SFU SIMON FRASER UNIVERSITY
BURNABY

Welcome to CMPT 135 !

Introduction to Computer Programming II

Instructor : Scott Kristjanson
TA : Wenqiang Peng

SFU Surrey, Spring 2016

CMPT 135 Overview

Introduction to Computing Programming II

- An introduction to computing science and computer programming using the C++ Programming Language
- Suitable for students who have completed CMPT 130 or have equivalent programming experience
- Intended for students who will major in computing science or a related program

Course Topics

General introduction

C++ Basics

Review of Elementary programming:

- basic data types and conversions
- control: if-then-else, for, while
- functions, procedures, modularity

Elementary data structures

- vectors, arrays, linked lists, stacks, queues

Object-oriented concepts:

- objects, classes, encapsulation
- Inheritance, overloading, and polymorphism
- Iterators, Abstract Data Types

Recursion

Fundamental algorithms

- Sorting and Searching

Design and Implementation of medium and large scale applications

Analysis of Algorithms: Computability and complexity

Exception handling

Templates and the Standard Template Library as time permits

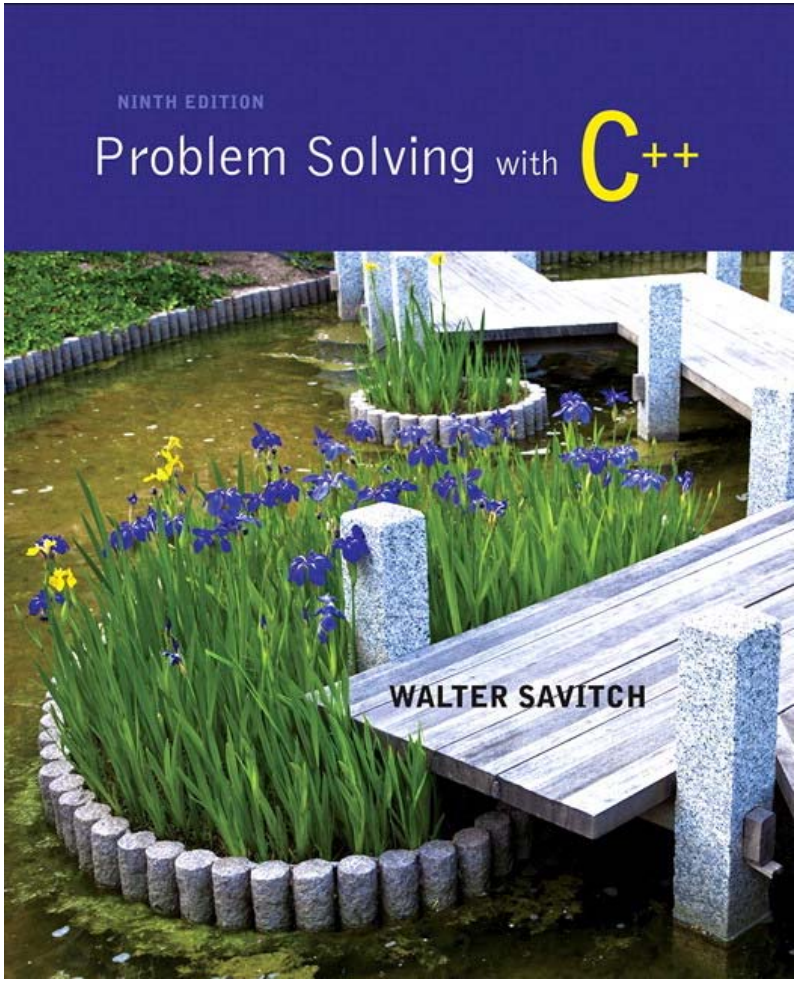
Required Text:

Problem Solving with C++ 9th Edition [1]

by Walter Savitch

Available from the SFU Bookstore in soft-cover or loose-leaf editions

The Library also has copies available in the Reserve section



Course Web Resources

Course Description available via Course Central:

https://portal.cs.sfu.ca/portal/outlines/1161-CMPT-135-D100/

Course Webpage:

https://courses.cs.sfu.ca/2016sp-cmpt-135-d1/pages/

- Course Outline and Schedules
- Assignments and Due Dates
- Marking Schemes and Solutions

Email Communications:

- Email course related questions to cmpt-135-help@sfu.ca
- For confidential questions, email me directly at: skristja@sfu.ca
- The TA and instructor will use email to send announcements and tips during the semester. You should read your SFU email account regularly, at least a few times each week.

Tutorials

Tutorials contain a mix of demos, exercises, and one-on-one help from TA
CSIL PCs can multi-boot either Windows or Linux
Encouraged to work in pairs on Lab Exercises

It is each student's responsibility during your scheduled section to:

- Attend and sit at a PC in Room 4080
- Read Lab Exercises ahead of time and be prepared
- Watch Lab Demos and ask questions
- Complete any work specified in the Lab Exercises by the Due Date
- Submit completed exercises on-line if specified to do so in the Lab Exercises
- Ensure to check in with TA for attendance ***before you leave***
- Logout at the end of Tutorial to make room for students in next Section
- You can continue your work in another lab or remotely

CMPT 135 Tutorials – Surrey 4080

Section D101 – 11:30am – 12:20PM
Section D102 – 12:30am – 1:20PM
Section D103 – 1:30am – 2:20PM

If you have never used CSIL before, read the CSIL FAQ here: [6]
http://www.sfu.ca/computing/about/support/csil.html

For remote access to a CSIL Linux Server: [18]
https://www.sfu.ca/computing/about/support/csil/unix/how-to-use-csil-linux-cpu-server.html

Office Hours

My Office Hours are:

Wed, Friday: Noon-1pm, Surrey 4136

If these times do not work for you, please feel free to email me at skristja@sfu.ca and we can arrange for a time to meet.

Course Marking Scheme

10% - Labs (5% Participation, 5% Lab Exercises)
4% - Quizzes
16% - Assignments
20% - Midterm
50% - Final Exam

Note:
Students must attain an overall passing grade on the weighted average of exams in the course in order to obtain a clear pass (C- or better).

Students who do not obtain a passing grade in the final exam may not obtain a pass (D or better).

Academic Dishonesty[4]

We take academic dishonesty very seriously in the School of Computing Science. Academic dishonesty includes (but is not limited to) the following:

- copying another student's assignment
- allowing others to complete work for you
- allowing others to use your work
- copying part of an assignment from an outside source
- cheating in any way on a test or examination

If you are unclear on what academic dishonesty is, please read Policy 10.02. It can be found in the "Policies & Procedures" section in the SFU web site.

