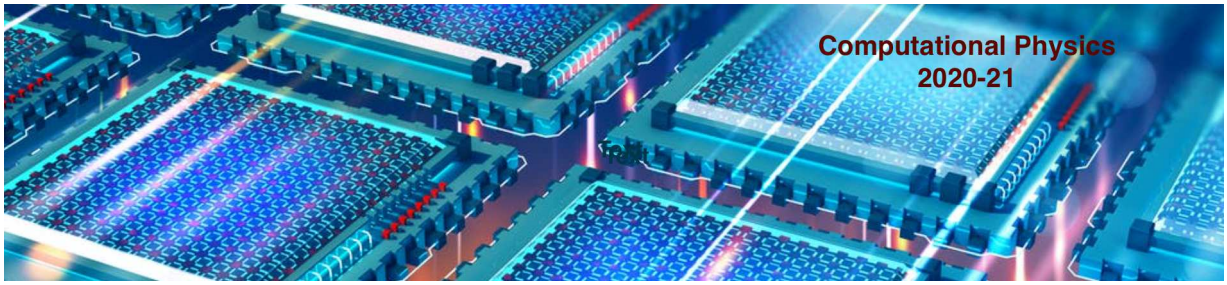
Computational Physics

numerical methods with C++ (and UNIX)

2020-21



Fernando Barao
Instituto Superior Tecnico, Dep. Fisica
email: fernando.barao@tecnico.ulisboa.pt

Creating class objects

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

locally

object point on stack

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

dynamically

object point on heap

```
// 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& operator+=(const point& p); //+=
 point& operator-(const point& p); //-
 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,y; //X, Y coordinate
};
```

```
1 point A; point B(A); //copy constructor
2 point A=B; //copy constructor
3 A=B; //A.operator=(B), assignment operator
4 A+=B; //A.operator+=(B),
5 A=A+B; //A.operator=(A.operator+(B))
6 point C = A+B; //A.operator+(B) && copy constr calle
7 point D = A-B; //A.operator-(B) && copy constr calle
```

Removing the object: destructor

- The *destructor* of a class it's 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 pointers as data members
  the default destructor would remove only the addresses variables and not the pointed objects!

C++ Classes: an example

Class header (IST.h)

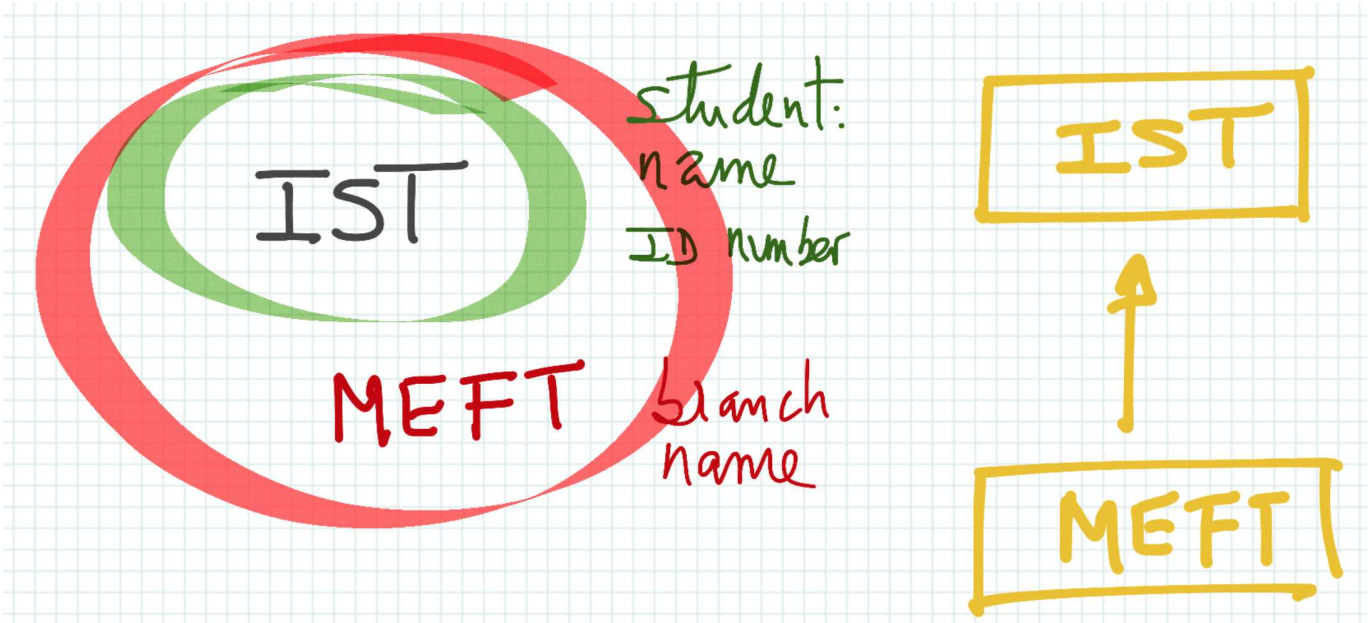
```
#ifndef __IST__
#define __IST__
class IST {
public:
 IST(); // constructor
 ~IST() {;} //destructor
 void SetName(string); // set name
 string GetName() { // accessor
   return name;
 }
private:
 string name; //nome aluno
 int ID; // ID number
};
#endif
```

