Chapter 1, Getting started

Programming Concepts in Scientific Computing

EPFL, Master class

September 20, 2023

Class organization

- Teaching staff: G. Anciaux, H. He

Class organization

- Teaching staff: G. Anciaux, H. He
- Lectures: on Wednesdays, exercises on Fridays

Class organization

- Teaching staff: G. Anciaux, H. He
- Lectures: on Wednesdays, exercises on Fridays
- Follow chapters of the book: Guide To Scientific Computing in C++

Class organization

- Teaching staff: G. Anciaux, H. He
- Lectures: on Wednesdays, exercises on Fridays
- Follow chapters of the book: Guide To Scientific Computing in C++
- Permanent homework: reading next chapter of the book

Class organization

- Teaching staff: G. Anciaux, H. He
- Lectures: on Wednesdays, exercises on Fridays
- Follow chapters of the book: Guide To Scientific Computing in C++
- Permanent homework: reading next chapter of the book
- Moodle: material (at the beginning)
- Ed discussion: forum, questions

Class organization

- Teaching staff: G. Anciaux, H. He
- Lectures: on Wednesdays, exercises on Fridays
- Follow chapters of the book: Guide To Scientific Computing in C++
- Permanent homework: reading next chapter of the book
- Moodle: material (at the beginning)
- Ed discussion: forum, questions
- Git: lectures, pdfs, solutions gitlab.epfl.ch/anciaux/pcsc

Class organization

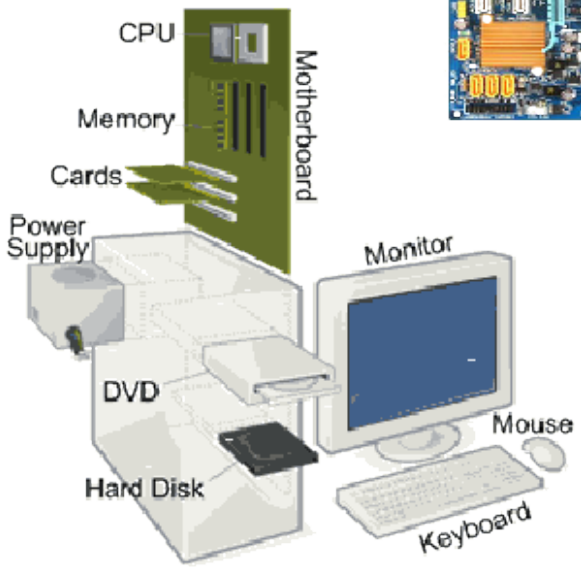
- Teaching staff: G. Anciaux, H. He
- Lectures: on Wednesdays, exercises on Fridays
- Follow chapters of the book: Guide To Scientific Computing in C++
- Permanent homework: reading next chapter of the book
- Moodle: material (at the beginning)
- Ed discussion: forum, questions
- Git: lectures, pdfs, solutions gitlab.epfl.ch/anciaux/pcsc
- 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 computer ?



What is a program ?

Emulating a computer

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

First program

```
*0 = 1
*1 = 2
```

What is a program ?

Emulating a computer

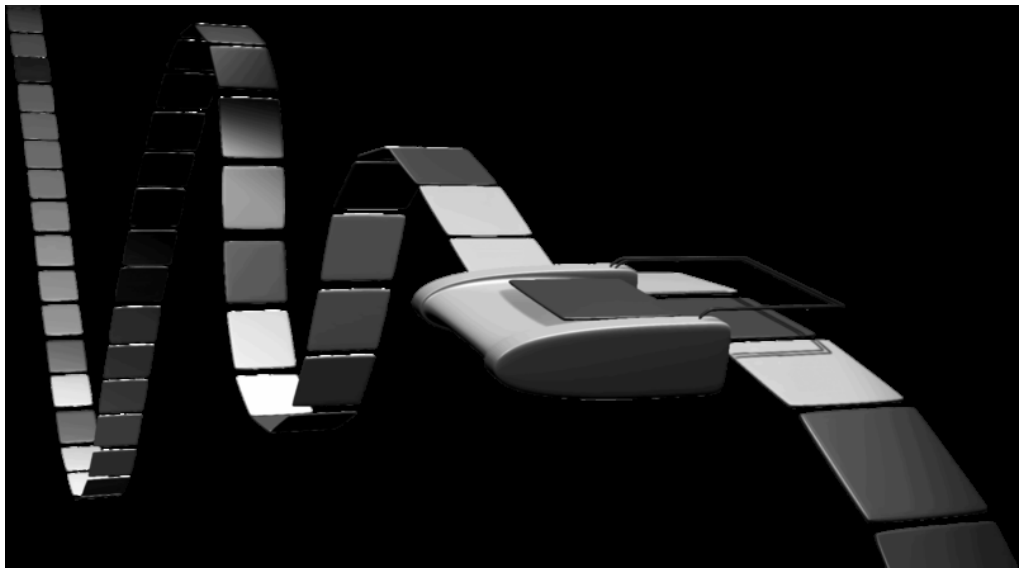
- One central memory
- One program memory
- One arithmetic logic unit