Cheating on a lab or assignment will result in a mark of 0 on the piece of work. At the instructor's option, further penalties may be requested. Any academic dishonesty will also be recorded in your file, as is required by University policy.

Any academic dishonesty on the midterm or final will result in a recommendation that an F be given for the course.

By the end of this course…

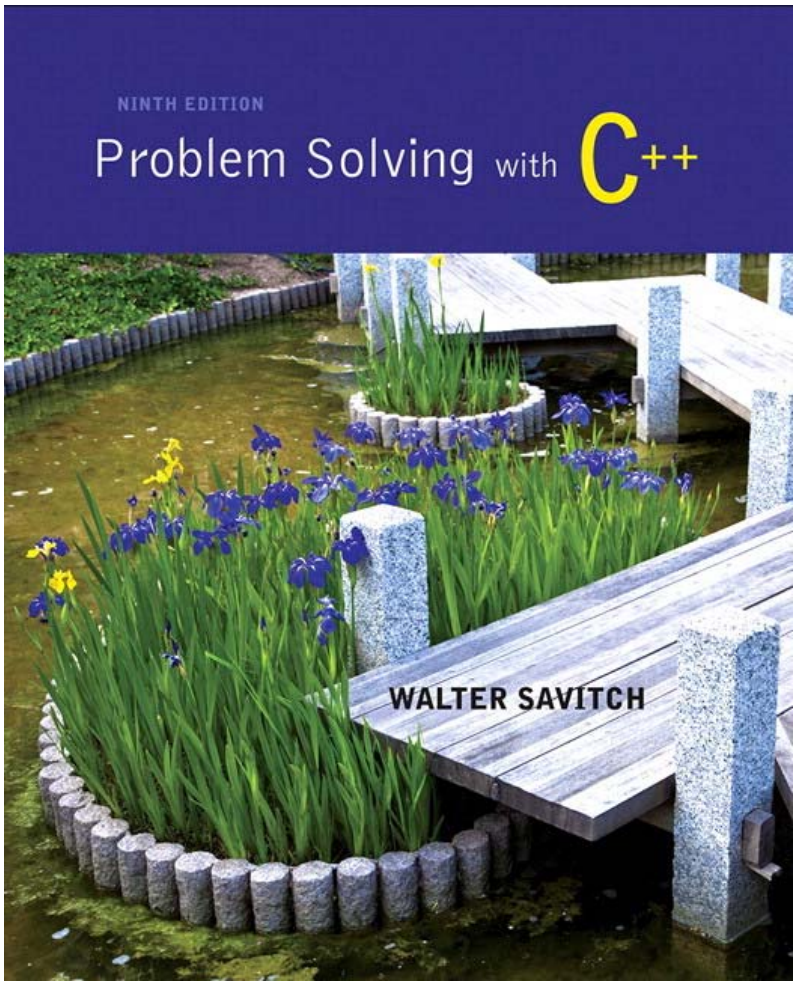
You should be able to:

- Describe some of the basic ideas of computer science
- Explain what Algorithms and computer programming are
- Identify Software Engineering principles of good design
- Design Data Structures appropriate to solve problems
- Design Algorithms to solve moderately complex problems
- Use Object Oriented concepts in your design
- Write Programs in the C++ Programming Language
- Run and Test C++ Programs in an IDE such as NetBeans
- Create Programs that are easy to understand and maintain
- Analyze how much memory and time an algorithm will take

Keep these goals in mind as you progress through the course

Now let's get started with CMPT 135

Introduction



Chapter 1

Introduction – Sections 1.1 to 1.2

We are going to embark on a Journey

Through the world of:

- Data Structures
- Algorithms
- Computer Programs using a high level language called C/C++

In order to Solve Problems

- That's the purpose of writing computer programs!
- C++ is just a tool, these concepts apply to many languages.

How does CMPT135 fit into Computing Science?

Computing science is the study of algorithms and their:

- Formal and mathematical properties
- Hardware realizations
- Linguistic realizations
- Application of algorithms to solve problems

Cmpt 135 is focused on the Application of algorithms

Other courses exist which focus on the other aspects:

- Formal and mathematical properties (CMPT 225/307/450)
- Hardware Architectures (CMPT150/250)
- Programming languages (CMPT383)

We can sum up the goal of this course with the question:

"How can I implement an algorithm for my application on a PC using C++?"

What is an Algorithm?

The concept of an “algorithm” is fundamental to all of computing science and programming.

> *An algorithm is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.*[7]

Notice some important key words in the definition:

- *Unambiguous*: All steps should be clear and explicit
- *Problem*: An algorithm should solve a particular problem
- *Legitimate input*: An algorithm typically needs valid input to do its job
- *Finite amount of time*: If we start the algorithm, it had better finish eventually

Stated simply:

An algorithm is a set of instructions that can be used to solve a problem.[4]

What kind of problems can Algorithms solve?

The kind of applications you use everyday:

- Smart Phones
- Websites
- Games

But also the kind of research happening at SFU right now:

- Application of the Cloud and Big Data to solve large problems
- Mapping diseases to the human genome
- Mapping patient brain function for improving brain surgery results
- Early detection of Skin Cancers using Smart Phones
- Energy saving techniques for Smartphone's
- Real-time encoding of Video for smart-phones with Cloud Computing
- Improved Multimedia Encoding Algorithms for 3D Video
- Faster Graphics and Computer Vision and Artificial Intelligence
- Bioinformatics, Robotics, Bio-molecular Computing
- Improved Computer Architecture Designs
- Telecom, Networks and Graph Theory, Network Usage Algorithms

Are there any problems Algorithms Cannot Solve?

Yes! ☹

Some problems just take too long, they are *intractable*:

- Finding optimal packing of N items in a backpack to maximize value
- Finding the most energy efficient folding of a protein of length N

(for large N, these problems can take billions of years to solve!)

Some problems are easily computed but *undecidable*:

- Does the decimal expansion of PI have a sequence of 7's of length N? (One solution prints "Yes", another prints "No", but which is correct?)

Some problems are *not computable* with ANY algorithm:

- Given an program to analyze, determine if the program ever halts (One can prove that no such algorithm is possible!)

Problem Solving

Solving a problem consists of multiple activities:

- Understanding the problem
- Designing a solution, representing the information
- Considering alternatives and refining the solution
- Implementing the solution
- Testing the solution …..