Class implementation (IST.C)

```
#include "IST.h"
IST::IST() { ////// default constructor
  name= "";
  ID=-1;
}
void IST::SetName(string fname) {
  name = fname;
}
```

```
1  //class header
2  #include "IST.h"
3
4  int main() {
5
6    // allocate memory
7    IST* pIST = new IST();
8
9    // set object data
10   pIST->SetName("Joao N.");
11   pIST->SetID(96000);
12
13   // vector of object pointers
14   vector<IST*> vIST;
15   vIST.push_back(new IST("JJ",97000));
16   vIST.push_back(pIST);
17
18   //free memory
19   delete pIST;
20   delete vIST[0];
21   vIST.clear();
22 }
```

C++: scaling sets (classes)

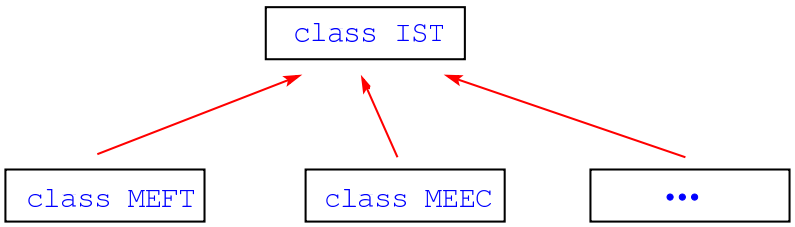


The *set* MEFT shall have all data members: those belonging to IST and to MEFT

C++ implements this through *inheritance* scheme that allows **class extension**

C++: class 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
};
```

- ✔ Reminder of class member permissions:
  public
  protected
  private

Access	public	protected	private
members of the same class	yes	yes	yes
members of derived class	yes	yes	no
not members	yes	no	no

see inheritance in cplusplus.com

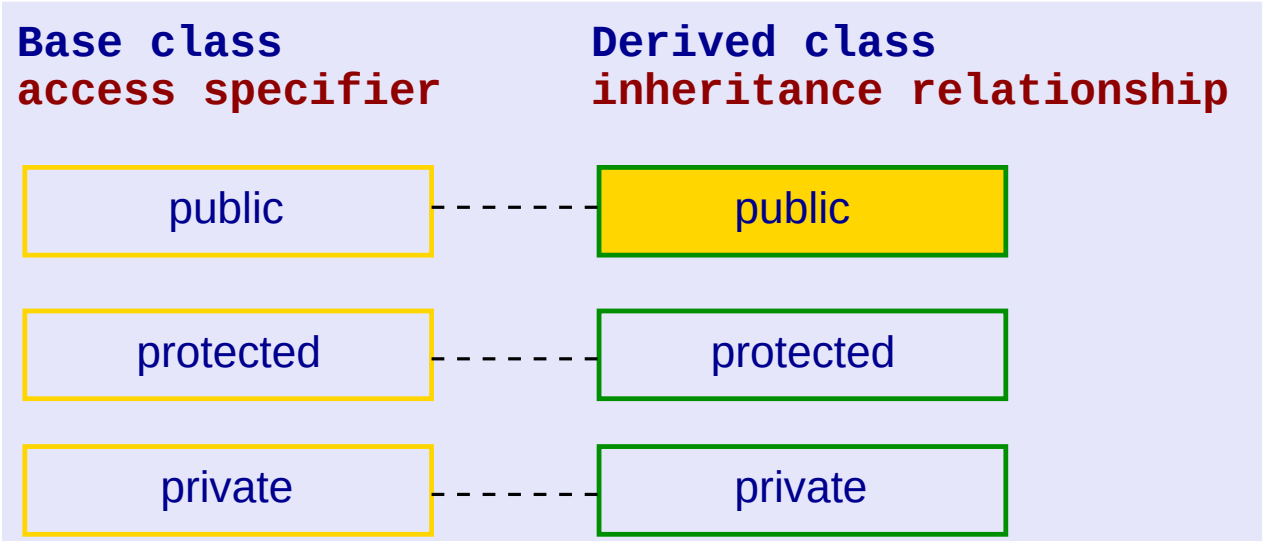
