Advanced C

C++ Getting started

Th.S Nguyen Minh Anh

Phenikaa University

Last Update: 22nd February 2023

Outline

Course Elements

- Lectures: 14 classes
- Assessment: consisting of multiple assignments in class during the course and two exams (midterm and final exams)
- 10% CC, 30% midterm, and 60% final

Course Material

- Regularly check annoucement and assignments on Canvas
- **Slides**: can be found on Canvas
- **Textbook**:
  Lippman, Stanley B., **C++ Primer**, Fifth Edition, Addison-Wesley Professional, 2012 (see textbook folder on Canvas)
- Consider **every web resource** highly suspect until you have reason to believe better of it

Aims of the Course

- Teach/learn:
  - Fundamental programming concepts
  - Key useful techniques
  - Basic Standard C++ facilities
  - Object-oriented programming
- After the course, you'll be able to
  - Write small C++ programs for scientific computations
  - Learn the basics of many other languages by yourself
  - Read much larger programs
- After the course, you will not (yet) be
  - An expert programmer
  - A C++ language expert
  - An expert user of advanced libraries

Motivation

- In your case: programming as a tool for science, to carry out complex computations and simulations, to analyze large amount of data, eg from LHC.
- Foster understanding of problems and their solutions in problem solving. Only by expressing a correct program and constructing and testing a program can you be certain that your understanding is complete.
- Like Mathematics it can be an intellectual exercise that sharpens our ability to think.
- It can be fun

Course Outline

Part I: The basics

- Types, variables
- Strings, vectors, arrays
- Functions

Part II: The C++ libraries

- IO libraries
- Sequential containers
- Associative containers

Part II: Class and more

- Class defining
- Overloaded operations and conversions
- Object-Oriented Programming

Further Remark

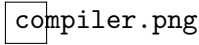
- We use ISO standard C++ 14
- It is good to sketch first the program on a piece of paper. This is a useful step in solving a programming task. The design is a human brain activity not a computer activity and the best way not to skip this stage is to stay away from computer
- We do not use an advanced IDE because our focus is on developing programming skills. When you have acquired them it will be easier to move to an advanced IDE. Also no code runner in VS Code.

Outline

The first program

```
#include <iostream>
int main () // main () is where a C ++ program starts
{
    std::cout << "Hello world !\n" ;
    // output the 13 characters Hello world !
    // followed by a new line
    return 0;
    // return a value indicating success
}
```

Compilation and Linking



- You write C++ source code. Source code is (in principle) human readable
- The compiler translates what you wrote into object code (sometimes called machine code)
  Object code is simple enough for a computer to "understand"
- The linker links your code to system code needed to execute
  E.g., input/output libraries, operating system code, and windowing code
- The result is an executable program: E.g., a .exe file on windows or an a.out file on Unix

Building a Program

Example: a simple program consisting of one single file, `mycode.cpp`, using the GNU C++ compiler on a Unix or Linux system

```
g++ mycode.cpp -o myprogram
```

To run the program

```
./myprogram
```

A First Look at Input/Output

A program that uses the IO library

```
#include <iostream>
int main()
{
    std::cout << "Enter two numbers:" << std::endl;
    int v1 = 0, v2 = 0;
    std::cin >> v1 >> v2;
    std::cout << "The sum of " << v1 << " and " << v2
              << " is " << v1 + v2 << std::endl;
    return 0;
}
```

Writing to a Stream
Reading from a Stream

Using Names from the Standard Library

- Careful readers will note that this program uses std::cout and std::endl rather than just cout and endl.
- The prefix std:: indicates that the names cout and endl are defined inside the namespace named std.
- Namespaces allow us to avoid inadvertent collisions between the names we define and uses of those same names inside a library. All the names defined by the standard library are in the std namespace.
- We can get rid of that by using

```
using namespace std;
```

Comments in C++

```
#include <iostream>
/*
 * Simple main function:
 * Read two numbers and write their sum
 */
int main()
{
    // prompt user to enter two numbers
    std::cout << "Enter two numbers:" << std::endl;
    int v1 = 0, v2 = 0;
    // variables to hold the input we read
    std::cin >> v1 >> v2; // read input
    std::cout << "The sum of " << v1 << " and " << v2
              << " is " << v1 + v2 << std::endl;
    return 0;
}
```

Comments in C++

There are two kinds of comments in C++

- **single-line**: starts with a double slash (//) and ends with a newline Everything to the right of the slashes on the current line is ignored by the compiler. A comment of this kind can contain any text, including additional double slashes.
- **paired**: begins with a /* and ends with the next */. These comments can include anything that is not a */, including newlines. The compiler treats everything that falls between the /* and */ as part of the comment.

Flow of Control: `if/else` statement

supports conditional execution. We can use an if to write a program to to print positive number entered by the user if the user enters a negative number, it is skipped

```
#include <iostream>
int main()
{
    int number;
    std::cout << "Enter an integer: ";
    std::cin >> number;
    // checks if the number is positive
    if (number > 0)
    std::cout << "You entered a positive integer: " << number <<
        endl;

    std::cout << "This statement is always executed.";
    return 0;
}
```

Flow of Control: `while` loop

repeatedly executes a section of code so long as a given condition is true. We can use a while to write a program to sum the numbers from 1 through 10 inclusive as follows:

```
#include <iostream>
int main()
{
    int sum = 0, val = 1;
    while (val <= 10) {
        sum += val; // assigns sum + val to sum
        ++val;
    }
    std::cout << "Sum of 1 to 10 inclusive is "
              << sum << std::endl;
    return 0;
}
```

Flow of Control: `for` loop

the pattern—using a variable in a condition and incrementing that variable in the body—happens so often that the language defines a second statement, the for statement, that abbreviates code that follows this pattern. We can rewrite this program using a for loop to sum the numbers from 1 through 10 as follows:

```
#include <iostream>
int main()
{
    int sum = 0;
    for (int val = 1; val <= 10; ++val)
        sum += val; // equivalent to sum = sum + val
    std::cout << "Sum of 1 to 10 inclusive is "
              << sum << std::endl;
    return 0;
}
```

Example: Reading an Unknown Number of Inputs

```
#include <iostream>
int main()
{
    int sum = 0, value = 0;
    // read until end-of-file, calculating a running total of all
        values read
    while (std::cin >> value)
        sum += value; // equivalent to sum = sum + value
    std::cout << "Sum is: " << sum << std::endl;
    return 0;
}
```

If the stream is valid—that is, if the stream hasn't encountered an error—then the test succeeds. An istream becomes invalid when we hit end-of-file or encounter an invalid input, such as reading a value that is not an integer. An istream that is in an invalid state will cause the condition to yield false.

Several cool things about C++ over C

- bool
- string
- vector
- struct/class member functions

In the next lecture

Will talk more about types, values, variables, declarations, input and output streams.