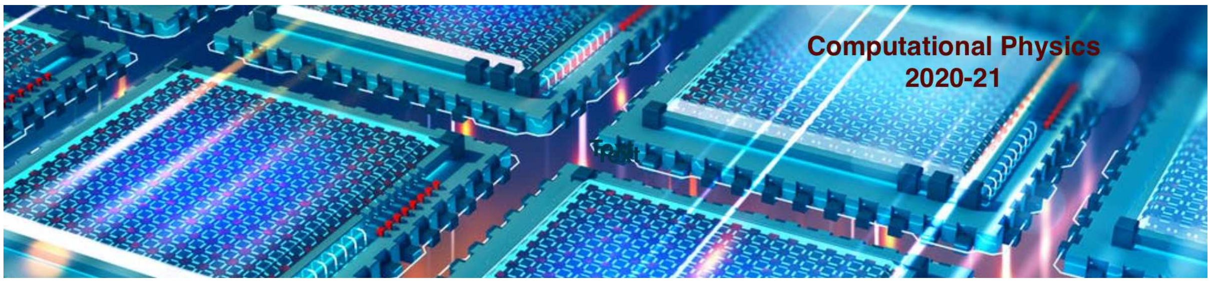
Computational Physics

numerical methods with C++ (and UNIX)

2020-21



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

Computational Physics

C++

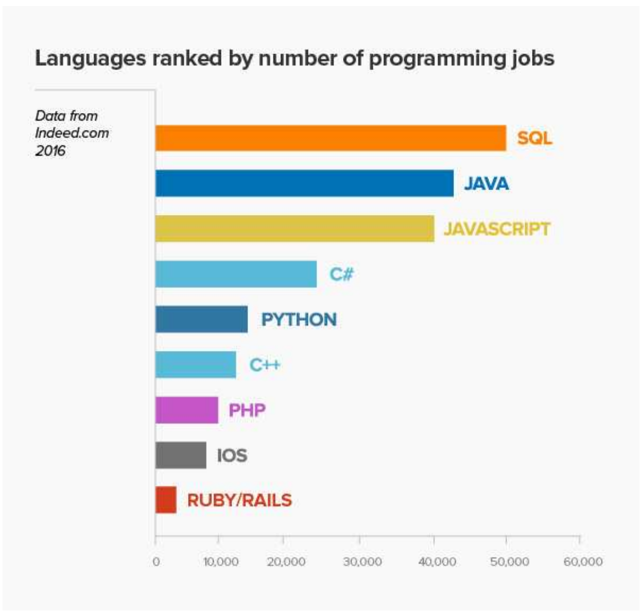
An object oriented language

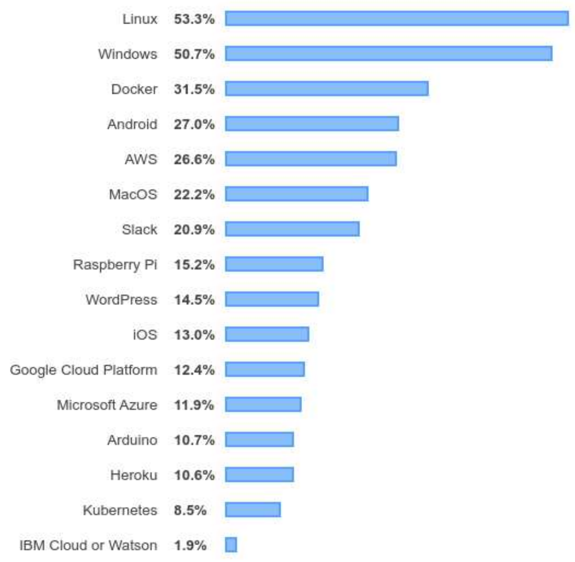
Fernando Barao, Phys Department IST (Lisbon)

C... Programming languages

- ✔ The **C language** was originally developed by computer scientists to write operating systems. It is considered a flexible and very powerful language. All UNIX operating systems are written in C. Although C is a high-level language, it incorporates many comparatively low-level features, as pointers.
- ✔ The **C++ language** is a major extension of C with the purpose of exploring the object-oriented programming. Object-oriented lamguages are well suited to large projects involving many people. But it requires some thinking about the problem before implementation...