And do it all over again!…

These activities are not purely linear,

they overlap and interact

Building on Your Existing Knowledge

From Cmpt 130, it's assumed that you will be familiar with the basic concepts of C programming. (Section numbers refer to the course text.)

- Data types and Variables (§2.1, 2.3)
- Expressions (§2.3)
- Strings (§2.3, 8.3)
- Conditionals (if-then-else) (§2.4)
- Definite (for) and indefinite (while) loops (§2.4-3.4)
- Functions and Procedures (§4.1-4.5, 5.1).
- Basic terminal input/output (§2.2)

We will review these concepts as we explore object-oriented programming using C++

C and C++

The C Programming Language

- 2nd most popular language (Java is #1) [9]
- Compiled Procedural language
- Developed in 1972 by Dennis Ritchie for UNIX
- Used for System Programming, Networking, Embedded systems
- Strengths : Speed
- Weakness: Pointer and Memory Management difficult to master

The C++ Programming Language

- 3rd most popular language
- Compiled Multi-Paradigm language
- Written as an update to C in 1979 by Bjarne Stroustrup
- Backwards-compatible with C and brings object-orientation, which can help in larger projects.
- Used to create a wide array of applications from games to office suites.
- Strengths : Speed
- Weakness: C++ is older and considered more clumsy than newer object-oriented languages such as Java or C#

Tools

In this course, you will need to become familiar with these tools:

- Linux : Most of you are already familiar with this operating system
- NetBeans : Interactive Development Environment (IDE)
- g++ : GNU C++ Compiler via Linux command line

Why Linux, NetBeans, and the Command Line?

- Combination considered to be a professional set of tools – very efficient
- NetBeans IDE has more than you need for this course, you are building a foundation for future courses and the job market. Eclipse is also good.
- Command Line often used in industry to build and test automatically

Can I use other environments like Windows and NetBeans?
Yes but…

- **TA will be using Linux and Command Line to test your Assignments**
- ***If TA cannot compile and run your assignment in CSIL, you get ZERO.***

You may develop on Windows or a MAC using VMWare Player [15][16][17]

But TEST it on Linux in the CSIL before submitting!

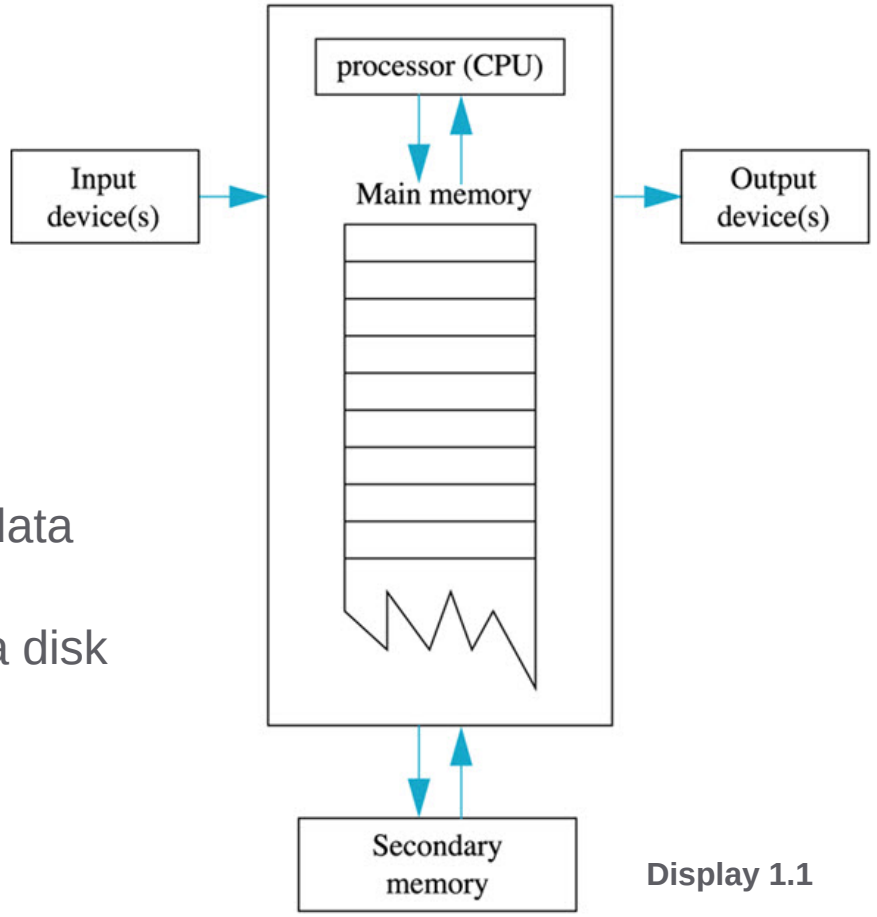
<u>Important</u>: Once submitted, Download it from CourSys and retest!!

Computer Organization

Five main components

- Input devices
  - Communication to the computer
- Output devices
  - Communication to the user
- Processor (CPU)
  - Does the computation
- Main memory
  - Contains the running program and data
- Secondary memory
  - Permanent record of data often on a disk

