CSE 5910
Software Foundations

Instructor: Aijun An
aan@cse.yorku.ca
Office: Lassonde 2048
Office Hours: Tue & Thur 2:45pm – 3:30pm

http://www.cse.yorku.ca/course/5910

1

Outline

Course Information
- Course content and objective
- Textbook
- Class format
- Marking scheme

Introduction to Computers

Introduction to Programming Languages

Introduction to C++

How to create, compile and run a C++ program

How to create an CSE account and use Linux (in the lab)

2

Course Content

C++
- One of the most popular programming languages
  - Used in many fields including quantitative finance
  - Huge code base in place that is hard to replace
- Has both high-level and low-level language features.
  - High-level features make it easy to use for general purpose
  - Low-level features make it powerful and its programs run fast.
- A multi-paradigm programming language, supporting both
  - Object-oriented programming
  - Procedural programming

3

Course Objective

Learn how to read, design and write C++ programs

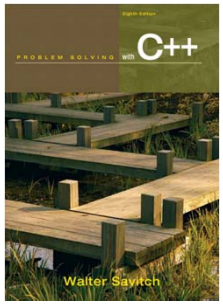
By the end of the course, you should be able to
- Understand C++ programs
- Design programs
- Write modest-sized programs in C++
- Test and debug C++ code

4

Textbook

Title: Problem Solving with C++ (8th Edition)

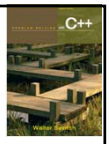
Author: Walter Savitch



- The book is written for students without prior programming experience.
- Has self-test exercises with answers.

5

Textbook



The book is sold in the following forms:

- hard copy - $119.70 at York University Bookstore
- eText - $49.95 at York University Bookstore
- hard copy plus MyProgrammingLab with Pearson eText Access Card, 8th Edition - $128.27 at Pearson
- eText with MyProgrammingLab – Instant Access – for Problem Solving with C++, 8/e - $90.10 at CourseSmart

6

Class Format

Lectures and Labs

- Time: 11:30am – 3:00pm
- Location: Lassonde 1002 (from the 2nd class)

Lab content:

- Exercise on computers with assistance from the instructor and TA

Format

- The first 2 to 2.5 hours are lectures.
- The last 1-1.5 hour is lab exercise.

7

Marking Scheme

The weight distribution of the course components is as follows:

- 30% - Assignments (5 assignments)
- 10% - Labs (1% for each lab)
- 25% - Midterm test (November 9 in class)
  - including written test and lab test
- 35% - Final exam
  - including written test and lab test

8

Course Website

http:www.cse.yorku.ca/course/5910

or

https://wiki.cse.yorku.ca/course_archive/2012-13/F/5910/

It contains all the information about the course

- Lecture notes will be posted there.
- Assignments will be posted there.
- How to remotely access the CSE Linux server in order to use the C++ complier.
- Course policies
- ……

9

Outline

Course Information

- Course content and objective
- Textbook
- Class format
- Marking scheme

Introduction to Computers

Introduction to Programming Languages

Introduction to C++

How to create, compile and run a C++ program

How to create an CSE account and use Linux (in the lab)

10

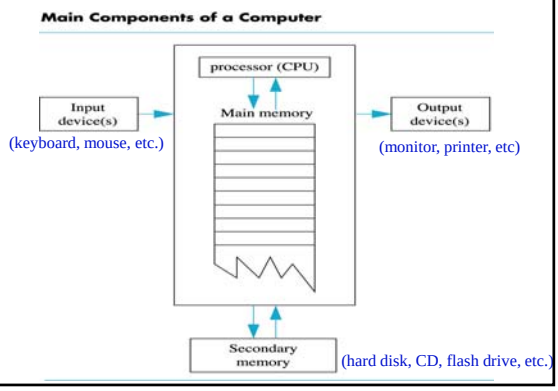
Computer Systems

A computer system contains

- Hardware
  - the collection of physical elements that comprise a computer system
  - Examples: processor, main memory, disk, screen, keyboard, etc.
- Software
  - The collection of programs used by a computer system
  - A computer program is
    - a set of instructions for a computer to follow
  - Examples: text editors, operating systems, etc.

11

Computer Hardware



12

Processor (CPU)

CPU stands for *Central Processing Unit*

It's the brain of the computer

- Follows the instructions in a program
- Perform calculations specified by the program

Typical capabilities of CPU include:

add
subtract
multiply
divide
move data from location to location

13

Main Memory

Internal storage area in the computer

- also called memory or RAM (Random Access Memory)

Stores program instructions and data currently being processed by the CPU

Compared to other storage media

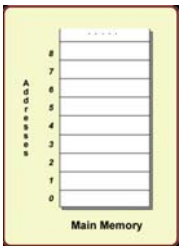
- Access to RAM is fast
- Information is not saved when the computer is powered off.