Computer languages: stackoverflow survey





C++ general rules

- ✔ C++ is case sensitive
- ✔ A C++ statement may begin at any place in the line and can continue into the next line
- ✔ The end of the statement is indicated by a semicolon **;**
- ✔ There can be multiple statements in a line `int a=5; int b=10;`
- ✔ Comments to code can be inserted by using // `int a=5; //...`
- ✔ A large part of the code can be commented using /* ...*/
- ✔ The name of a variable must start with a letter and shal contain only letters, numbers and underscore _
- ✔ Every C++ program has a main function

```cpp
#include <iostream> // std::cout, std::endl

int main()  {
  int a = 5;
  std::cout << a << std::endl;
  return 0; //successful return (can be omitted)
}
```

C++ data types

- ✔ A variable has "allways"(unless C++11 **auto** declaration could be used!!!) to be declared such that the appropriate space can be reserved in memory by the compiler
- ✔ Once declared, a numerical variable can be initialized or evaluated

```cpp
 // integers
int a = 5;
int a; a=5;
int a(5);
unsigned int year; //positive integer
long a; // long integer (8 bytes)

 // characters (one byte)
char a = 66; //'B' (66 = int code)
char a = 'B'; //single quotes

 // constants
const int a = 5; //cannot be modified

 // reals
float b = -10.50; //single precision
float b = -1.05e+1;
double pi = 3.141592....; //double prec
```

```cpp
 // boolean vars
bool flag = true; //or false

 // strings (C++ standard lib, <string>)
string name = "alberto";
string name("alberto");

 // character strings (c-strings)
char word[20] = "four";
/* word[4]='\0' (null character)
the null character is automatically
added to the end of the character
string enclosed in double quotes */

 // convert string to c-string
const char* c = name.c_str(); //
      immutable
char *c = name.data(); //mutable
```

C++ data types (cont.)

Type	Description	Byte size
integers		
short int	short integer	2
short	ranges from -32768 to 32767	
signed short int	ranges from -32768 to 32767	
unsigned short int	ranges from 0 to 65535	
int	integer	4
signed int	ranges from -21474836487 to 2147483647	
unsigned int	ranges from 0 to 4294967295	
long int	long integer	8
long	long integer	
reals		
float	floating point number, single precision	4
double	floating point number, double precision	8
long double	floating point number, long double precision	16

C++ data types (cont.)

Type	Description	Byte size
others		
bool	boolean value, *true* or *false*	1
char	character	1
signed char	one byte integer from -128 to 127	
unsigned char	one byte integer from 0 to 255	

Type sizes and infos: C++ example

```
// compilation: g++ -std=c++11 <file>.C -o <file>.exe
#include <cstdio> // printf
#include <typeinfo> // typeid
#include <iostream> // std::cout (otherwise, using namespace std;)

int main() {
 // size of types
 std::cout << "unsigned int:\t" << sizeof(unsigned int) << " bytes" << std::endl;
 std::cout << "int:\t\t\t" << sizeof(int) << " bytes" << std::endl;
 std::cout << "short: \t\t'' << sizeof(short) << " bytes" << std::endl;
 std::cout << "long: \t\t'' << sizeof(long) << " bytes" << std::endl;
 std::cout << "long long: \t'' << sizeof(long long) << " bytes" << std::endl;
 std::cout << "float: \t\t\t'' << sizeof(float) << " bytes" << std::endl;
 std::cout << "double: \t\t\t'' << sizeof(double) << " bytes" << std::endl;
 std::cout << "long double: \t\t'' << sizeof(long double) << " bytes" << std::endl;
 std::cout << "int* (pointer): \t\t'' << sizeof(int*) << " bytes" << std::endl;
 std::cout << "long* (pointer): \t\t'' << sizeof(long*) << " bytes" << std::endl;
 // type info
 int a = 10;
 auto b = 10.0; // UHM!!! to be discussed later on...
 double* p = new double(1.); // pointer to double initialized to 1.0
 printf("type a=%s b=%s p=%s \n", typeid(a).name(), typeid(b).name(), typeid(p).name());
}
```