Main Components of a Computer

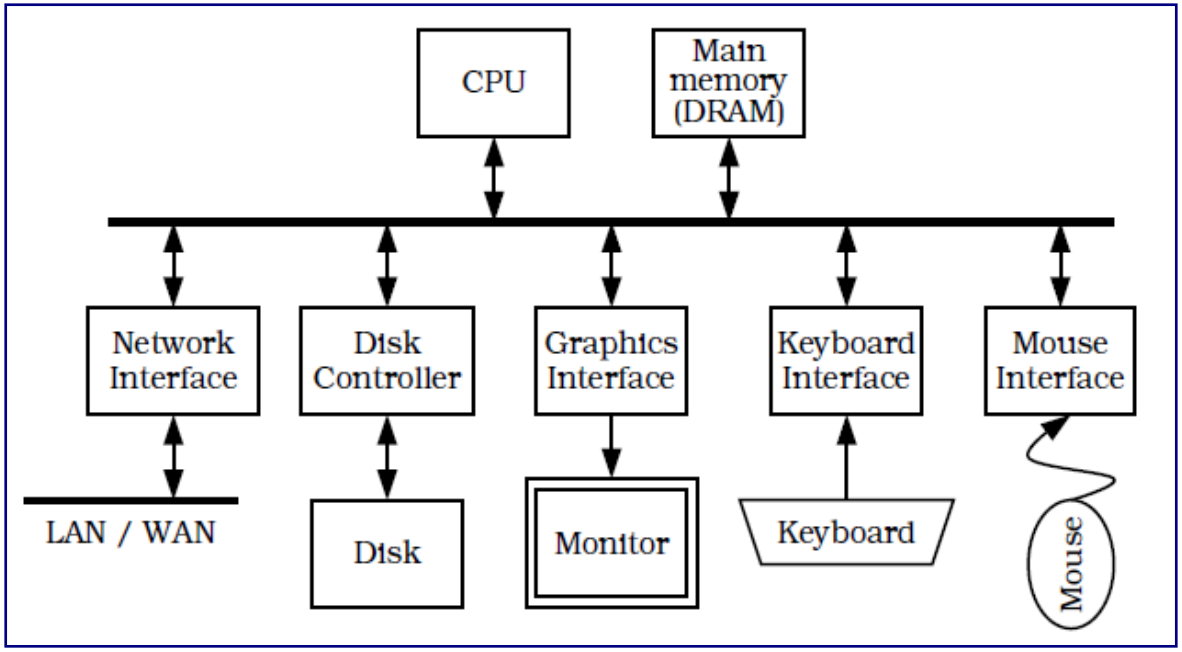


Display 1.1

A little more on ‘Hardware Realization’

With a little more detail, a simple computer looks like this:

1. Programs are loaded from disk and into Main Memory
2. The CPU interprets the 1’s and 0’s in memory as instructions
3. CPU interacts with i/o devices to perform some task

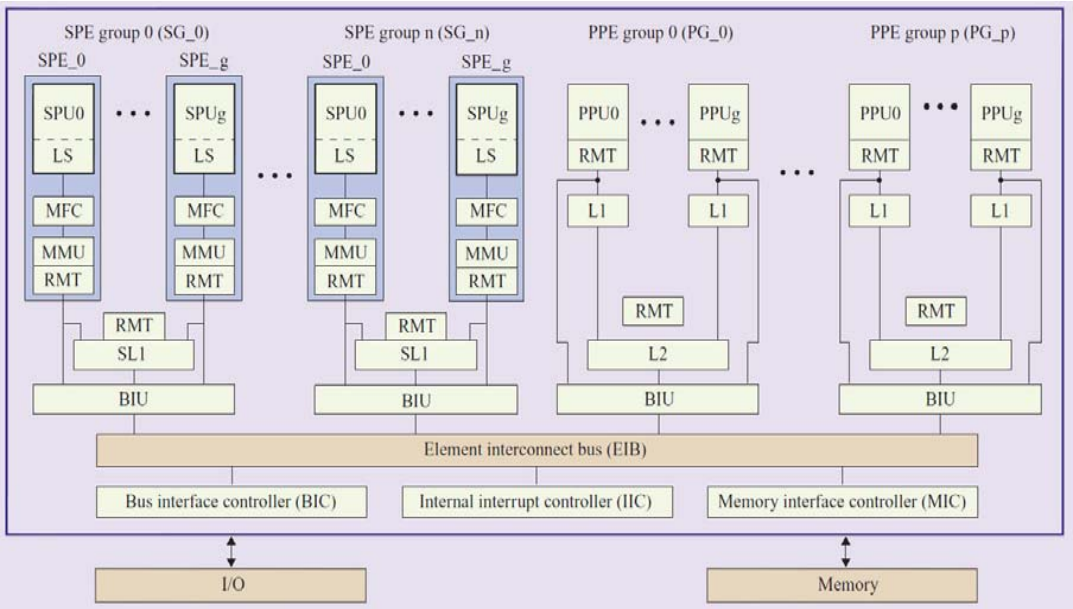


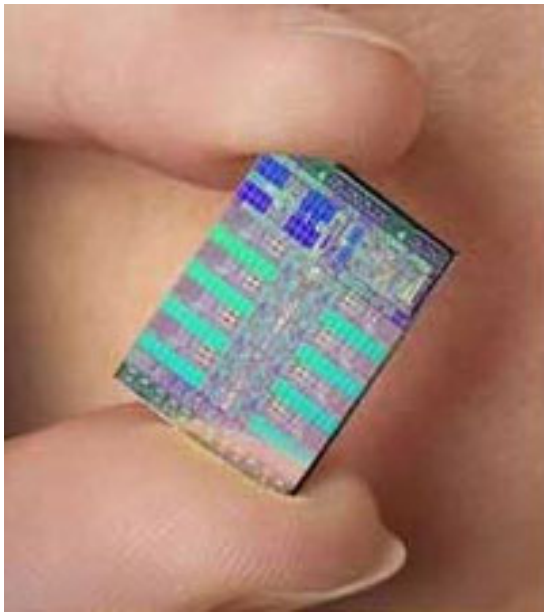
These instructions of 1’s and 0’s are called Machine Language. Definitely NOT C++!

A More Complicated Computer…

Cell Broadband Engine by Sony, Toshiba, and IBM:

- a set of IBM PowerPC 64-bit Processor Units (PPUs)
- a set of Graphical Processing Units (SPUs)





Do you recognize it?

It's the processor inside of the Sony PlayStation 3 and Xbox 360[8]

(PlayStation 4 and Xbox-One are based on AMD "Jaguar" 2x4 CPU Core architecture)[5]

Program Development

The mechanics of developing a program include several activities

- writing the program in a specific programming language (such as C++)
- translating the program into a form that the computer can execute
- investigating and fixing various types of errors that can occur

Software tools can be used to help with all parts of this process

Language Levels

There are four programming language levels

- machine language
- assembly language
- high-level language
- fourth-generation language

Each type of CPU has its own specific *machine language*

The other levels were created to make it easier for a human being to read and write programs

High Level Languages

Common High Level Programming languages include:

C C++ Java Pascal Visual Basic FORTRAN Perl
PHP Lisp Scheme Ada APL C# Python

These High Level languages:

- Resemble human languages
- Are designed to be easy to read and write
- Use more complicated instructions than the CPU can follow
- Must be translated to zeros and ones for the CPU to execute a program

Language Levels

In 1980's, they built CPUs that understood High Level Languages in HW
These machines worked well but were VERY expensive!
Cheaper and more flexible to translate into machine code

A high-level expression and its lover level equivalents:

High-Level Language	Assembly Language	Machine Language

<a + b>	ld [%fp-20], %o0
	ld [%fp-24], %o1
	add %o0, %o1, %o0
		...
		1101 0000 0000 0111
		1011 1111 1110 1000
		1101 0010 0000 0111
		1011 1111 1110 1000
		1001 0000 0000 0000
		...

