Chapter 1, Getting started

Programming Concepts in Scientific Programming

EPFL, Master class

September 25, 2017

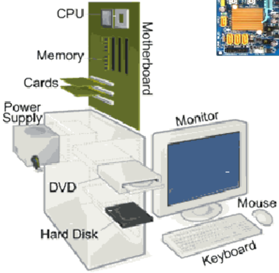
Class organization

- Teaching staff: G. Anciaux, A. Nielsen, L. Pegolotti.
- Lectures: on Mondays, exercises on Fridays
- Follow chapters of the book: Guide To Scientific Computing in C++
- Permanent homework: reading next chapter of the book
- Moodle (password: PCSC2017): material, forum (at the beginning
- Git: material, pdfs, solutions
- Evaluation: project realization and oral presentation

Today

- Introduction to class
- What is a computer ?
- What is a program ?
- Compilation
- Starting chapter 1, pp 1-7
- Tutorial on exercises/projects
  - GNU-Linux
  - Exercises Chap. 1

What is a computer ?



What is a program ?

What is a program ?

Animation with 3 people

- One central memory
- One program memory
- One arithmetic logic unit

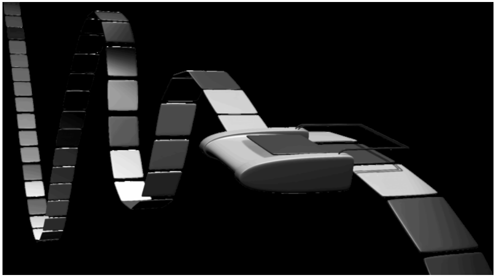
Second program

First program

```
0: *1 = (0)
1: *2 = (0)
2: *0 = (*1 >= 4)
3: if *0 goto 7:
4: *2 = (*2 + *1)
5: *1 = (*1 + 1)
6: goto 3
7: END
```

Turing machine

- A Turing machine is a theoretical device that manipulates symbols contained on a strip of tape
- A computer is a form/implementation of a Turing machine
- Instructions are read sequentially
- Instructions are of the type:
  - Memory access (moving, copying)
  - Algebraic computation (add,sub,mult,div)

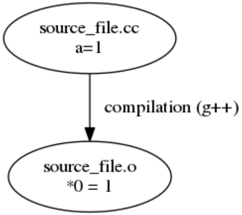


Compilation and linking

A **compiler** is a computer program that transforms **source code** written in a programming/source language into a computer.

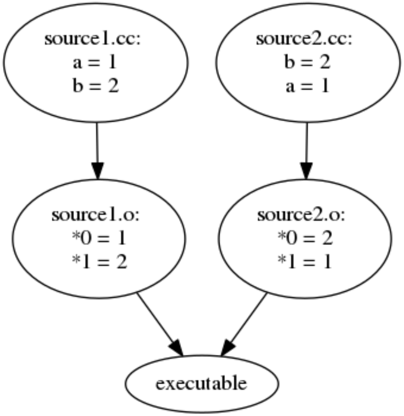
The **GNU compiler** (g++) is a C++ compiler

```
g++ -Wall -c source_file.cc
```



- This will produce an object `source_file.o` file
- "-c" requests for a *compilation*
- "-Wall" to output all warnings and errors

Link editor



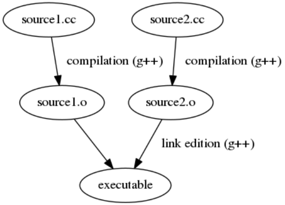
Question:

What are the addresses when files are separated ?

Link editor

A linker or link editor is computer program that

- takes one or more object files (generated by a compiler)
- combines them into a single executable program.



```
g++  object1.o object2.o object3.o -o exec
```

Programming languages

- Lowest level language is denoted as assembler. Processor instructions are explicitly called. Instruction are simply coded and address are translated.
- C language is a low level but is more generic and practical than assembler. Pointer is an important concept of the addressing system in C.
- FORTRAN is dedicated to scientific computing and vector manipulation.
- C++ and java are object oriented programming languages.
- Perl, Python, sh (shell) are script (interpreted) languages that do not need to be compiled.