Second program

```
1: *1 = (0)
2: *2 = (0)
3: *0 = (*1 >= 4)
4: if *0 goto 8:
5: *2 = (*2 + *1)
6: *1 = (*1 + 1)
7: goto 3
8: 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)

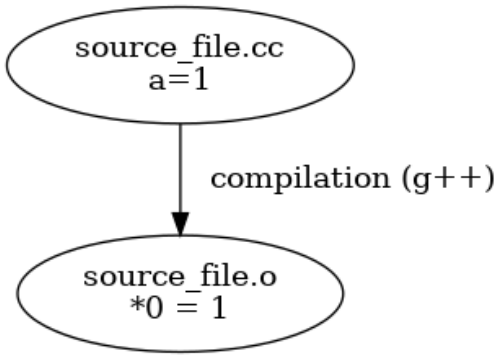


Compilation and linking

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

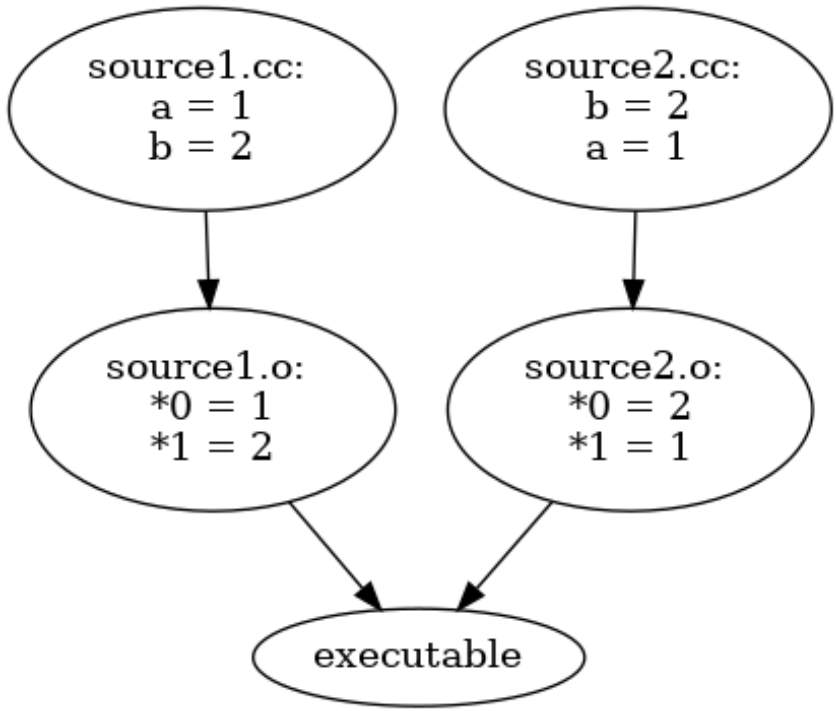
The **GNU compiler** (g++) is a C++ compiler

```
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



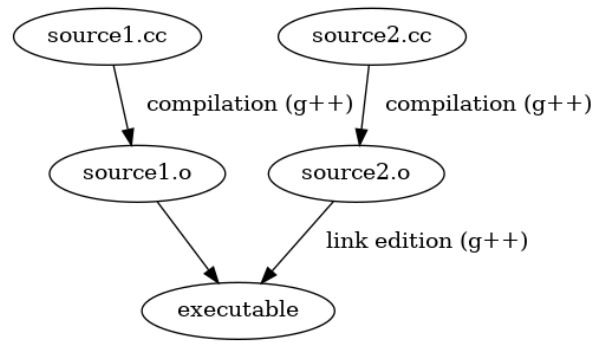
Question:

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

Programming languages

- Lowest level language is denoted as assembler. Processor instructions are explicitly called. Instruction are simply coded and address are translated.

