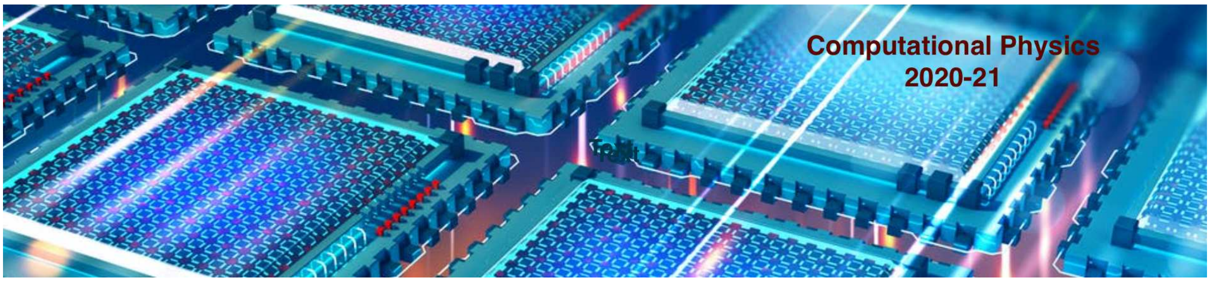
Computational Physics

numerical methods with C++ (and UNIX)

2020-21



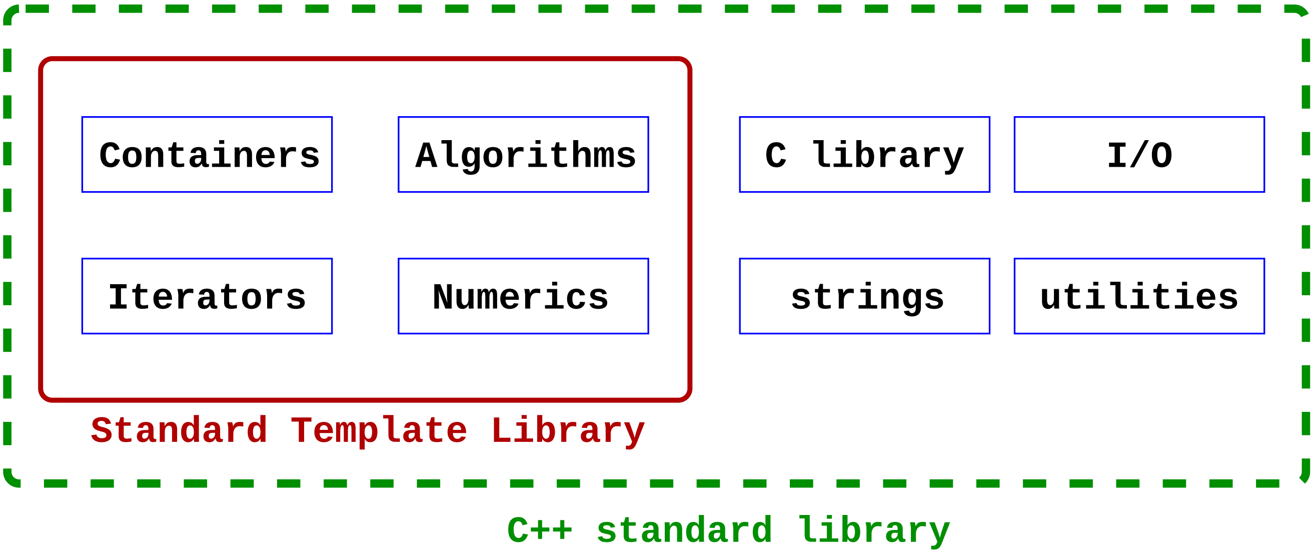
Fernando Barao
Instituto Superior Tecnico, Dep. Fisica
email: fernando.barao@tecnico.ulisboa.pt

C++ standard library

- a library is a collection of software components that can be used to build your software components (programs, functions, classes)
- the strength of a modern computer language is intrinsically related to the existence of a rich set of accompanying libraries
- the *C++ standard library* comes with the official version of the language
- all elements of the C++ standard library are declared on *header* files

```
#include <header file name>
```

C++ standard library



Containers

Algorithms

Iterators

Numerics

Standard Template Library

C library

I/O

strings

utilities

C++ standard library

C++ standard library: I/O headers

Header	Purpose	Examples
<cstdio>	C-style I/O	*printf()*
<cstdlib>	Conversion between numbers and C-strings	*atoi()*
<cwchar>	Multibyte character functions	
<fstream>	I/O class to operate on files	*ifstream*
<iomanip>	Manipulators with arguments	*std::setprecision(5)*
<ios>	I/O stream base classes, manipulators with no arguments, format flags, failure bits, open modes	*eof()*
<iosfwd>	forward declarations for the types of the standard input/output library	
<iostream>	basic output stream, output formatting	*cin, cout*
<istream>	Input stream objects can read and interpret input from sequences of characters	*getline()*