C++ classes inheritance: access control

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

derived class access to base class (inheritance relationship) declared as:

✔ public `class Derived: public Base /* ... */`

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

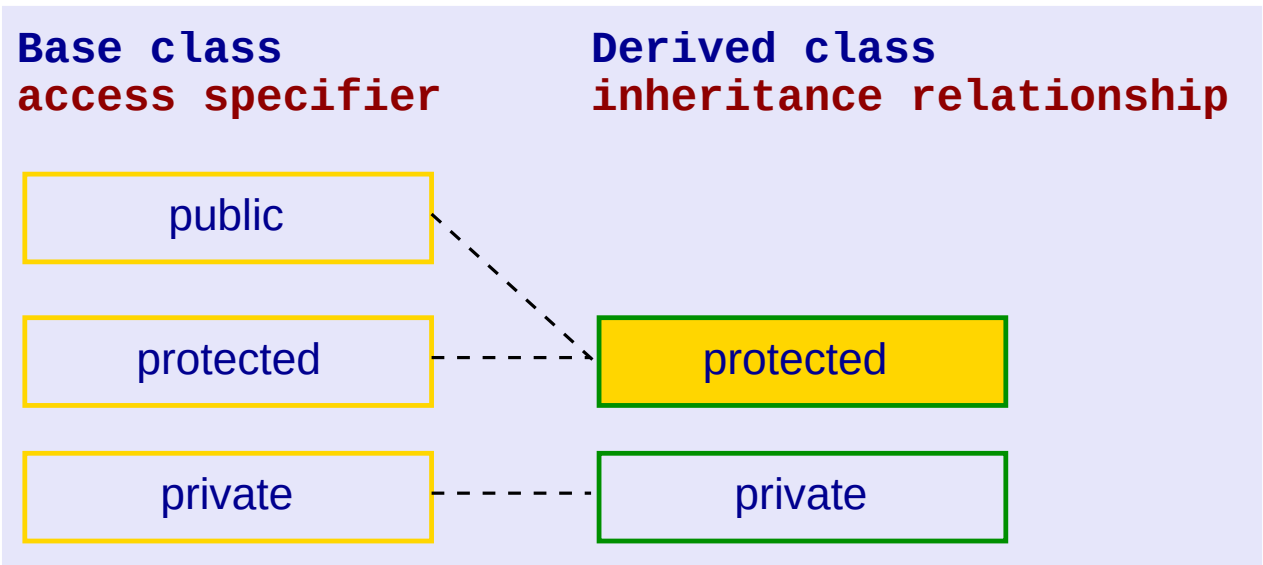


C++ classes inheritance: access control

derived class access to base class (inheritance relationship) declared as:

✔ protected `class Derived: protected Base /* ... */`

public and protected members of the **base class** become protected members of the **derived class**

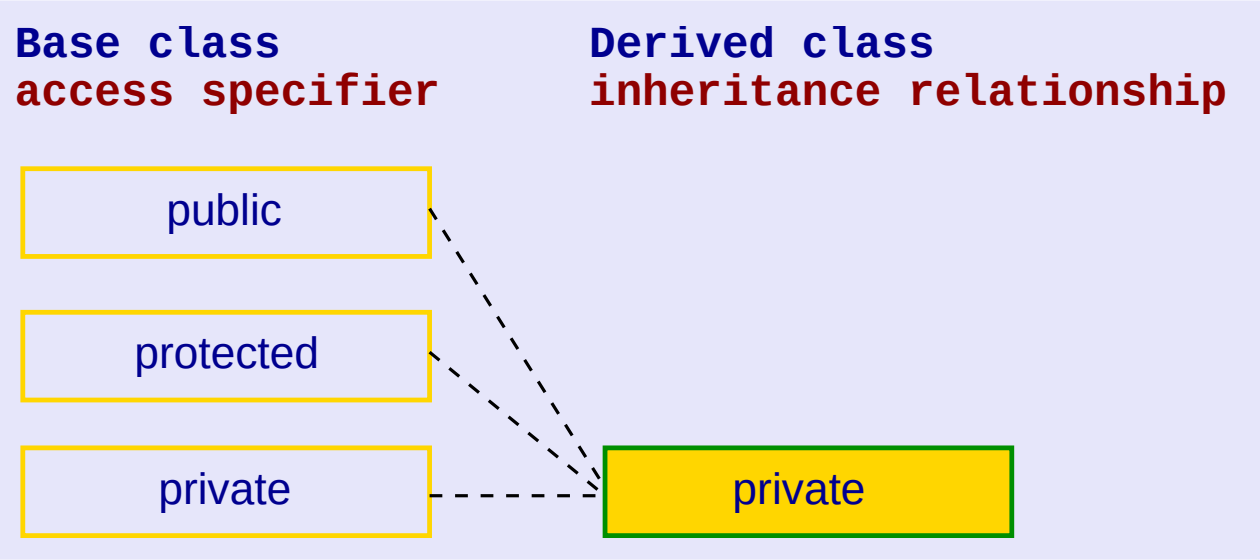


C++ classes inheritance: access control

derived class access to base class (inheritance relationship) declared as:

- ✔ private `class Derived: private Base /* ... */`

  public and protected members of the **base class** become private members of the **derived class**



C++ classes inheritance (cont.)

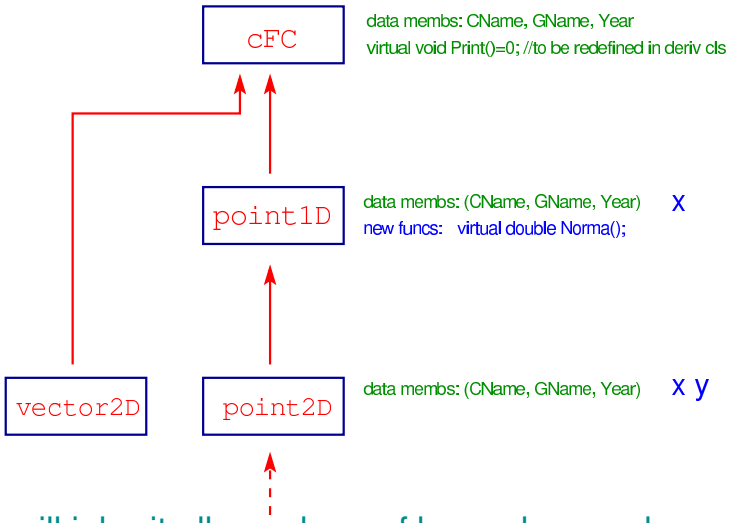
- ✔ If no access level is specified for the inheritance, the compiler assumes *private* for classes declared with keyword *class* and *public* for those declared as *struct*
