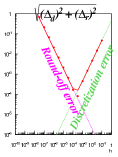
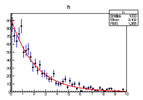
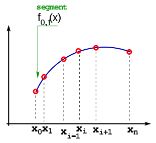
Computational Physics

numerical methods with C++ (and UNIX)

2018-19







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

C++ STL library (cont.)

✔ **pair**

This class couples together a pair of values, which may be of different types. The individual values can be accessed through its public members **first** and **second**.

```
// pair::operator= example
#include <utility>      // std::pair, std::make_pair
#include <string>       // std::string
#include <iostream>     // std::cout

int main () {
  std::pair <std::string,int> planet, homeplanet;
  planet = std::make_pair("Earth",6371);
  homeplanet = planet; // = operator working!
  std::cout << "Home planet: " << homeplanet.first << '\n';
  std::cout << "Planet size: " << homeplanet.second << '\n';

  // vector of pairs
  vector<pair<int,int>> vpair;
  vpair.push_back(std::make_pair(1,2));
  vpair.push_back(std::make_pair(3,4));
  return 0;
}
```

C++ STL library (cont.)

✔ **list**

```cpp
#include <iostream> // cout
#include <list> // list
using namespace std; // namespace

int main() {
   list<int> L;
   L.push_back(1);          // Insert a 1 integer at the end
   // [1]
   L.push_front(2);         // Insert a 2 integer at the beginning
   // [2 1]
   L.insert(++L.begin(),0); // Insert 0 before position of first argument
   // [2 0 1]

   L.push_back(5); // [2 0 1 5]
   L.push_back(6); // [2 0 1 5 6]

   list<int>::iterator i; // define iterator
   for (i=L.begin(); i != L.end(); ++i) cout << *i << " ";
   cout << endl;
}
```

C++ STL library (cont.)

✔ **map container**

Maps are associative containers that store elements formed by a combination of a key value and a mapped value, following a specific order.
In a map, the key values are generally used to sort and uniquely identify the elements, while the mapped values store the content associated to this key.

✔ In the example we use a key *string* that names the engineering branch (MEFT, MEEC,...) and a vector of data structures containing students data

C++ STL library (cont.)

✔ map container (cont.)

```
#include <string>
#include <iostream>
#include <map>
#include <vector>
#include <utility>
using namespace std;

struct IST {
  string name; // nome
  float mark; // nota
};

int main() {
  map<string, vector<IST> > M;
  vector<IST> vMEFT, vMEEC;
  M["MEFT"] = vMEFT;
  M["MEEC"] = vMEEC;

  // fill vector structures
  IST A;
```

```
A.name = "John Lob";
A.mark = 15.5;
M.find("MEFT")->second.push_back(A);
A.name = "Tiago Num";
A.mark = 17.0;
M.find("MEFT")->second.push_back(A);

cout << "vector size="
     << vMEFT.size() << endl; // = 0
cout << "MEFT vector size=''
     << M.find("MEFT")->second.size()
     << endl; // = 2

//list map contents
map< string,vector<IST> >::iterator it;
for( it=M.begin(); it!=M.end(); ++it) {
  cout << it->first << ‘‘: ‘‘
       << it->second.size() << endl;
}
// retrieve vector MEFT
vector<IST> meft=M.find("MEFT")->second;
```

C++ STL library (cont.)

✔ stack

```
// stack::push/pop
#include <iostream>       // std::cout
#include <stack>          // std::stack

int main () {
  std::stack<int> mystack;

  for (int i=0; i<5; ++i) mystack.push(i);

  std::cout << "Popping out elements..." << std::flush;
  while (!mystack.empty())  {
     std::cout << ' ' << mystack.top(); //points to last element of stack
     mystack.pop(); //removes element on top of stack
  }
  std::cout << '\n';

  return 0;
}

Output:
Popping out elements... 4 3 2 1 0
```

C++ const declaration

- ✔ The *const* declaration allows to avoid further changes on variables or pointers
- ✔ *const* variables shall be initialized when declared
- ✔ constant value

```
int const MyVariable = 0; //const applies to the left declaration (int)
const int MyVariable = 0; //do the same (nothing on left => right declaration)
MyVariable = 10; //compiler error, value cannot be changed
```

```
int const * pMyVar1 = NULL; // ERROR, because not initialized

int i = 10;                 // GOOD
int const * pMyVar1 = &i;
```

- ✔ constant pointer

```
int i=10, j=10;
int* const pMyVar2 = &i; //const pointer to variable i
pMyVar2 = &j; // can it be done????? (ERROR)
```

C++ const correctness (cont.)

- ✔ constant pointer to constant value

```
     int i = 10;
     int const * const q = &i;
```

It will not be possible to change the address and the value pointed to!