14

Main Memory

Can be considered as an array of boxes, each of which holds a single byte of information

- A byte contains 8 bits. A bit is a binary digit (0 or 1)
- Each box has an address
- *Random access*:
  - Given an address, CPU can directly obtain the information at that address.



15

Secondary Memory

Stores programs and data in the forms of files

Files cannot be directly processed by CPU.

- A file must be loaded into the main memory to be processed.

Slower to access than main memory

Often requires *sequential access*

- read through the file from the beginning in the order in which it is stored

The information on the secondary memory is not lost when the system is powered off.

16

Types of Secondary Memory

Hard disk

- Fast (but slower than main memory)
- Usually fixed in the computer and not normally removed

Floppy disk

- Slow
- Easily shared with other computers

Compact disk

- Slower than hard disks
- Easily shared with other computers
- Can be read only or re-writable

Flash memory

- Slower than hard disks
- Easily shared with other computers

17

Computer Program

A sequence of instructions written to perform a specified task on a computer

Two forms of a program:

- Executable program
  - Binary code (consists of 0s and 1s).
  - Can be directly executed by the computer
- Source code
  - Written in a *programming language* (human understandable)
  - Need to be either converted into an executable program by a *compiler* or may be executed with the aid of an *interpreter*.

18

Computer Software

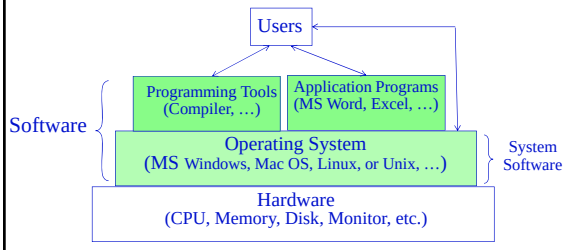
A collection of programs and related data

Types of software (according to functionalities)

- System software
  - the programs used to operate and manage computer hardware
  - Examples: operating systems (Windows, Mac OS, Linux), device drivers, utilities, etc.
- Programming tools
  - Used to create, translate, combine, debug and maintain programs
  - Examples: compilers, interpreters, linkers, debuggers, text editors.
- Application software
  - Designed to help the user to perform specific tasks
  - Examples: Word processors, web browsers, accounting, financial analysis, computer games, media plays, etc.

19

Computer Software Structure



20

Operating System

The most important system program that

- manages and allocates computer hardware resources
- allows us to communicate with the hardware
- responds to user requests to run other programs

Common operating systems include:

- Windows, MAC OS, Unix, Linux, DOS, etc.
- We are going to use Linux in this course.

21

Outline

Course Information

- Course content and objective
- Textbook
- Class format
- Marking scheme

Introduction to Computers

Introduction to Programming Languages

Introduction to C++

How to create, compile and run a C++ program

How to create an CSE account and use Linux (in the lab)

22

Programming Languages

An artificial language designed to write computer programs

Common programming languages include

C, C++ , Java, Pascal, Visual Basic, FORTRAN, Perl, COBOL, Lisp, Scheme, Ada, C#, Python, MATLAB

23

Programming Languages

Programming languages can be classified along multiple axes:

- High-level or low-level
- Compiled, interpreted or just-in-time compiled
- Procedural, object-oriented, etc.

24

High-level vs. Low-level

Low-level language

- deals more with direct hardware interaction, and thus
- is more suitable for programs like device drivers code that really needs access to the hardware.
- difficult to port to other platforms.
- must be converted into executable machine code (zeros and ones)

Example: assembly language

- An assembly language command such as

ADD X Y Z

might mean add the values found at x and y in memory, and store the result in location z.

25

High-level vs. Low-level

High-level language

- resembles human languages
- is designed to be easy to read and write
- takes less time to develop a program
- generally portable among different platforms
- must be translated to zeros and ones for the CPU to execute a program

Examples:

C, C++ , Java, Pascal, Visual Basic, FORTRAN, Perl, COBOL, Lisp, Scheme, Ada, C#, Python, MATLAB

26

Compiled vs. Interpreted

Compiled language

- Programs need to be translated into an executable machine code by a *compiler* and later executed.
- The executable program runs faster.
- Example: C, C++, Fortran

Interpreted language

- Programs can be executed immediately with the aid of an *interpreter*
- Program runs slower
- Example: Perl, MATLAB, Basic

Just-in-time compiled language

- Program is first compiled into portable byte code, which can later be executed with the aid of an interpreter.
- Slower than the compiled, but faster than the purely interpreted.
- Example: Java

27

Procedural vs. object-oriented

Procedural:

- Program consists of a set of functions or procedures
- Examples: C, C++, Visual Basic, MATLAB, ….

