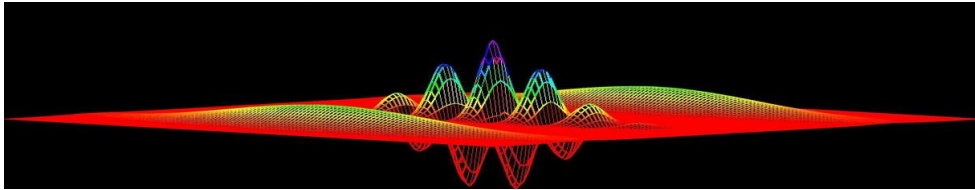


Computational Physics

numerical methods with C++ (and UNIX)



Fernando Barao

Instituto Superior Tecnico, Dep. Fisica

email: barao@lip.pt

C++ Input / Output

- ✓ The *iostream* library allow us to enter data from keyboard and display data on monitor

```
1 #include <iostream>
2 using namespace std;
3
4 ...
5 // read several real values from the keyboard
6 float a, b, ...;
7 cin >> a >> b >> ...;
8
9 // read a string from keyboard (no blank spaces)
10 string s;
11 cin >> s;
12
13 // read a full line (including blank spaces)
14 string s;
15 getline(cin, s);
16
17 // output line
18 cout << s << endl;
19 cout << s << "\n"; //similar to previous line
```

C++ Input / Output (cont.)

- ✓ The *fstream* library allow us read from and write to files

```
1 // read from file
2
3 #include <fstream>
4 using namespace std;
5
6 ...
7 // declare input file stream and open "filename.dat" file
8 ifstream F;
9 F.open("filename.dat"); //shortly could be: ifstream F("filename.dat");
10
11 // read file values
12 int i=0;
13 double a[10];
14 while (F>>a[i] && i<10) { // logical true if reading OK
15     cout << i << " " << a[i] << endl;
16     i++;
17 }
18
19 F.close(); // close file
```

C++ Input / Output (cont.)

- ✓ The *fstream* library allow us read from and write to files

```
1 // write to file
2
3 #include <fstream>
4 using namespace std;
5
6 ...
7 // declare output file stream and open "filename.dat" file
8 ofstream F("filename.dat");
9
10 // output values read before to file
11 int i=0;
12 double a[10];
13 while (i<10) { // logical true if reading OK
14     cout << i << " " << a[i] << endl;
15     F << a[i];
16     i++;
17 }
18
19 F.close(); // close file
```

C++ Input / Output (cont.)

- ✓ The *fstream* library allow us read from and write to files

```
1 // read and write to file
2
3 #include <fstream>
4 using namespace std;
5
6 ...
7 // declare output file stream and open "filename.dat" file
8 fstream F("filename.dat", ios::in | ios::out | ios::app); //app=if file
    exists write at end
9
10 // output values read before to file
11 int i=0;
12 double a[10];
13 while (i<10) { // logical true if reading OK
14     cout << i << " " << a[i] << endl;
15     F << a[i];
16     i++;
17 }
18
19 F.close(); // close file
```

C++ output formatting

- ✓ Formatted output can be done using the C-style *cstdio* library

```
1 printf("formatted output: integer=%d float=%f float=%12.3f\n",a,b,c); //
    try \n\la
```

- ✓ The input/output *iomanip* library allow us to print data in formatted way
- ✓ The width of the decimal part (including the decimal point) is given by *setprecision(n)* and total width is given by *setw(n)*

```
1 #include <iostream>
2 #include <iomanip>
3 using namespace std;
4 ...
5 double pi = 3.14159265358;
6 cout << setprecision(7) << setw(10) << pi << endl;
```

The number 3.141592 would be printed !

C++ output formatting (cont.)

```
1 #include <iostream>
2 #include <iomanip>
3 using namespace std;
4 #include <cmath> // M_PI
5 #include <cstdio>
6 int main() {
7     printf('1) %28.26f\n', M_PI);
8     cout << '2) ' << M_PI << endl;
9     cout << '3) ' << setprecision(27) << M_PI << endl;
10    cout << '4) ' << setiosflags(ios::scientific) << M_PI << endl;
11    cout << setiosflags(ios::scientific) << setprecision(5);
12    cout << '5) ' << M_PI << endl;
13
14    cout << resetiosflags(ios::scientific);
15    cout << setprecision(15) << setiosflags(ios::fixed | ios::showpoint) <<
        endl;
16    for (int i=0; i<4; i++) {
17        cout << i << ' ' << sin(M_PI/((double)((i+1)))) << endl;
18    }
19 }
```

1) 3.14159265358979311599796347
2) 3.14159
3) 3.14159265358979311599796347
4) 3.141592653589793115997963469e+00
5) 3.14159e+00

0 0.0000000000000000
1 1.0000000000000000
2 0.866025403784439
3 0.707106781186547

C++ dynamic memory allocation

- ✓ In a C++ program memory can be allocated dynamically at running time through the **new** operator and is responsibility of the user to delete it through the **delete** operator (otherwise remain there through all the program execution !)
- ✓ Memory is allocated by using the **new** operator followed by a data type and it returns a pointer to the first element of the sequence

```
1 float *f = new float; // memory allocated for 1 float
2 *f = 2.354; // value set
3
4 float *fv = new float[10]; // memory allocated for 10 floats
5 fv[0] = 2.345; // 1st element set
6 *(fv+1) = 3.245; // 2nd element
```

- ✓ To free memory the operator **delete** is used followed by the pointer to the object

```
1 delete f; // memory is freed
2
3 delete[] fv; // the destructors are called for every object pointer
```

- ✓ To obtain in linux, information about memory occupation in MBytes
> **free -m**

C++ dynamic memory allocation (cont.)

