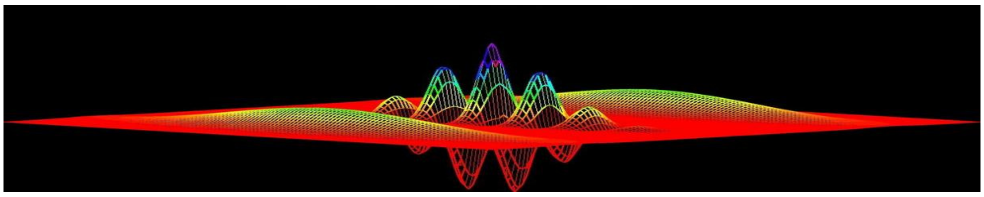
Computational Physics

numerical methods with C++ (and UNIX)



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

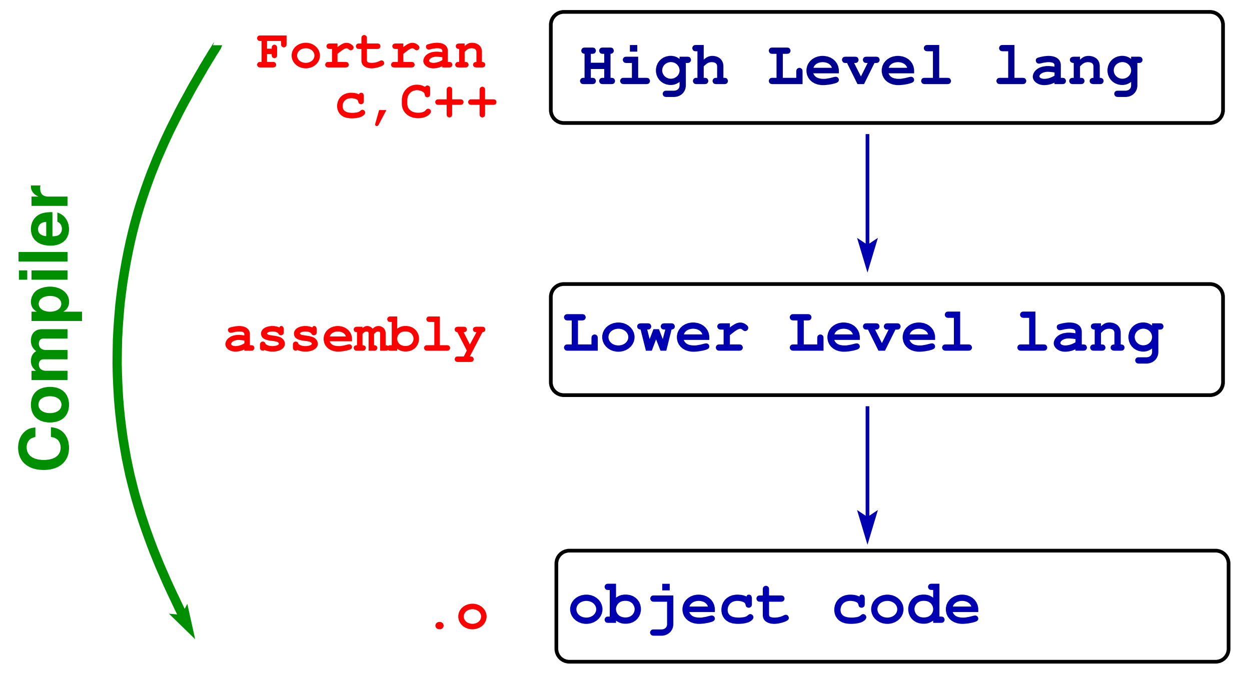
Computational Physics
Compiling a C++ program

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

C++ Compiling chain

- ✔ The C++ program (source code) consists in a set of symbolic instructions and data *(test.C)*
- ✔ The language compiler *(C++ $\rightarrow$ g++)* produces the object code *(test.o)* which is a translation of the source code into the machine language that can be understood by the CPU
  the compiler assigns memory addresses to variables and translates arithmetic and logical operations into machine-language instructions
  ```
  > g++ -c test.C
  ```
- ✔ Thirdly, the object code *(test.o)* is linked with other codes installed or with system binary libraries called by the user code, producing the executable *(test.exe)*
  ```
  > g++ -o test.exe test.C
  ```
- ✔ Additional flags can be used in the compiler process :
  -g - turn on debugging (so GDB gives more friendly output)
  -Wall - turns on most warnings
  -O or -O2 - turn on optimizations
  -o <name> - name of the output file
  -c - output an object file (.o)
  -I<include path> - specify an include directory
  -L<library path> - specify a lib directory
  -l<library> - link with library lib<library>.a