Compilation

Each type of CPU executes only a particular *machine language*

A program must be translated into machine language before it can be executed

A *compiler* is a software tool which translates *source code* into a specific target language *object code*

Often, that target language is the machine language for a particular CPU type

Syntax and Semantics

The *syntax rules* of a language define how we can put together symbols, reserved words, and identifiers to make a valid program

The *semantics* of a program statement define what that statement means (its purpose or role in a program)

A program that is syntactically correct is not necessarily logically (semantically) correct

A program will always do what we tell it to do, not what we meant to tell it to do

Errors

A program can have three types of errors:

- The compiler will find syntax errors and other basic problems (*compile-time errors*)
- A problem can occur during program execution, such as trying to divide by zero, which causes a program to terminate abnormally (*run-time errors*)
- A program may run, but produce incorrect results, perhaps using an incorrect formula (*logical errors*) or having a race condition (Heizenbugs)

Problem Solving

The purpose of writing a program is to solve a problem

Solving a problem consists of multiple activities

- Understand the problem
- Design a solution (this is the hard part!)
- Review the solution ***before*** you start coding
- Consider alternatives and refine the solution
- Implement the solution (Now start coding)
- Test Individual components first
- Test the solution as a whole

These activities are not purely linear – they *should* overlap and interact

Problem Solving

The key to designing a solution is breaking it down into manageable pieces

When writing software, we design separate pieces that are responsible for certain parts of the solution

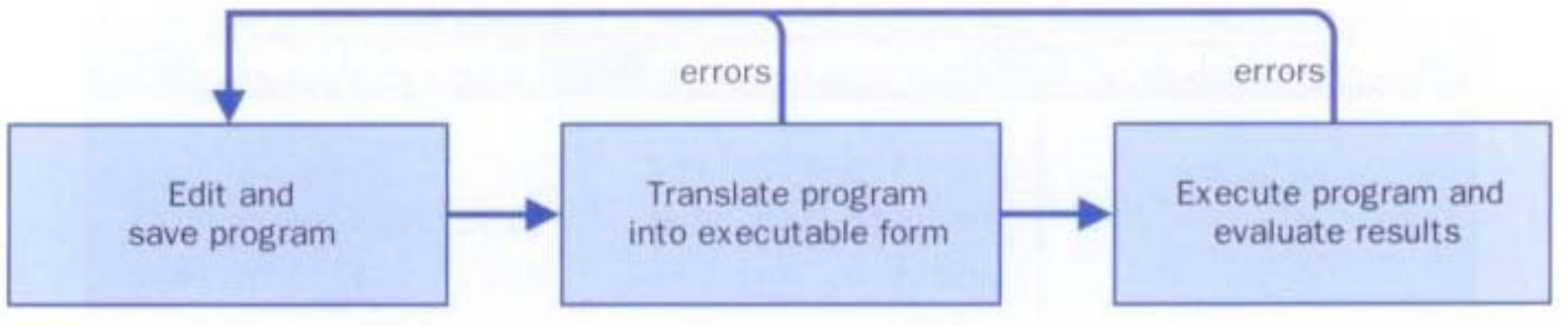
An *object-oriented approach* lends itself to this kind of solution decomposition

We will dissect our solutions into pieces called Classes

Basic Programming Steps

A program is written in an editor, compiled into an executable form, and then executed

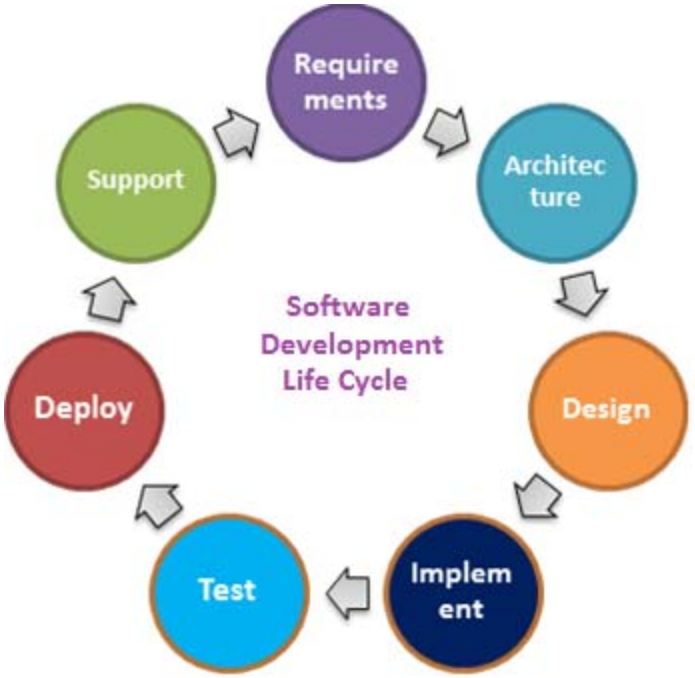
If errors occur during compilation, an executable version is not created



Development Activities

Software development consists of main basic *development activities:*

- establishing the requirements
- creating a design
- implementing the design
- Testing



These steps also are never purely linear and often overlap and interact!

Development Activities

Establish the Requirements:

Software requirements specify *what* a program must accomplish

- A *Functional Specification* document captures what the requirements are

Create a design:

- A *Software Design* document indicates how a program will implement these requirements

Implement the Design:

- *Implementation* is the process of writing the source code that will solve the problem

Testing

- *Testing* is the act of validating that a program will solve the intended problem given all of the constraints under which it must perform

Integrated Development Environments

There are many environments that support the development of C++
Two excellent ones are:

- Eclipse
- NetBeans

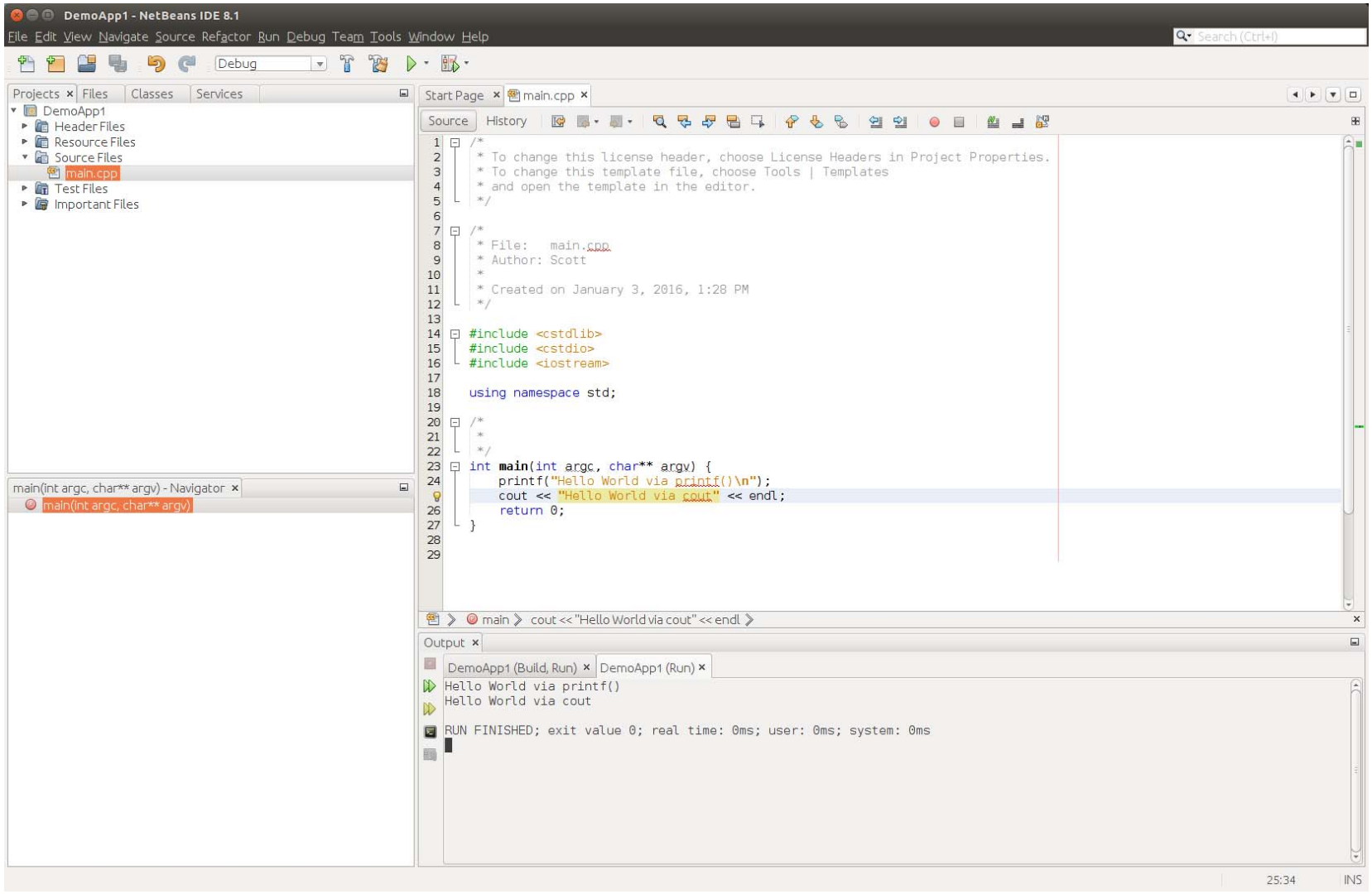
Though the details of these environments differ, the basic compilation and execution process is essentially the same

IDEs might seem complex and intimidating. Don't panic!
You've used similar tools before, you just have to recognize the resemblance. Very similar to a DVD Burner program:

- Invoke the program and it pops up a GUI with several window panes
- A pane exists to move files and assemble your project
- Another pane contains controls for building your project

These IDEs come pre-installed in CSIL

Example IDE Screenshot



Procedural Programming

C++ can be used as a Procedural Programming language

Informally, an algorithm is an ordered sequence of instructions that is guaranteed to solve a specific problem. It is a list that looks something like this:

STEP 1: Do something
STEP 2: Do something
STEP 3: Do something
..
..
STEP N: Stop, you are finished

If you are handed this list and carefully follow its instructions in the order specified, when you reach the end you will have solved the task at hand.

Procedural Operations come in three forms [7]

Sequential Operations
Conditional Operations
Iterative Operations
Plus also need Functional decomposition!

We execute procedural algorithms all the time, we just don't call it that!

For example, here is the procedure to program your DVR:

Step 1 If the clock or calendar are not correctly set, then go to page 9 of the instruction manual, follow the instructions before proceeding to Step 2
Step 2 Place a blank disc into the DVR disc slot
Step 3 Repeat Steps 4 through 7 for each program that you want to record
Step 4 Enter the channel number that you want to record
Step 5 Enter the time that you want recording to start
Step 6 Enter the time that you want recording to stop. This completes the programming of one show
Step 7 If you do not want to record anything else, press the END button
Step 8 Turn off your DVR. Your DVR is now in TIMER mode, ready to record

Object-Oriented Programming

C++ is also an *object-oriented* programming language

An Object contains both State and Methods

Objects can be used effectively to represent real-world entities

For instance, an object might represent a particular employee in a company

Each employee object handles the processing and data management related to that employee

Objects

An object has

- *state* - time-varying data describes characteristics
- *behaviors* - what it can do (or what can be done to it)

The state of a bank account includes its account number and its current balance

The behaviors associated with a bank account include the ability to make deposits and withdrawals

