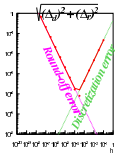
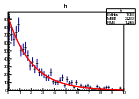
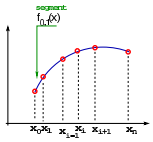
Computational Physics

numerical methods with C++ (and UNIX)

2018-19



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

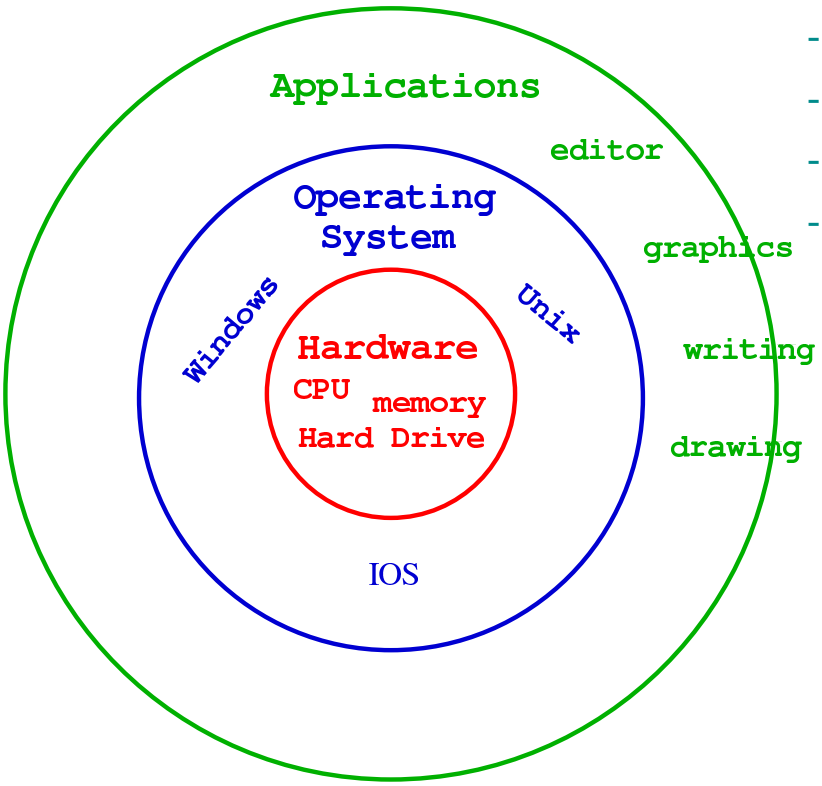
Computational Physics

Operating systems

UNIX (linux)

Fernando Barao, Phys Department IST (Lisbon)