Object-oriented:

- Program consists of classes and objects
- Examples: Java, C++, Python, ….

28

Outline

Course Information

- Course content and objective
- Textbook
- Class format
- Marking scheme

Introduction to Computers

Introduction to Programming Languages

Introduction to C++

How to create, compile and run a C++ program

How to create an CSE account and use Linux (in the lab)

29

Introduction to C++

A high-level programming language with some low-level constructs

- High-level constructs (such as branching and looping statements, functions and objects) make it easy to write programs
- Low-level constructs (such as pointers) make it possible to manipulate hardware, such as direct access to main memory.

A compiled language

A multi-paradigm language

- Supporting both procedural and object-oriented programming

C++ programs generally run

- faster than Java, C# and other high-level language programs,
- but slower than C programs.

30

Introduction to C++

Where did C++ come from?

- Derived from the C language
  - C is a procedural language. C++ can be considered C with objects.
- C was derived from the B language
- B was derived from the BCPL language

Why the '++'?

- ++ is an operator in C++ and results in a cute pun

31

C++ History

C developed by Dennis Ritchie at AT&T Bell Labs in the 1970s.

- Used to write and maintain UNIX operating systems
- Many commercial applications written in C

C++ developed by Bjarne Stroustrup at AT&T Bell Labs in the 1980s.

- Overcame several shortcomings of C
- Incorporated object oriented programming
- C remains a subset of C++

32

A Sample C++ Program

```
#include<iostream>
using namespace std;

int main()
{
  cout << "Hello\n";
  cout << "This is my first C++ program!\n";

  return 0;
}
```

A **simple** C++ program begins this way

It ends this way

Main part of the program with 2 ***output statements***

Output of this program when it runs:

```
Hello
This is my first C++ program!
```

33

Another Sample C++ Program with Input

```
#include<iostream>
using namespace std;

int main()
{
  int number1, number2;
  int sum;

  cout << "Please input two numbers: ";
  cin >> number1 >> number2;
  sum = number1 + number2;
  cout << "The sum of these two number is ";
  cout << sum << "\n";

  return 0;
}
```

A **simple** C++ program begins this way

Variable declarations

Input statement

Assignment statement

It ends this way

Output of this program when it runs:

34

Layout of a Simple C++ Program

```
#include<iostream>
using namespace std;

int main()
{
  Variable Declarations;

  Statement 1;
  Statement 2;
  ……
  Statement last;

  return 0;
}
```

Directives

Every C++ program must contain a **`main()`** function. Braces{ and } mark the beginning and end of the main function.

35

Explanation of Code (1)

The **`#include`** directive:

```
#include<iostream>
```

- It tells the compiler where to find information about certain items (such as **`cout`** and **`cin`**) used in the program.
- **`iostream`** is the name of a library file that contains the definitions of **`cin`** and **`cout`**

The **`using`** directive:

```
using namespace std;
```

- It says the names defined in **`iostream`** are to be interpreted in namespace **`std`** (to be explained later in the course).
- **`std`** stands for "standard".

36

Explanation of Code (2)

Variable declaration line

```
int number1, number2;
int sum;
```

- Declare three variables to hold integers
- **int** means integers

37

Explanation of Code (3)

Output statements:

```
cout << "Hello\n";
cout << "This is my first C++ program!\n";
cout << "Please input two numbers: ";
cout << "The sum of these two number is ";
cout << sum << "\n";
```

- **cout** (see-out) used for output to the monitor
  - Think of **cout** as a name for the monitor
- **<<** is the ***insertion*** operator
  - It inserts the data that follows it into the monitor.
  - **"<<"** points to where the data is to end up
  - Can use more than one **"<<"** in one output statement
- '**\n**' causes a new line to be started on the monitor

38

Explanation of Code (4)

Input statement:

```
cin >> number1 >> number2;
```

- **cin** (see-in) used for input from the keyboard
  - Think of **cin** as a name for the keyboard
- "**>>**" is called the ***extraction*** operator
  - It extracts data from the keyboard
  - "**>>**" points from the keyboard to a variable where the data is stored

When the program runs, the above input statement takes two numbers from the keyboard, separated by space, tab or newline.