Brief Introduction to C++

Object Oriented Language, including:

- Modularity: class data and related operations can be worked on independently;
- Abstraction: features and functionality of a class are exposed (public members and methods in .hpp);
- Encapsulation: implementation is hidden (.cpp);
- Extensibility: functionality can be reused with selected parts extended;
- Polymorphism: The same code can be used for a variety of objects;
- Inheritance: allows for code reuse, extensibility and polymorphism.

Why C++?

Object Oriented, Fast, large number of tested and optimized numerical libraries, wide range of compilers (open source and commercial), flexible memory management model.

A first C++ Program

Open the file 'hello.cpp'

A first C++ Program

```
#include <iostream>

int main(int argc, char* argv[])
{
  /* This is a comment and will be ignored by the compiler
     Comments are useful to explain in English what
     the program does */

  // Print "Hello World" to the screen
  std::cout << "Hello World\n";
  return 0;
}
```

Key points:

- instruction: line ending with ;
- the includes
- the main function
- the block
- comments

A first C++ Program

```
#include <iostream>

int main(int argc, char* argv[])
{
  /* This is a comment and will be ignored by the compiler
     Comments are useful to explain in English what
     the program does */

  // Print "Hello World" to the screen
  std::cout << "Hello World\n" ;
  return 0 ;
}
```

Key points:

- instruction: line ending with ;
- the includes
- the main function
- the block
- comments

A first C++ Program

```
#include <iostream>

int main(int argc, char* argv[])
{
  /* This is a comment and will be ignored by the compiler
     Comments are useful to explain in English what
     the program does */

  // Print "Hello World" to the screen
  std::cout << "Hello World\n";
  return 0;
}
```

Key points:

- instruction: line ending with ;
- the includes
- the main function
- the block
- comments

A first C++ Program

```
#include <iostream>

int main(int argc, char* argv[]
{
  /* This is a comment and will be ignored by the compiler
     Comments are useful to explain in English what
     the program does */

  // Print "Hello World" to the screen
  std::cout << "Hello World\n";
  return 0;
}
```

Key points:

- instruction: line ending with ;
- the includes
- the main function
- the block
- comments

A first C++ Program

```
#include <iostream>

int main(int argc, char* argv[])
{
  /* This is a comment and will be ignored by the compiler
     Comments are useful to explain in English what
     the program does */

  // Print "Hello World" to the screen
  std::cout << "Hello World\n";
  return 0;
}
```

Key points:

- instruction: line ending with ;
- the includes
- the main function
- the block
- comments

A first C++ Program

```
#include <iostream>

int main(int argc, char* argv[])
{
  /* This is a comment and will be ignored by the compiler
     Comments are useful to explain in English what
     the program does */

  // Print "Hello World" to the screen
   std::cout << "Hello World\n";
  return 0;
}
```

Key points:

- instruction: line ending with ;
- the includes
- the main function
- the block
- comments

A first C++ Program

```
#include <iostream>

int main(int argc, char* argv[])
{
  /* This is a comment and will be ignored by the compiler
     Comments are useful to explain in English what
     the program does */

  // Print "Hello World" to the screen
  std::cout << "Hello World\n";
  return 0;
}
```

Key points:

- instruction: line ending with ;
- the includes
- the main function
- the block
- comments

Compiling: Try it

```
g++ -Wall -o HelloWorld hello.cpp
```

C++ development

C and C++ are compiled languages. The workflow is:

- Edit source
- Compile
- Run program
- (Debug and go back to editing)

Compiling options

The basic command:

```
g++ -o HelloWorld HelloWorld.cpp
```

With warnings:

```
g++ -Wall -o HelloWorld HelloWorld.cpp
```

With optimization:

```
g++ -O -o HelloWorld HelloWorld.cpp
```

With debugging information:

```
g++ -g -o HelloWorld HelloWorld.cpp
```

When additional libraries are needed:

```
g++ -o HelloWorld HelloWorld.cpp -lm
```

C++ basics

Basic C++ syntax

Variables

```
int row, column;
double temperature;

row = 1;
column = 2;
temperature = 3.0;
```