- ✔ A derived class with **public access** inherits every member of a base class except:
  - ➜ its constructors and destructor
  - ➜ 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)
  - ➜ calling a different constructor is possible in the initializer list,

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

- ✔ the destructor of the derived class, has to free all the memory allocated by the derived class (not the base class)

An inheritance scheme for Fis Comp

- 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

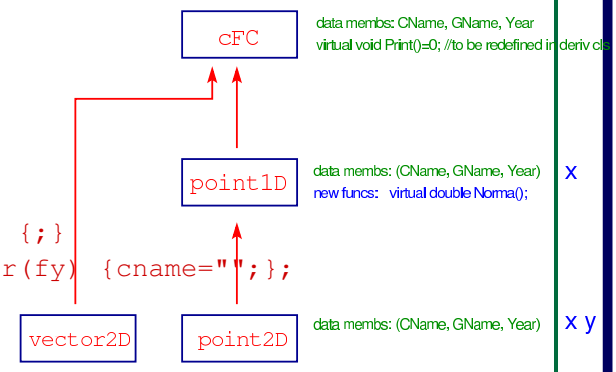


- The classes that derive from *cFC class* will inherit all members of base class and will:
  - provide replacements for virtual's funcs
  - add new data members
  - add new functions
- A derived class can be a base of another derived class (see previous slides about inheritance relationship)

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) {cname="";};
  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

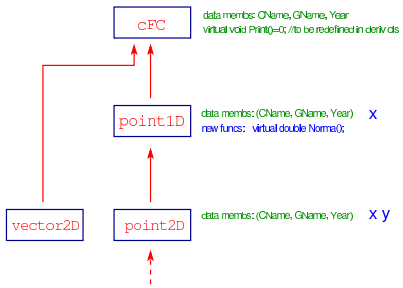
Let's define a class to manipulate one-dimensional points: **Class header (point1D.h)**