39

Explanation of Code (5)

Assignment statement:

```
sum = number1 + number2;
```

- Performs a computation
- **+** is used for addition
- '**=**' causes variable **sum** to get a new value based on the calculation shown on the right of the equal sign

40

Program Layout and Style

Compiler accepts almost any pattern of line breaks and indentation

Example 1:

```
cout << "Hello\n";
```

is the same as

```
cout    <<"Hello\n";
```

Eaxmple 2:

```
cin >> number1 >> number2;
sum = number1 + number2;
```

is the same as:

```
cin >> number1 >> number2; sum = number1 + number2;
```

41

Program Layout and Style (Cont'd)

Programmers format programs so they are easy to read

- Use only one statement per line
- Indent statements
- Place opening brace '{' and closing brace '}' on a line by themselves

Variables are declared before they are used

- Typically variables are declared at the beginning of the program
- Statements and declarations end with a semi-colon

42

Outline

Course Information
- Course content and objective
- Textbook
- Class format
- Marking scheme

Introduction to Computers

Introduction to Programming Languages

Introduction to C++

How to create, compile and run a C++ program

How to create an CSE account and use Linux (in the lab)

Creating a C++ program

You can create a C++ program (source code) with a text editor

There are many text editors:
- On Linux: **`nedit`**, **`nano`**, **`pico`**, **`emacs`**, **`vi`**, etc
- On Windows: Notepad, WordPad, etc.
- On Mac: TextEdit

With a text editor, type in the program and save it in a file
- The file should have an extension name **`cpp`** , for example,

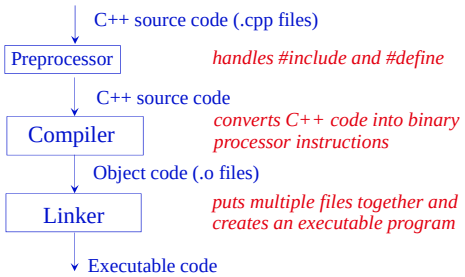
  `program1.cpp`

- The prefix of the file can be named using letters, digits and underscores (in Linux).

Compiling a C++ program

A C++ source code needs to be converted to machine code (i.e., executable program) to be run by CPU

A C++ executable program is built in three stages:



C++ Compiler on Linux

Compiler: **`g++`**
- It does all the three steps: preprocessing, compiling and linking

How to use **`g++`**: on Linux command line, type
- **`g++ program1.cpp`**

  This will generate an executable program named **`a.out`** in the current directory.
- **`g++ program1.cpp -o program1`**

  or
- **`g++ –o program1 program1.cpp`**

  This will generate an executable program named **`program1`** in the current directory.

Running a C++ Executable Program

On a Linux command line, type the name of the executable program. For example,

- **`a.out`**

  This will invoke the **`a.out`** program.

- **`program1`**

  This will invoke the **`program1`** program.

.

Debugging a Program

Bug
- A mistake in a program

Debugging
- Eliminating mistakes in programs
- Term used when a moth caused a failed relay on the Harvard Mark 1 computer. Grace Hopper and other programmers taped the moth in logbook stating:

  "First actual case of a bug being found."

Types of Program Errors

Syntax errors
- Violation of the grammar rules of the language
- Discovered by the compiler
  - Error messages may not always show correct location of errors
- Example: miss a semicolon (;) at the end of a statement

Run-time errors
- Error conditions detected by the computer at run-time
- Example: divide a number by zero

Logic errors
- Errors in the program's algorithm
- Most difficult to diagnose
- Computer does not recognize an error

49

Lab 1

Create a CSE account

Use Linux commands

Use Text Editor on Linux to Create a C++ program
- nedit
- nano (http://mintaka.sdsu.edu/reu/nano.html)

Compile a C++ program

Run a C++ program

Remove login instructions

50

Home Exercises

Practice remote login from your home computer or laptop to the CSE Linux Server
- Instructions are at

https://wiki.cse.yorku.ca/course_archive/2012-13/F/5910/accessmatlab

Practice creating, compiling and running a C++ program
- from home through remove login

Read Chapter 1 and do as many self-test exercise questions in Chapter 1 as possible

51

Home Exercises (Cont'd)

Write, compile and run a program that outputs:

```
     Hello
Hello     Hello
     Hello
```

Write, compile and run a program that outputs:

```
  XXX
 X   X
X
X
X
X
 X   X
  XXX
```

52