Variables

```
double tolerance1 = 0.0001;
double tolerance2 = 1e-4;
```

Constant variable ?

```
const double density = 45.621;
```

Variables

Non signed numbers ?

```
signed long int integer4;
unsigned int integer5;
```

Large numbers ?

```
float x1;
double x2;
long double x3;
```

Operations on numerical variables

```
int a = 5, b = 2, c;

c = a+b; // integer addition
c = a-b; // integer substraction
c = a*b; // integer multiplication
c = a/b; // integer division (careful!)
c = a%b; // modulo operation
```

Operations on numerical variables

```
int a = 5, b = 2, c;

c = a+b; // integer addition
c = a-b; // integer substraction
c = a*b; // integer multiplication
c = a/b; // integer division (careful!)
c = a%b; // modulo operation
```

Operations on numerical variables

```
int a = 5, b = 2, c;

c = a+b; // integer addition
c = a-b; // integer substraction
c = a*b; // integer multiplication
c = a/b; // integer division (careful!)
c = a%b; // modulo operation
```

Operations on numerical variables

```
double x = 1.0, y = 2.0, z;

z = (double)a / (double)b; // cast integer to a float

z = x/y;      // floating point division
z = sqrt(x);   // square root
z = exp(y);    // exponential function
z = pow(x, y); // x to the power of y
z = M_PI;      // z stores the value of pi
```

Arrays

```
int array1[2];
double array2[2][3];
```

Arrays

```
int array1[2];
double array2[2][3];
```

Arrays

```
int array1[2];
double array2[2][3];
```

Arrays

```
int array1[2];

array1[0] = 1;
array1[1] = 10;

double array2[2][3];

array2[0][0] = 6.4;
array2[0][1] = -3.1;
array2[0][2] = 55.0;
array2[1][0] = 63.0;
array2[1][1] = -100.9;
array2[1][2] = 50.8;

array2[1][2] = array2[0][1] + array2[1][0];

// Declaration and initialization
double array3[3] = {5.0, 1.0, 2.0};
int array4[2][3] = {{1, 6, -4}, {2, 2, 2}};
```

Arrays

How is the memory organized ?

```
double array2[2][3];
```

ASCII characters and boolean variables

ASCII characters:

```
char letter;
letter = 'a'; // note the single quotation marks

std::cout << "The character is " << letter << "\\n";
```

Boolean variables:

```
bool flag1, flag2;
flag1 = true;
flag2 = false;
```

Strings

```
#include <string>

  std::string city; // note the std::
  city = "Oxford";  // note the double quotation marks

  std::cout << "String length = " << city.length() << "\\n";
  std::cout << "Third character = " << city.at(2) << "\\n";
  std::cout << "Third character = " << city[2] << "\\n";
  // Prints the string in city
  std::cout << city << "\\n";
```

Basic console output

Output a string and a new line:

```
#include <iostream>

  std::cout << "Hello World!\\n";

  int x = 1, y = 2;
  std::cout << "x = " << x << " and y = " << y << "\\n";

  std::cout << "Hello World\\n";
  std::cout.flush();
```

Basic keyboard input

What about input ?

```
int pin;
std::cout << "Enter your PIN, then hit RETURN\\n";
std::cin >> pin;
```

Basic keyboard input

What about input ?

```
int pin;
std::cout << "Enter your PIN, then hit RETURN\\n";
std::cin >> pin;
```

Basic keyboard input

What about input ?

```
int pin;
std::cout << "Enter your PIN, then hit RETURN\\n";
std::cin >> pin;
```

String input

Reading strings containing spaces ?

```
std::string name;
std::cout << "Enter your name and then hit RETURN\\n";
std::getline(std::cin, name);
std::cout << "Your name is " << name << "\\n";
```

The assert statement

Simplest/First way to handle errors

```
#include <cassert>

    double a;

    std::cout << "Enter a non-negative number\\n";
    std::cin >> a;
    assert(a >= 0.0);
    std::cout << "The square root of " << a;
    std::cout << " is " << sqrt(a) << "\\n";
```

The assert statement

Simplest/First way to handle errors

```
#include <cassert>

    double a;

    std::cout << "Enter a non-negative number\\n";
    std::cin >> a;
       assert(a >= 0.0);
    std::cout << "The square root of " << a;
    std::cout << " is " << sqrt(a) << "\\n";
```