```
#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 instantiated 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.)

- ✔ 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)
```

class vector2D

Let's make a class **vector2D** making use of the class **point2D** before defined; it will include two point2D data members dynamically allocated that will require the user to define copy's constructor and assignment

a possible class definition with two points (vector2D.h)

```
class vector2D : public cFC {
  public:
   vector2D(point2D pf, point2D pi) : cFC("A01", "2014-15"), Pf(pf), Pi(pi) {
     cout << "point2D constructor/1" << endl;}
   vector2D(point2D pf) : cFC("A01", "2014-15"), Pf(pf), Pi() {
     cout << "point2D constructor/2" << endl;}
  private:
   point2D Pi; //initial point
   point2D Pf; //final
};
```

class definition with a point2D pointer (vector2D.h)

```
class vector2D {
  public:
   vector2D(point2D pf, point2D pi);
   vector2D(point2D pf);
  private:
   point2D *P; //pointer
};
```

class implementation (vector2D.C)

```
vector2D::vector2D(point2D p2, point2D p1){
  P = new point2D[2];
  P[0] = p1; P[1] = p2; }
vector2D::vector2D(point2D pf) {
  P = new point2D[2];
  P[0] = point2D(); //default constructor
  P[1] = pf; }
```


class vector2D (cont.)

vector2D class: copy and assignment constructor declarations (vector2D.h)

```
class vector2D {
  public:
   vector2D(const vector2D&); //copy constructor
   vector2D& operator=(const vector2D&); //copy assignment
   ...
};
```

vector2D class: copy and assignment constructor implementation (vector2D.C)

```
   vector2D::vector2D(const vector2D& t) {//copy constructor
    P = new point2D[2]; // array with two points created
    P[0] = t.P[0];
    P[1] = t.P[1];
   }
   vector2D& vector2D::operator=(const vector2D& t) {//copy assignment
    if (this != &t) { //this is a const pointer to current object (member func invoked)
       P[0] = t.P[0];
       P[1] = t.P[1];
    }
    return *this;
   }
```