Operating systems



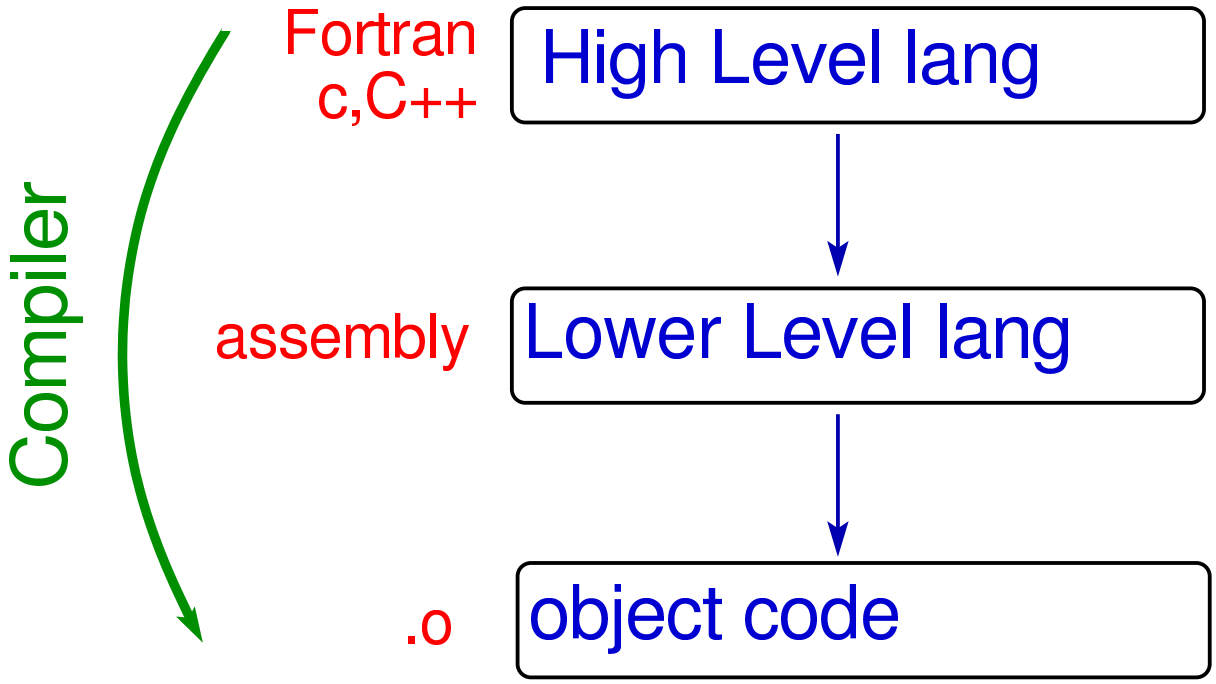
Many linux distributions:

- Debian
- Ubuntu (& derivatives)
- Linux Mint
- Fedora

Computer programming

- ✔ Symbolic languages use words (“add”, “move”, ...) instead of operation codes
- ✔ High-level symbolic languages:
  - FORTRAN FORmula TRANslator mid 1950’s
  - BASIC Beginner’s All-purpose Symbolic Instruction Code mid 1960’s
  - PASCAL early 1970’s
  - C mid 1970’s
  - C++, Java, ... mid 1980’s on
- ✔ C and C++ allow the manipulation of bits and bytes and memory addresses (some people tag it as mid-level languages)
- ✔ Other languages like Mathematica, Matlab or Maple: very rapid coding up but...code is interpreted (slower)
- ✔ The lowest level symbolic language is called the *assembly language*
- ✔ The **assembler** program translates the assembly into **machine code (object code)** that will be understood by the CPU

Computer programming



Creating an executable

- ✔ An executable file contains binary code encoding machine-language instructions
- ✔ To create it, we need to start by writing a program in a symbolic language, **the source code**
  - use some Unix editor like **pico, gedit, emacs**
- ✔ Next, we produce the object code, by compiling the source code and eventually linking with other pieces of code located in libraries or being compiled at the same time
  - compilers: **C++** → **g++**, **c** → **gcc**, **FORTRAN** → **gfortran**
  - the compiler assigns memory addresses to variables and translates arithmetic and logical operations into machine-language instructions
- ✔ The object code is loaded into the memory (RAM) and it is runned by the CPU (no further need of the compiler)
  - the object files are specific to every CPU and are not necessarily portable across different versions of the operating system