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.

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.

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.

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:

Brief Introduction to C++

Object Oriented Language, including:

- Modularity: class data and related operations can be worked on independently;

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);

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);

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;

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;

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.

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

```
1  #include <iostream>
2
3  int main(int argc, char *argv[]) {
4    /* This is a comment and will be ignored by the compiler
5       Comments are useful to explain in English what
6       the program does */
7
8    // Print "Hello World" to the screen
9    std::cout << "Hello World\n";
10   return 0;
11 }
```

Key points:

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

A first C++ Program

```
1  #include <iostream>
2
3  int main(int argc, char *argv[]) {
4    /* This is a comment and will be ignored by the compiler
5       Comments are useful to explain in English what
6       the program does */
7
8    // Print "Hello World" to the screen
9    std::cout << "Hello World\n" ;
10   return 0 ;
11 }
```

Key points:

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

A first C++ Program

```
1  #include <iostream>
2
3  int main(int argc, char *argv[]) {
4    /* This is a comment and will be ignored by the compiler
5       Comments are useful to explain in English what
6       the program does */
7
8    // Print "Hello World" to the screen
9    std::cout << "Hello World\n";
10   return 0;
11 }
```

Key points:

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

A first C++ Program

```
1  #include <iostream>
2
3  int main(int argc, char *argv[])
4    /* This is a comment and will be ignored by the compiler
5       Comments are useful to explain in English what
6       the program does */
7
8    // Print "Hello World" to the screen
9    std::cout << "Hello World\n";
10   return 0;
11 }
12
```

Key points:

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

A first C++ Program

```
1  #include <iostream>
2
3  int main(int argc, char *argv[]) {
4    /* This is a comment and will be ignored by the compiler
5       Comments are useful to explain in English what
6       the program does */
7
8    // Print "Hello World" to the screen
9    std::cout << "Hello World\n";
10   return 0;
11 }
```

Key points:

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

A first C++ Program

```
1  #include <iostream>
2
3  int main(int argc, char *argv[]) {
4    /* This is a comment and will be ignored by the compiler
5       Comments are useful to explain in English what
6       the program does */
7
8    // Print "Hello World" to the screen
9    std::cout << "Hello World\n";
10   return 0;
11 }
12
```

Key points:

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

A first C++ Program

```
1  #include <iostream>
2
3  int main(int argc, char *argv[]) {
4    /* This is a comment and will be ignored by the compiler
5       Comments are useful to explain in English what
6       the program does */
7
8    // Print "Hello World" to the screen
9    std::cout << "Hello World\n";
10   return 0;
11 }
```

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
```

Basic C++ syntax

Variables (File 'variable.cpp')

```
3       int row, column;
4       double temperature;
```

Variables (File 'variable.cpp')

```
3     int row, column;
4     double temperature;
5     row = 1;
6     column = 2;
7     temperature = 3.0;
```

Variables (File 'variable.cpp')

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

Constant variable ?

Variables (File 'variable.cpp')

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

Constant variable ?

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

Variables (File 'variable.cpp')

Non signed numbers ?

Variables (File 'variable.cpp')

Non signed numbers ?

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

Variables (File 'variable.cpp')

Non signed numbers ?

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

Large numbers ?

Variables (File 'variable.cpp')

Non signed numbers ?

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

Large numbers ?

```
21    float x1;
22    double x2;
23    long double x3;
```

Operations on numerical variables (File 'operations.cpp')

```
3     int a = 5, b = 2, c;
4
5     c = a + b; // integer addition
6     c = a - b; // integer substraction
7     c = a * b; // integer multiplication
8     c = a / b; // integer division (careful!)
9     c = a % b; // modulo operation
```

Operations on numerical variables (File 'operations.cpp')

```
3     int a = 5, b = 2, c;
4
5     c = a + b; // integer addition
6     c = a - b; // integer substraction
7     c = a * b; // integer multiplication
8     c = a / b; // integer division (careful!)
9     c = a % b; // modulo operation
```

Operations on numerical variables (File 'operations.cpp')