C++ standard library: Gen Utilities

Header	Purpose	Examples
<ctime>	System time and date functions	*time()*
<functional>	Function objects to be used on algorithms	*greater()*
<memory>	Allocators, raw memory, and autopointers	
<utility>	Generic relational operators, pair data structure	*swap(), pair*

C++ standard library: strings

Header	Purpose	Examples
<cctype>	set of functions to classify and transform individual characters	*isalnum(), isdigit()*
<cstdlib>	convert numbers to strings, random numbers, memory allocation	*malloc(), rand(), atoi()*
<cstring>	C-strings (null-terminated strings)	
<string>	C++ string classes and functions	
<cwchar>		
<cwctype>		

C++ timing

- ✔ The header file *<ctime>* defines a number of library functions which can be used to assess how much CPU time a C++ program consumes during execution
- ✔ A call to the function *clock()* will return the amount of CPU time used so far
- ✔ To normalize the time to seconds the returned number shall be divided by the variable *CLOCKS_PER_SEC*, defined inside *<ctime>*
- ✔ Next example computes time per operation in microseconds spent in calculating x^4, in a direct way and through the *pow()* function

C++ timing (cont.)

```
#include <ctime> // clock()
#include <cmath> // pow()
#include <iostream> // cout
using namespace std;
#define N 1000000

int main() {
  double a=12345678967598.0, b; //variable declaration

  //compute time spent on power to the fourth the double
  clock_t time1 = clock();
  for (int i=0; i<N; i++) b=a*a*a*a;
  clock_t time2 = clock();
  double dtime1 = (double)(time2-time1)/(double)CLOCKS_PER_SEC;

  //...using pow
  clock_t time1 = clock();
  for (int i=0; i<N; i++) b=pow(a,4.);
  clock_t time2 = clock();
  double dtime2 = (double)(time2-time1)/(double)CLOCKS_PER_SEC;

  cout << dtime1 << " | " << dtime2 << endl;
  return 0;
}
```

C++ random numbers

- Some calculations require the use of random numbers like the Monte-Carlo calculations
- The system header file *<cstdlib>* provides the function *rand()* that returns a random integer (fairly good approximation) in the range *[0, RAND_MAX]*
- The sequence seed can be fixed through a call to *srand(int)* rendering therefore the random sequences repeatable
  by default, rand() is seeded with the value 1
- To generate independent sequences a common practice is to use the current UNIX time (number of seconds elapsed since January 1st, 1970)
  time(NULL) returns an integer

```
// to see the manual page of time() function do in your OS linux / macOS
$ man 3 time
// to see all included functions in header ctime
$ man 3 ctime
// to see rand() and srand() infos
$ man 3 rand
```

- The next example produces a sequence of 10^5 values between 0 and 1

C++ random numbers (cont.)

```
#include <ctime> // time()
#include <cstdlib> // rand()
#include <iostream> // cout
using namespace std;

int main() {
  //set random seed
  srand(time(NULL));

  //generate random values and compute mean and variance
  double sum=0.;
  double var=0.;
  for (int i=0; i<100000; i++) {
    double x = (double)rand()/(double)RAND_MAX;
    sum += x;
    var += (x-0.5)*(x-0.5);
  }
  double mean = sum/100000.;
  var /= 100000.;

  cout << mean << `` | `` << var << "(expected variance = 1/12) WHY???'' << endl;
  return 0;
}
```

C++ Input / Output

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

```
#include <iostream>
using namespace std;

  ...
  // read several real values from the keyboard
  float a, b, ...;
  cin >> a >> b >> ...;

  // read a string from keyboard (no blank spaces)
  string s;
  cin >> s;

  // read a full line (including blank spaces)
  string s;
  getline(cin, s);

  // output line
  cout << s << endl;
  cout << s << "\n"; //similar to previous line
```

C++ Input / Output (cont.)

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

```
// read from file

#include <fstream>
using namespace std;

  ...
  // declare input file stream and open "filename.dat"file
  ifstream F;
  F.open("filename.dat");    //shortly could be: ifstream F("filename.dat");

  // read file values
  int i=0;
  double a[10];
  while (F>>a[i] && i<10) {  // logical true if reading OK
    cout << i << " " << a[i] << endl;
    i++;
  }

  F.close(); // close file
```

C++ Input / Output (cont.)

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

```cpp
// write to file

#include <fstream>
using namespace std;

  ...
  // declare output file stream and open "filename.dat"file
  ofstream F("filename.dat");

  // output values were read before
  int i=0;
  double a[10];
  while (i<10) { // logical true if reading OK
    cout << i << " " << a[i] << endl;
    F << a[i];
    i++;
  }

  F.close(); // close file
```

C++ Input / Output (cont.)