Computational Physics

C++

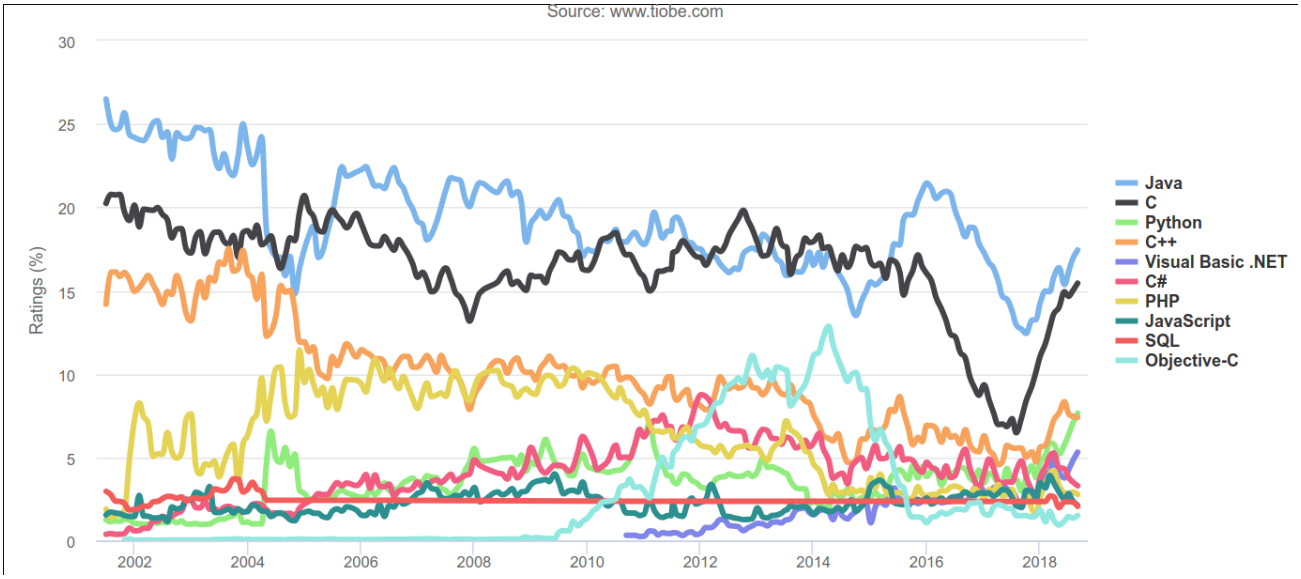
An object oriented language

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: a possible ranking...



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 staements 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

```
1 #define PRINT
2 #include <iostream>
3   int main()  {
4   int a = 5;
5   std::cout << a << std::endl;
6   return 0; //successful return (can be omitted)
7   }
```

C++ data types

- ✔ A variable has allways to be declared in order the appropriate space is reserved in memory by the compiler
- ✔ Once declared, a numerical variable can be initialized or evaluated

```
1   // integers
2   int a = 5;
3   int a; a=5;
4   int a(5);
5   unsigned int year; //positive integer
6
7   // characters
8   char a = 66; //'B' (66 = int code)
9   char a = 'B'; //single quotes
10
11  // constants
12  const int a = 5; //cannot be modified
13
14  // reals
15  float b = -10.50; //single precision
16  float b = -1.05e+1;
17  double pi = 3.141592....; //double prec
```

```
1   // boolean vars
2   bool flag = true; //or false
3
4   // strings (C++ std lib)
5   string name = "alberto";
6   string name("alberto");
7
8   // character strings
9   char word[20] = "four";
10  /* word[4]='\0' (null character)
11     the null character is automatically
12     added to the end of the character
13     string enclosed in double quotes */
```


C++ data types (cont.)

Type	Description	Byte size
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 -2147483648 to 2147483647	
unsigned int	ranges from 0 to 4294967295	
float	floating point number, single precision	4
double	floating point number, double precision	8
long double	floating point number, long double precision	12
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	

C++ data structures

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

```
1  #include <string>
2  using namespace std;
3
4  // define structure
5  struct alunoIST {
6    string name; // nome
7    float mark; // nota
8  };
9
10 int main() {
11   alunoIST A;
12   A.name = "Joao";
13   A.mark = 20.0;
14 }
```


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 shit
&|	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)
```


C++ control statements

```
1  // if-else
2  int a = 10;
3  if ( a < 5) {
4    true statement;
5  } else {
6    false statement;
7  }
8
9  // while
10 double dx=1., eps=1.e-6;
11 while (dx > eps) {
12   statements;
13 }
14
15 // do-while
16 do {
17 } while (dx > eps);
18
19 //for loop
20 for (int i=0; i< 10; i++) {
21    statements;
22 }
```