Computational Physics
Compiling a C++ program

Fernando Barao, Phys Department IST (Lisbon)
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

Compiler → Fortran, C, C++ → High Level lang → assembly → Lower Level lang → .o → object code
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)
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 languages are well suited to large projects involving many people. But it requires some thinking about the problem before implementation...
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

```cpp
int a=5; int b=10;
```

✔ Comments to code can be inserted by using //

```cpp
int a=5; //...
```

✔ A large part of the code can be commented using /* ...*/

✔ The name of a variable must start with a letter and shall contain only letters, numbers and underscore _

✔ Every C++ program has a main function

```cpp
#define PRINT
#include <iostream>
int main() {
int a = 5;
std::cout << a << std::endl;
return 0; //successful return (can be omitted)
}
```
A variable has always 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.

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

// characters
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

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

// strings (C++ std lib)
string name = "alberto";
string name("alberto");

// character 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 */
```
<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Byte size</th>
</tr>
</thead>
<tbody>
<tr>
<td>short int</td>
<td>short integer</td>
<td>2</td>
</tr>
<tr>
<td>short</td>
<td>ranges from -32768 to 32767</td>
<td></td>
</tr>
<tr>
<td>signed short int</td>
<td>ranges from -32768 to 32767</td>
<td></td>
</tr>
<tr>
<td>unsigned short int</td>
<td>ranges from 0 to 65535</td>
<td></td>
</tr>
<tr>
<td>int</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td>signed int</td>
<td>ranges from -2147483648 to 2147483647</td>
<td></td>
</tr>
<tr>
<td>unsigned int</td>
<td>ranges from 0 to 4294967295</td>
<td></td>
</tr>
<tr>
<td>float</td>
<td>floating point number, single precision</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>floating point number, double precision</td>
<td>8</td>
</tr>
<tr>
<td>long double</td>
<td>floating point number, long double precision</td>
<td>12</td>
</tr>
<tr>
<td>bool</td>
<td>boolean value, <em>true</em> or <em>false</em></td>
<td>1</td>
</tr>
<tr>
<td>char</td>
<td>character</td>
<td>1</td>
</tr>
<tr>
<td>signed char</td>
<td>one byte integer from -128 to 127</td>
<td></td>
</tr>
<tr>
<td>unsigned char</td>
<td>one byte integer from 0 to 255</td>
<td></td>
</tr>
</tbody>
</table>
A data structure groups a set of characteristics of a given object (it is the prelude of a \textit{class} in C++)

```cpp
#include <string>
using namespace std;

// define structure
struct alunoIST {
    string name; // nome
    float mark; // nota
};

int main() {
    alunoIST A;
    A.name = "Joao";
    A.mark = 20.0;
}
```
# C++ operators

<table>
<thead>
<tr>
<th>arithmetic</th>
<th>logical</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ sum</td>
<td>$a == b$ equal to</td>
</tr>
<tr>
<td>− subtraction</td>
<td>$a! = b$ not equal to</td>
</tr>
<tr>
<td>* multiplication</td>
<td>$a &lt; b$ less than</td>
</tr>
<tr>
<td>/ division</td>
<td>$a &lt;= b$ less than or equal to</td>
</tr>
<tr>
<td>% modulo (remainder)</td>
<td>$a &gt; b$ greater than</td>
</tr>
<tr>
<td></td>
<td>$a &gt;= b$ greater than or equal to</td>
</tr>
<tr>
<td>compound assignation</td>
<td></td>
</tr>
<tr>
<td>$a+ = b$ $a = a + b$</td>
<td></td>
</tr>
<tr>
<td>$a- = b$ $a = a - b$</td>
<td></td>
</tr>
<tr>
<td>$a* = b$ $a = a \times b$</td>
<td>$a&amp;&amp;b$ AND</td>
</tr>
<tr>
<td>$a/ = b$ $a = a/b$</td>
<td>$a</td>
</tr>
<tr>
<td>$a* = b + c$ $a = a \times (b + c)$</td>
<td></td>
</tr>
<tr>
<td>$a ++$ $a = a + 1$</td>
<td>$!a$ boolean opposite</td>
</tr>
<tr>
<td>$++ a$ $a = a + 1$</td>
<td></td>
</tr>
<tr>
<td>$a --$ $a = a - 1$</td>
<td></td>
</tr>
<tr>
<td>$-- a$ $a = a - 1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>bitwise</td>
<td></td>
</tr>
<tr>
<td>$&lt;&lt; &gt;&gt;$ left and right bit shit</td>
<td></td>
</tr>
<tr>
<td>&amp;</td>
<td>bit AND OR</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>others</td>
<td></td>
</tr>
<tr>
<td>sizeof(a) byte size</td>
<td></td>
</tr>
</tbody>
</table>
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;
b = a++; // b=?
c = ++a; // c=?
```
// if-else
int a = 10;
if (a < 5) {
  true statement;
} else {
  false statement;
}

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

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

// for loop
for (int i=0; i < 10; i++) {
  statements;
}