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

```cpp
// read and write to file

#include <fstream>
using namespace std;

  ...
  // declare output file stream and open "filename.dat"file
  // app = append, if file exists write at end
  fstream F("filename.dat", ios::in | ios::out | ios::app);

  // output values were read before
  int i=0;
  double a[10];
  while (i<10) { // logical true if reading OK
    cout << i << " " << a[i] << endl;
    F << a[i];
    i++;
  }

  F.close(); // close file
```

C++ output formatting

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

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

✔ The input/ouput *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)*

```
#include <iostream>
#include <iomanip>
using namespace std;
  ...
  double pi = 3.14159265358;
  cout << setprecison(7) << setw(10) << pi << endl;
```

The number 3.141592 would be printed!

C++ output formatting (cont.)

```
#include <iostream>
#include <iomanip>
using namespace std;
#include <cmath> // M_PI
#include <cstdio>
int main() {
  printf("1) %28.26f\n",M_PI);
  cout << "2) " << M_PI << endl;
  cout << "3) " << setprecision(27) << M_PI << endl;
  cout << "4) " << setiosflags(ios::scientific) << M_PI << endl;
  cout << setiosflags(ios::scientific) << setprecision(5);
  cout << "5) " << M_PI << endl;

  cout << resetiosflags(ios::scientific);
  cout << setprecision(15) << setiosflags(ios::fixed | ios::showpoint) << endl;
  for (int i=0; i<4; i++) {
    cout << i << " " << sin(M_PI/(double)((i+1))) << endl;
  }
}
```

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

```
0 0.000000000000000
1 1.000000000000000
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 responsability 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 elemnt of the sequence

```cpp
float *f = new float; // memory allocated for 1 float
*f = 2.354; // value set

float *fv = new float[10]; // memory allocated for 10 floats
fv[0] = 2.345; //1st element set
*(fv+1) = 3.245; // 2nd element
```

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

```cpp
delete f; //memory is freed (or deallocated)

delete[] fv; // the destructors are called for every object
```

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

C++ dynamic memory alloc: exception

- 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

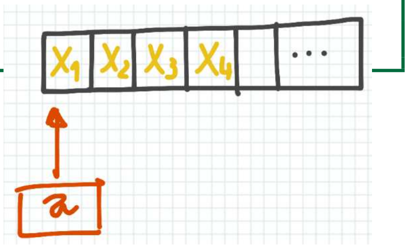
```cpp
#include <cstdlib> //exit()
#include <new> //std::nothrow
...
// allocated memory for 10 floats
float *fv = new (nothrow) float[10];
if (fv != NULL) { // check for null pointer
   fv[0] = 2.345; //1st element set
   *(fv+1) = 3.245; // 2nd element
   *(fv+2) = 2.46; // 3rd element
   ...
} else {
   exit(1);
}
```

C++ dynamic memory alloc examples

An array of 10 objects is allocated
The *delete []* operator will call the destructors of every object of the array

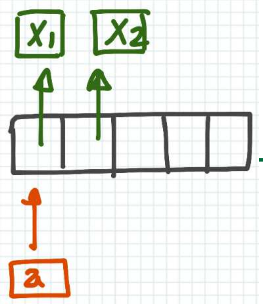
```
class A {
  public:
  A() {printf("%s ",
          __PRETTY_FUNCTION__);}
 ~A() {printf("%s ",
          __PRETTY_FUNCTION__);}
};

int main() {
 // create array of objects
 A *a = new A[10];
 // deallocate
 // object destructor is called
 delete [] a;
}
```



An array of 10 pointers to objects is allocated
The *delete []* operator **will not call** the destructors of every object of the array; do not forget what we store were pointers!

```
int main() {
// create array of pointers objects
A **a = new A*[10];
// create objects
for (int i=0; i<10; i++) {
 a[i] = new A();
}
// deallocate
// call destructor
for (int i=0; i<10; i++) {
 delete a[i];
}
delete [] a;
}
```



string class

- ✔ Strings are objects that represent sequences of characters.
- ✔ The standard string class provides support for such objects with an interface similar to that of a standard container of bytes, but adding features specifically designed to operate with strings of single-byte characters.

```
#include <iostream>
#include <string>
int main (){
  std::string str="We think in generalities, but we live in details.";
  std::string str2 = str.substr (3,5);  //"think"
  std::size_t pos = str.find("live"); // position of "live" in str
  std::string str3 = str.substr(pos);   // get from "live" to the end
  std::cout << str2 << ' ' << str3 << '\n';
  return 0;
}
```

C++ complex numbers

- ✔ complex numbers are implemented in C++ through the complex class

```
#include <complex> //C++ standard library
using namespace std;

int main() {
  complex<double> Z(2.5, 4.0);
  double Zmod = abs(Z);
  double Zr = Z.real();
  double Zi = Z.imag();
  complex<double> Zc = conj(Z);
}
```

- ✔ C++ example of using complex class: Tcomplex.C