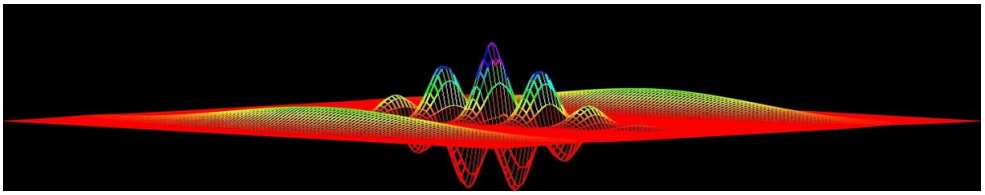
Computational Physics

numerical methods with C++ (and UNIX)



Fernando Barao
Instituto Superior Tecnico, Dep. Fisica
email: barao@lip.pt

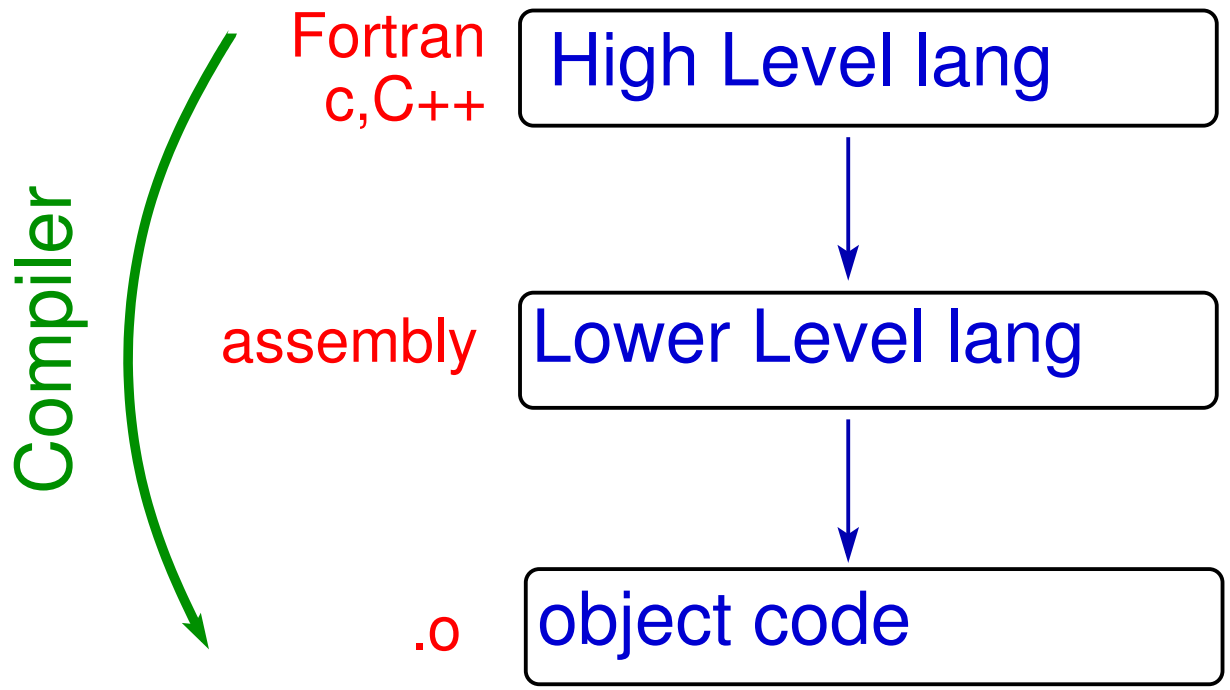
Computational Physics
Compiling a C++ program

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



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++** $\rightarrow$ **g++**, **c** $\rightarrow$ **gcc**, **FORTRAN** $\rightarrow$ **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

g++ compiler

compiler flags that can be used in the compilation process :

generic

-c - output an object file (.o)

-o <**name**> - name of the output file

-g - turn on debugging (so GDB gives more friendly output)

-I<**include path**> - specify an include directory

-L<**library path**> - specify a lib directory

-l<**library**> - link with library lib<library>.a

warnings

-Wall - turns on most current warnings

-Wextra - turns on extra warnings (indicates unitialized variables)

-pedantic - it checks if it is C++ standard code

-Wfloat-equa - checks if one tests an equality between reals (common error)

-Woverloaded-virtual - message signaling that virtual function implemented is different from base class

g++ compiler (cont.)

compiler flags that can be used in the compilation process :

-Wshadow - two similar variables in the same block code

variables conversion

-Wconversion - warns when automatic variable conversions are done

-Wdouble-promotion - warns when a float is converted into double

-Wold-style-cast - warns when conversion a la "c" is done (C++ : static_cast<type>())

optimization

-O or -O2 - turn on optimizations

g++ compiler (cont.)

produce object code and check syntax of test.C (.o) (-v verbose)

```
> g++ -v -c test.C
```

produce executable code of test.C (.exe)

```
> g++ -o text.exe test.C
```

optimizating compiled code and count nb of bytes : (-O0= no optimization, -O1, -O2)

```
> g++ -O1 -o text.exe test.C | wc -c
```

compilink + linking for debugging (no optimization and good code)

```
> g++ -g -Wall -Wextra -o text.exe test.C
```

g++ compiler (cont.)

compilink + linking with static libraries (libm.a)

```
> g++ -o text.exe test.C -L/usr/local/LIB -lm
```

code macro definitions (#define BUFFER 512) can be defined at the command line

```
> g++ -DBUFFER=512 -o test.exe test.C
```

display de preprocessed version of your C++ code

```
> g++ -E test.C > test.i
```

COMMON ERRORS TO AVOID!!!!!!! WARNING!

```
> g++ -o test.C test.C #program disappears
> g++ -E test.C > test.C
```


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...

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   }
```