- ✔ constant functions

concept applied to member functions (in classes) - the function will be applied to an object that shall not be modified!

```
class T {                        // ********* Implementation
  public:                        void T::bar() const {
    ...                            i=100; //ERROR, the object cannot be changed!
    void bar() const;             }
  private:
  int i;
};
```

C++ const correctness (cont.)

- ✔ constant references

  we want to pass an object as argument of a function in an optimized (light) way => by reference

  we want to avoid any modifications of my object!

```
class T {
  public:
    ...
    void bar(const T&) const;
  private:
  int i;
};
```

Computational Physics

Classes and Objects

OOP programming

Fernando Barao, Phys Department IST (Lisbon)

C++ Classes and Objects

- ✔ In Object Oriented Programming (OOP) a group is a **class**, a class member is an **object** and a member function implements an **operation**
- ✔ Classes in OOP can be as simple as the set of numbers *int*, *float*, ...
- ✔ The member functions also called **methods** accomplish a broad range of tasks
  - ➜ constructors: default and parametered constructor
  - ➜ accessor member methods: query the objects
  - ➜ mutator member methods: operate and change the object
- ✔ Class members can be **public**, **private** or **protected**
  - ➜ public members can be accessed from the user program or user functions
  - ➜ private members can only be accessed from class members
  - ➜ protected: see inheritance

C++ Classes and Objects (cont.)

- ✔ A member of a class is **private** by default
- ✔ Particular member functions are used to:
  - ➜ create and initialize objects - **constructors**
  - ➜ destroy objects - **destructors**
- ✔ The class declaration needs a semi-colon (**;**) at the end
- ✔ There can be functions, called **friends**, which are not members of the class but have access to private members of the class

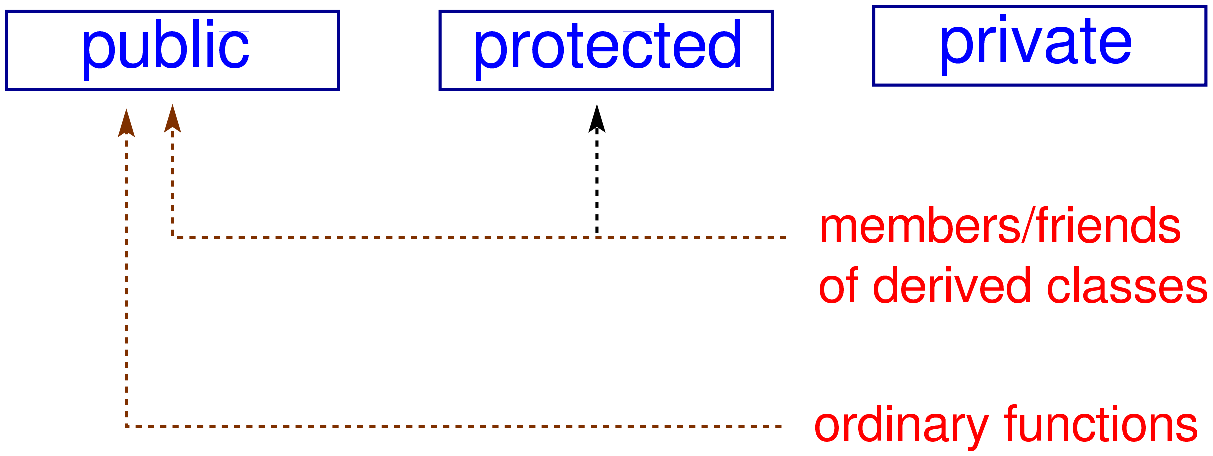
  friend functions can be declared on the private or public sector of the class

```
friend double function();
```

- ✔ Member functions **inline** need to be defined (coded) inside a class declaration (why? compiler needs to know it...cannot be in a library!)
- ✔ The **struct** data type in C++, is a class with all members **public**

accessing members

kind of members in a class



OOP programming

- ✔ A very simple class defining an object *point*
- ✔ the *point class* contains two data fields of type *double*: *x* and *y* to store the x and y coordinates of the point object
- ✔ **This is not Object Oriented Programming!** In OOP we would like the user to think about the **point** as an object, never dealing directly with its data members!

point class

```
class point {
public:
  double x; //X coordinate
  double y; //Y coordinate
};
```

main.C

```
  point P;
  P.x = 10.;
  P.y = 2.;
```

- ✔ The class shall have methods to access the data members (now private)

point class

```
class point {
public:
 double X() const {return x;} // method to access the value of the x coordinate
 double Y() const {return y;} // method to access the value of the Y coordinate
private: //could not be explicitely written (by default they are private)
  double x; //X coordinate
  double y; //Y coordinate
};
```

- ✔ *const* declaration implies that a compilation error arises if there is a trial to change the point object being called