C++ data structures

✔ A data structure groups a set of characteristics of a given object (it is the prelude of a *class* in C++)

```
//string class
#include <string>
using namespace std;

// define structure
struct alunoIST {
  string name; // nome
  int idnumber; // num mecanográfico
  float mark; // nota
  // we can even add methods! (functions)
  int WeirdChar() {
    //check how many non alphanumeric chars
    const char *c = name.c_str();
    int n=0;
    for (int i=0; c[i]!='\0'; ++i) {
      if (!isalnum(c[i])) n++;
    }
    return n;
  };
};
```

```
#include <iostream>
int main() {
  alunoIST A;
  A.name = ``João Semfim'';
  A.mark = 20.0;
  A.idnumber = 88000;
  std::cout << A.WeirdChar() << std::
       endl;
}
```

C++ operators

arithmetic	
$+$	sum
$-$	subtraction
$*$	multiplication
$/$	division
%	modulo (remainder)

compound assignation	
$a+=b$	$a=a+b$
$a-=b$	$a=a-b$
$a*=b$	$a=a\times b$
$a/=b$	$a=a/b$
$a*=b+c$	$a=a\times(b+c)$
$a++$	$a=a+1$
$++a$	$a=a+1$
$a--$	$a=a-1$
$--a$	$a=a-1$

logical	
$a==b$	equal to
$a!=b$	not equal to
$a<b$	less than
$a<=b$	less than or equal to
$a>b$	greater than
$a>=b$	greater than or equal to
$a\&\&b$	AND
$a\|\|b$	OR
$!a$	boolean opposite

bitwise	
$<<\ >>$	left and right bit shift
&|	bit AND OR

others	
sizeof(a)	byte size

C++ operators (cont.)

✔ Arithmetic operators **(*) and (/)** have precedence over **(+) and (-)**

```
What C++ code to evaluate:
a + b/c +d
```

✔ Unary operators (only act on single operands) like **(++), (−−)** and signs (+), (-) have precedence over arithmetic operators

```
 What does this C++ code:
   int a,  b=5, c;
            // (a=0, b=5, c=0)
   b = a++; // b=? (after execution: b=0, a=1)
   c = ++a; // c=? (after execution: a=2, c=2)
```

✔ bit shift

```
#include<iostream>
int main() {
  short a = 1; // bit contents: 00000001
  a = a << 3; // Left shifting it by 3: 00001000, ie, 8
  std::cout << a << std::endl;
  a = a >> 2; // Right shifting a by 2: 00000010, ie, 2
  std::cout << a << std::endl;
  return 0;
  }
```

C++ control statements

```cpp
// if-else
if ( boolean expression1 ) {
  executed if expression1 is true;
} else if ( boolean expression2) {
  executed if expression2 is true;
} else {
  if none of above are true;
}

// while
double dx=1., eps=1.e-6;
while (dx > eps) {
  (...)
}

// do-while
do {
  (...)
} while (dx > eps);
```

```cpp
// for loops

/* pre-increment or post-increment has no impact
     on loop */

for (int i = 0; i < 100; i++) {
}
for (int i = 0; i < 100; ++i) {
}

/* here is a different way of making loop...
   - check of i<10 made before increment
   - increment counter after (post-increment) */

for (int i=0; i++ < 10; ) {// loop is made for i=1,...,10
  std::cout << i << std::endl;
}

/* increment is made before check of i<10  (pre-
     increment) */
for (int i=0; ++i < 10; ) {// loop is made for 1 ... 9
  std::cout << i << std::endl;
}
```