Note that the behavior of an object often changes its state

Classes

An object is defined by a *class*

A *class* is the blueprint of an object

The class uses *methods* to define the behaviors of the object

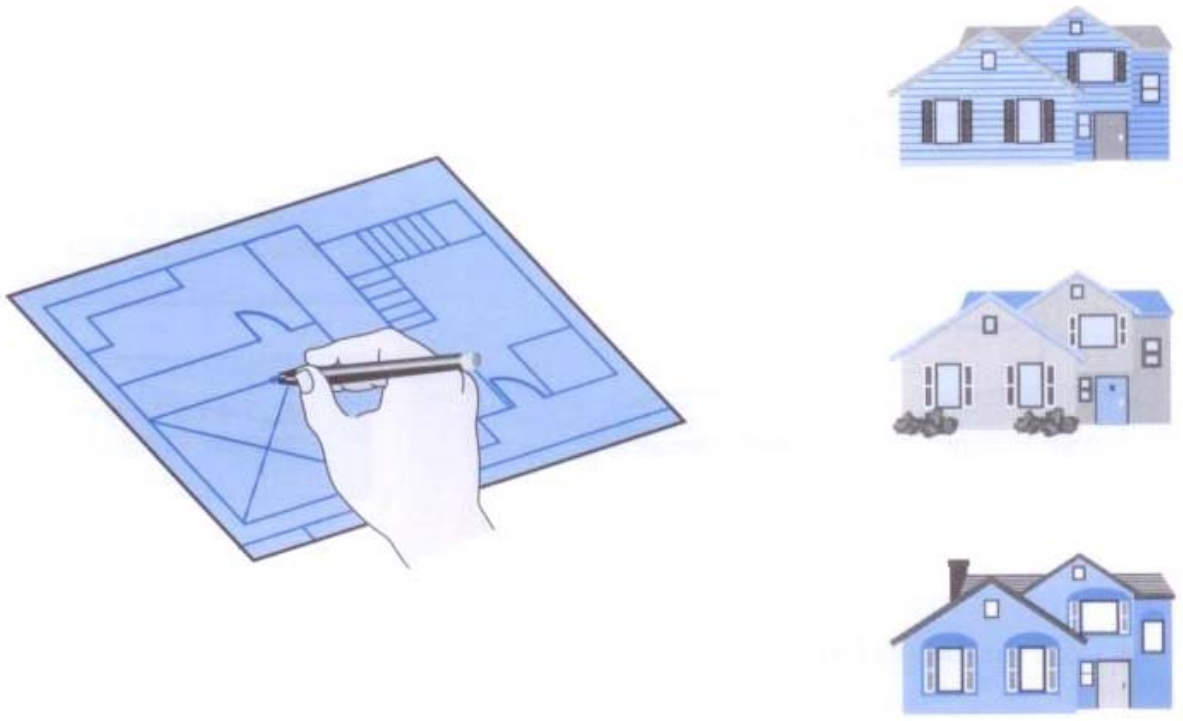
The class that contains the ***main*** method of a C++ program represents the entire program

A class represents a concept, and an object represents an instantiation of that concept

Multiple objects can be created from the same class

Classes and Objects

A class is like a blueprint from which you can create many of the "same" house with different characteristics



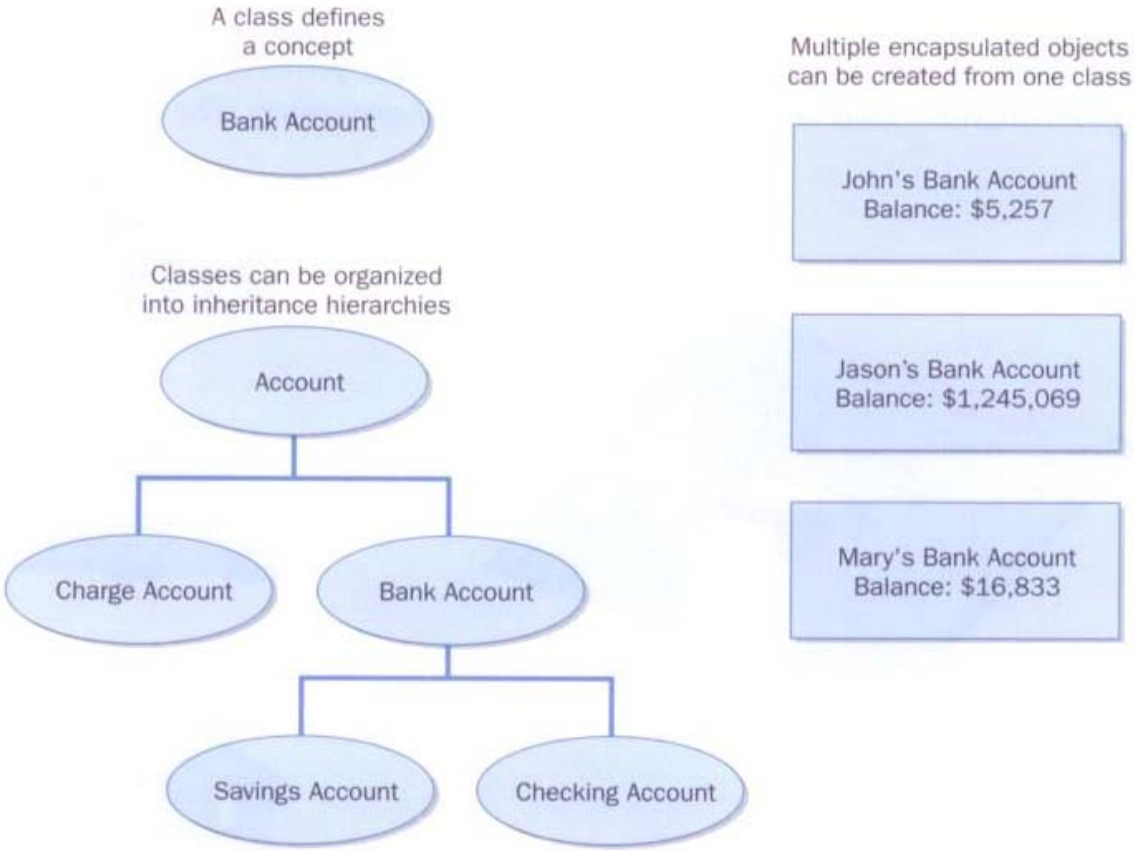
Classes and Objects

An object is *encapsulated*, protecting the data it manages

One class can be used to derive another via *inheritance*

Classes can be organized into *hierarchies*

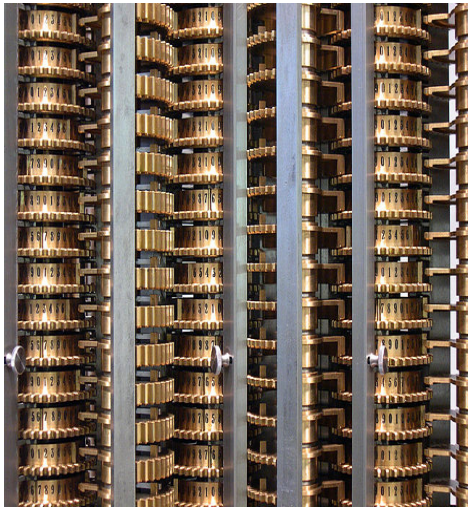
Classes and Objects



History Note – Famous Firsts

First programmable computer

- Difference Engine by Charles Babbage [11]
- Based on J.H.Müller's 1786 idea [10]
- Began in 1822, Not completed in Babbage's life time

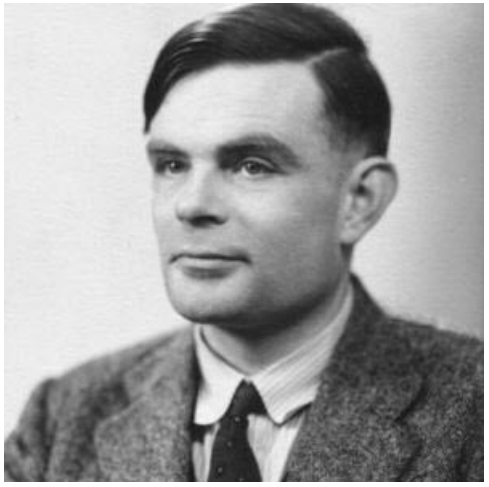


First programmer

- Ada Augusta,
  the Countess of Lovelace
  - Colleague of Babbage

First Formal Model of Computation

- Alan Turing, invented the Turing Machine
- Also broke the Enigma Code during WWII