```
3     int a = 5, b = 2, c;
4
5     c = a + b; // integer addition
6     c = a - b; // integer substraction
7     c = a * b; // integer multiplication
8     c = a / b; // integer division (careful!)
9     c = a % b; // modulo operation
```

Operations on numerical variables (File 'operations.cpp')

```
11      double x = 1.0, y = 2.0, z;
12
13      z = (double)a / (double)b; // cast integer to a float
14
15      z = x / y;      // floating point division
16      z = sqrt(x);    // square root
17      z = exp(y);     // exponential function
18      z = pow(x, y); // x to the power of y
19      z = M_PI;       // z stores the value of pi
```

Arrays (File 'arrays.cpp')

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

Arrays (File 'arrays.cpp')

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

Arrays (File 'arrays.cpp')

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

Arrays (File 'arrays.cpp')

```
3       int array1[2];

6       array1[0] = 1;
7       array1[1] = 10;
```

Arrays (File 'arrays.cpp')

```
3     int array1[2];

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

4     double array2[2][3];

9     array2[0][0] = 6.4;
10    array2[0][1] = -3.1;
11    array2[0][2] = 55.0;
12    array2[1][0] = 63.0;
13    array2[1][1] = -100.9;
14    array2[1][2] = 50.8;
```

Arrays (File 'arrays.cpp')

```
3     int array1[2];

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

4     double array2[2][3];

9     array2[0][0] = 6.4;
10    array2[0][1] = -3.1;
11    array2[0][2] = 55.0;
12    array2[1][0] = 63.0;
13    array2[1][1] = -100.9;
14    array2[1][2] = 50.8;

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

Arrays (File 'arrays.cpp')

```
3     int array1[2];

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

4     double array2[2][3];

9     array2[0][0] = 6.4;
10    array2[0][1] = -3.1;
11    array2[0][2] = 55.0;
12    array2[1][0] = 63.0;
13    array2[1][1] = -100.9;
14    array2[1][2] = 50.8;

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

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

Arrays

How is the memory organized ?

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

ASCII characters and boolean variables

ASCII characters (File 'ascii.cpp'):

```
4     char letter;
5     letter = 'a'; // note the single quotation marks
6
7     std::cout << "The character is " << letter << "\n";
```

ASCII characters and boolean variables

ASCII characters (File 'ascii.cpp'):

```
4     char letter;
5     letter = 'a'; // note the single quotation marks
6
7     std::cout << "The character is " << letter << "\n";
```

Boolean variables (File 'bool.cpp'):

```
2     bool flag1, flag2;
3     flag1 = true;
4     flag2 = false;
```

Strings (File 'string.cpp')

```
2   #include <string>
```

Strings (File 'string.cpp')

```
2   #include <string>

5     std::string city; // note the std::
6     city = "Oxford";  // note the double quotation marks
7
8     std::cout << "String length = " << city.length() << "\n";
9     std::cout << "Third character = " << city.at(2) << "\n";
10    std::cout << "Third character = " << city[2] << "\n";
11    // Prints the string in city
12    std::cout << city << "\n";
```

Basic console output (File 'console_output.cpp')

Output a string and a new line:

```
1  #include <iostream>

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

Basic console output (File 'console_output.cpp')

Output a string and a new line:

```
1  #include <iostream>

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

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

Basic console output (File 'console_output.cpp')

Output a string and a new line:

```
1  #include <iostream>

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

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

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

Basic keyboard input (File 'keyboard_input.cpp')

What about input ?

Basic keyboard input (File 'keyboard_input.cpp')

What about input ?

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

Basic keyboard input (File 'keyboard_input.cpp')

What about input ?

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

Basic keyboard input (File 'keyboard_input.cpp')

What about input ?

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

String input (File 'string_input.cpp')

Reading strings containing spaces ?

String input (File 'string_input.cpp')

Reading strings containing spaces ?

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

The assert statement (File assert.cpp')

Simplest/First way to handle errors

```
1   #include <cassert>

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

The assert statement (File assert.cpp')

Simplest/First way to handle errors

```
1
2  #include <cassert>
3
7
8     std::cout << "Enter a non-negative number\n";
9     std::cin >> a;
10      assert(a >= 0.0);
11    std::cout << "The square root of " << a;
12    std::cout << " is " << sqrt(a) << "\n";
13
14
```