- ✓ An exception of type *bad_alloc* is thrown when the memory allocation fails
- ✓ The simplest way of controlling if the memory was properly allocated is to avoid the *Exception* to occur and check if a null pointer is returned

```
1 // allocated memory for 10 floats
2 float *fv = new (nothrow) float[10];
3 if (fv != nullptr) { // check for null pointer
4     fv[0] = 2.345; // 1st element set
5     *(fv+1) = 3.245; // 2nd element
6 } else {
7     exit(1);
8 }
```

C++ STL library

- ✓ The C++ *STL (Standard Template Library)* is a powerful set of C++ template classes to provides general-purpose templated classes and functions that implement many popular and commonly used algorithms and data structures like *vectors, lists, queues, and stacks*
- ✓ **vector container**
similar to an array but can be dinamically enlarged or shrinked

```
1 #include <iostream>
2 #include <vector>
3 #include <algorithm> // sort vector
4 using namespace std;
5
6 int main() {
7     vector<float> vec; // create a vector to store integers
8
9     // push 5 random values between 0 and 1 into the vector
10    for (int i = 0; i < 5; i++) {
11        float f = rand() / (float)RAND_MAX;
12        vec.push_back(f);
13    }
14    cout << "vector size=" << vec.size() << endl; // vector size
```

C++ STL library (cont.)

✓ vector container (cont.)

```
1 // add 5 vector values
2 float sum = 0;
3 for(int i = 0; i < 5; i++){
4     sum += vec[i]; // vec.at(i) could also be used
5 }
6
7 // use iterator to access the values
8 vector<int>::iterator vecit = vec.begin();
9 while( vecit != vec.end()) {
10     cout << "value =" << *vecit << endl;
11     vecit++;
12 }
13
14 // sort a vector contents and another way of inserting vector values
15 int myints[] = {32,71,12,45,26,80,53,33};
16 vector<int> v(myints, myints+8); // 32 71 12 45 26 80 53 33
17 sort(v.begin(), v.begin()+4); //(12 32 45 71)26 80 53 33
18 float max = *( max_element( v.begin(), v.end() ) ); //iterator
19
20 // clear vector
21 vec.clear(); v.clear();
```

C++ vector (cont.)

```
// an empty vector of integers
vector<int> v;
vector<int> v1(5); // a vector with 5 elements, each an integer

// An array of 5 empty vector<int> elements
vector<int> va[5];

// A vector with 5 elements each having the value 15
vector<int> v2(5, 15);

// A vector with the size and values of v2
vector<int> v3(v2);

// A vector with the size and values of v2
vector<int> v4(v2.begin(), v2.end());

// Create a vector from an array
int a[] = {1,2,3,4,5,6};
vector<int> v5(&a[0], &a[0]+4); //store 4 values
vector<int> v5; v5.assign(a, a+4); //or
```

C++ vector (cont.)

```
// An empty vector of vectors.
// The space appearing between the 2 end greater signs is mandatory
vector<vector<int> > v2d;

// If you intend creating many vectors
typedef vector<vector<int> > vecM;
vecM matrix;

// Create a 2 x 5 matrix
// ...First, create a vector row vector (5 elem)
vector<int> vr(5, 15);
// ...Now create a vector of 2 elements with each element a copy of v2
vector<vector<int> > vm(2,vr);

// Print out the elements
for(int i=0;i<vm.size(); i++) { //loop on rows
    for (int j=0;j<vm[i].size(); j++) { // loop on every row elem
        cout << vm[i][j] << " ";
    }
    cout << endl;
}

//clean
vm.clear();
}
```

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.)

```
1 #include <string>
2 #include <iostream>
3 #include <map>
4 #include <vector>
5 #include <utility>
6 using namespace std;
7
8 struct IST {
9     string name; // nome
10    float mark; // nota
11 };
12
13 int main() {
14     map<string, vector<IST> > M;
15     vector<IST> vMEFT, vMEEC;
16     malunos["MEFT"] = vMEFT;
17     malunos["MEEC"] = vMEEC;
18
19     // fill vector structures
20     IST A;
```

```
1 A.name = "John Lob";
2 A.mark = 15.5;
3 M.find("MEFT")->second.push_back(A);
4 A.name = "Tiago Num";
5 A.mark = 17.0;
6 M.find("MEFT")->second.push_back(A);
7
8 cout << "vector size="
9      << vMEFT.size() << endl; // = 0
10 cout << "MEFT vector size="
11      << M.find("MEFT")->second.size()
12      << endl; // = 2
13
14 // list map contents
15 map< string, vector<IST> >::iterator it;
16 for( it=M.begin(); it!=M.end(); ++it) {
17     cout << it->first << ": "
18          << it->second.size() << endl;
19 }
20 // retrieve vector MEFT
21 vector<IST> meft=M.find("MEFT")->second;
```