Historical Note - Flow of Control

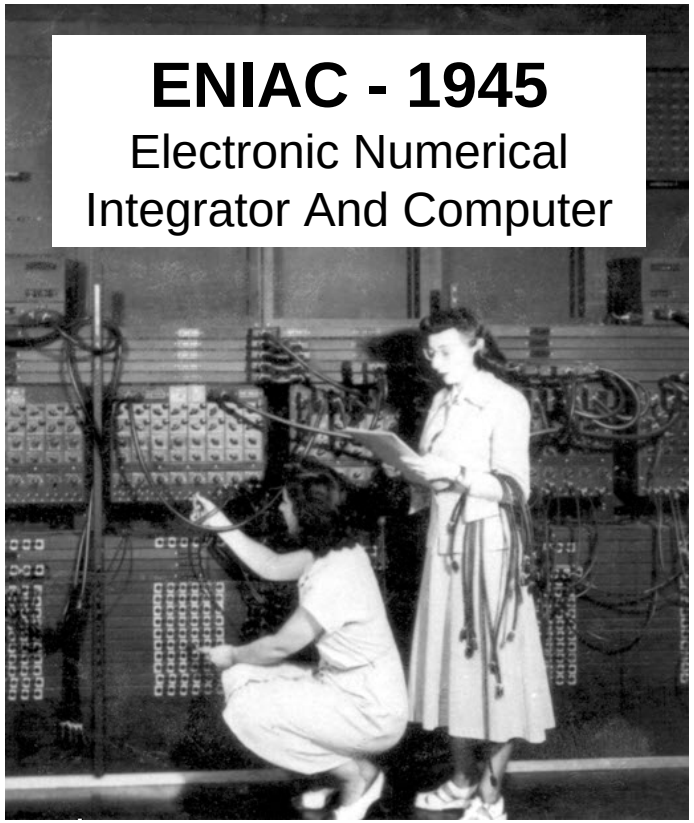
Some programming statements allow us to:

- decide whether or not to execute a particular statement
- execute a statement over and over, repetitively

Decisions based on *boolean expressions* (or *Conditions*) that evaluate to true or false

First Electronic Computer:

- The ENIAC in 1943-1945 [13]
- Programmed via patch cord cables
- Conditional branches invented by the team of 6 women programmers
- They cross-wired data lines with control lines to form conditional branches
- On display at University of Pennsylvania



Summary

Key Things to take away from this presentation:

- Computer systems consist of hardware and software that work in concert to help us solve problems
- All programs must be translated into a CPU's Machine Code before it can be executed
- High-level Languages allow a programmer to ignore CPU specific details of Machine Code and write portable code
- Many different development environments exist to help you create, test, and modify programs
- Syntax rules dictate the form of a program. Semantics dictate meaning of statements.
- Problem Solving involves breaking a problem into smaller pieces
- Each Object has a state defined by attributes, and behaviours defined by methods.
- A Class is a blueprint for creating object instances. Multiple objects can be created from one class definition.

References:

1. Walter Savitch, *Problem Solving with C++*. Pearson, 9th Edition, 2014, ISBN 978-0-13-359174-3
2. Brooks, Frederick P., *No Silver Bullet: Essence and Accidents of Software Engineering*, IEEE Computer, Vol. 20, No. 4 (April 1987) pp. 10-19, http://www.cgl.ucsf.edu/Outreach/pc204/NoSilverBullet
3. TIOBE Index of Programming Language Popularity, http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
4. G. Baker, *The Computing Science 120 Study Guide: Introduction to Computing Science and Programming I*, Fall 2010 Edition, http://www2.cs.sfu.ca/CourseCentral/120/ggbaker/guide/guide
5. J. Rupley, *"JAGUAR" AMD's Next Generation Low Power x86 Core, Aug 28 2012,* http://www.hotchips.org/wp-content/uploads/hc_archives/hc24/HC24-1-Microprocessor/HC24.28.120-Jaguar-Rupley-AMD.pdf
6. *Computer Science Instructional Lab (CSIL) General Information and FAQs page, http://www.sfu.ca/computing/about/support/csil.html*
7. G.M.Schneider and J.L.Gersting, *An Invitation to Computer Science, Cengage Learning,* ISBN-13: 978-1-133-19082-0
8. C. R. Johns, D. A. Brokenshire, *Introduction to the Cell Broadband Engine Architecture,* IBM J.Res.& Dev. Vol.51 No.5 Sept 2007, pp.503-519
9. *Top 10 Most Popular Programming Languages* website, http://www.english4it.com/unit/9/reading
10. Difference Engine, https://en.wikipedia.org/wiki/Difference_engine
11. "London Science Museum's Replica Difference Engine" by Carsten Ullrich. https://commons.wikimedia.org/wiki/File:LondonScienceMuseumsReplicaDifferenceEngine.jpg#/media/File:LondonScienceMuseumsReplicaDifferenceEngine.jpg
12. "Ada Lovelace portrait" by Alfred Edward Chalon - Science & Society Picture Library. Licensed under Public Domain via Commons, https://commons.wikimedia.org/wiki/File:Ada_Lovelace_portrait.jpg#/media/File:Ada_Lovelace_portrait.jpg
13. ENIAC - Electronic Numerical Integrator And Computer http://en.wikipedia.org/wiki/ENIAC
14. Alan Turing and the Turing Machine, https://en.wikipedia.org/wiki/Alan_Turing, https://en.wikipedia.org/wiki/Turing_machine
15. B.Fraser*, Install VMWare Player & Creating an Ubuntu VM (Part 1),* https://youtu.be/TXGREvxPbL4
16. B.Fraser, *Configure a VMWare Player VM (Part 2)*, https://youtu.be/WvWsb5fh2fQ
17. B.Fraser, Installing Netbeans for C++ on Ubuntu VM, https://youtu.be/46SDMxtWTSw
18. How to access (remote login to) the CSIL Linux CPU Server, https://www.sfu.ca/computing/about/support/csil/unix/how-to-use-csil-linux-cpu-server.html

Time for Questions