Lecture Notes

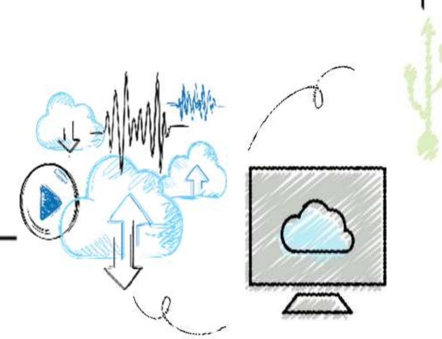
Chapter 1

An Overview of Computers and Programming Languages

ECE 111: Introduction to C and C++ Programming

Instructor: Dr. Shayan (Sean) Taheri

Gannon University (GU)



Personal Information

- Name: Shayan (Sean) Taheri.
- Date of Birth: July/28/1991.
- Current Position: Assistant Professor at Gannon University
- Previous Position: Postdoctoral Fellow at University of Florida.
- Ph.D. Degree: Electrical Engineering from the University of Central Florida.
- M.S. Degree: Computer Engineering from the Utah State University.
- University Profile: https://www.gannon.edu/FacultyProfiles.aspx?profile=taheri001

Objectives (1 of 2)

- In this chapter, you will:
  - Learn about different types of computers
  - Explore the hardware and software components of a computer system
  - Learn about the language of a computer
  - Learn about the evolution of programming languages
  - Examine high-level programming languages
  - Discover what a compiler is and what it does

Objectives (2 of 2)

- Examine a C++ program
- Explore how a C++ program is processed
- Learn what an algorithm is and explore problem-solving techniques
- Become aware of structured design and object-oriented design programming methodologies
- Become aware of Standard C++, ANSI/ISO Standard C++, C++11, and C++14

Introduction

- Without software, a computer is useless
- Software is developed with programming languages
  - C++ is a programming language
- C++ is suited for a wide variety of programming tasks

A Brief Overview of the History of Computers (1 of 3)

- Early calculation devices
  - Abacus
  - Pascaline
  - Leibniz device
  - Jacquard’s weaving looms
  - Babbage machines: difference and analytic engines
  - Hollerith machine

A Brief Overview of the History of Computers (2 of 3)

- Early computer-like machines
  - Mark I
  - Electronic Numerical Integrator and Calculator (ENIAC)
  - Von Neumann architecture
  - Universal Automatic Computer (UNIVAC)
  - Transistors and microprocessors

A Brief Overview of the History of Computers (3 of 3)

- Categories of computers
  - Mainframe computers
  - Midsize computers
  - Micro computers (personal computers)

Elements of a Computer System

- Two main components
  - Hardware
  - Software

Hardware

- Central processing unit (CPU)
- Main memory (MM) or random access memory (RAM)
- Secondary storage
- Input/output devices

Central Processing Unit and Main Memory (1 of 4)

- Central processing unit
  - Brain of the computer
  - Most expensive piece of hardware
  - Operations
    - Carries out arithmetic and logical operations

Central Processing Unit and Main Memory (2 of 4)

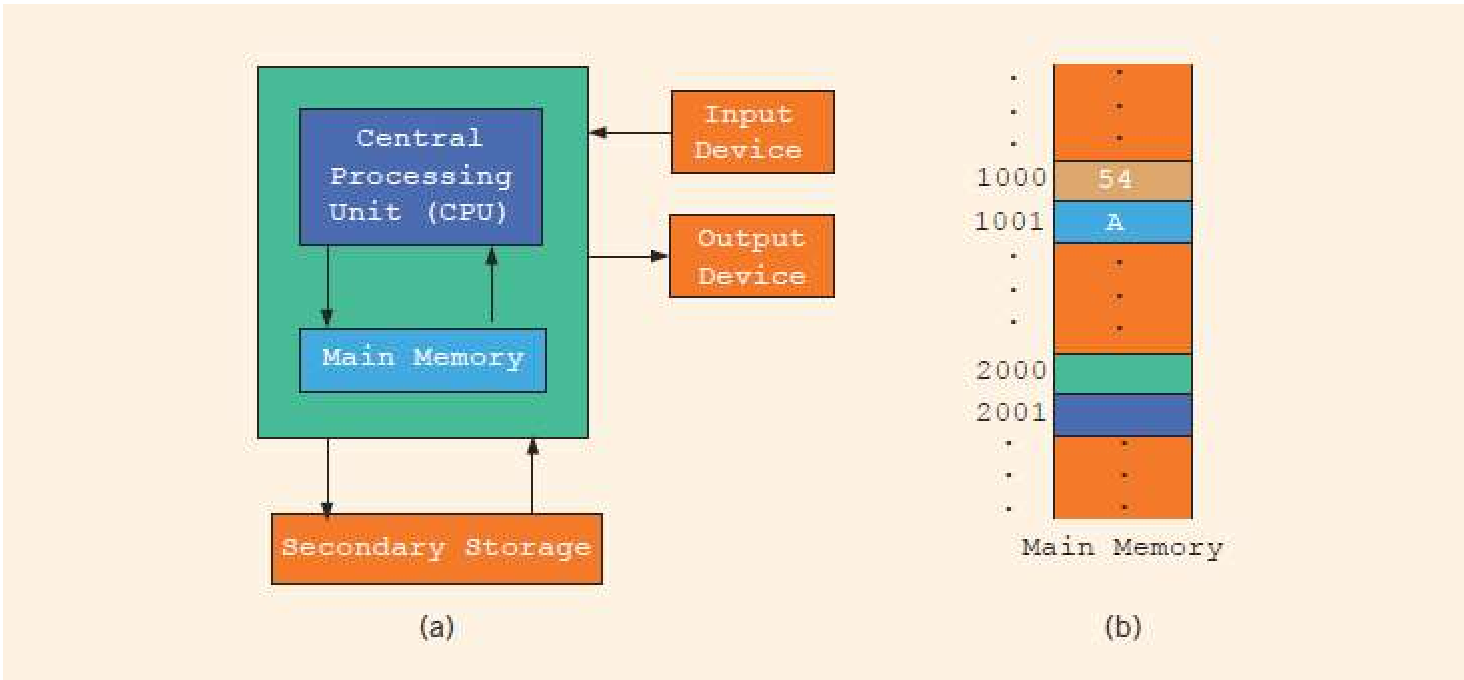


FIGURE 1-1 Hardware components of a computer and main memory

Central Processing Unit and Main Memory (3 of 4)

- Random access memory (or main memory) is directly connected to the CPU
- All programs must be loaded into main memory before they can be executed
- All data must be brought into main memory before it can be manipulated
- When computer power is turned off, everything in main memory is lost

Central Processing Unit and Main Memory (4 of 4)

- Main memory is an ordered sequence of memory cells
  - Each cell has a unique location in main memory, called the address of the cell
- Each cell can contain either a programming instruction or data

Secondary Storage

- Secondary storage: device that stores information permanently
- Examples of secondary storage
  - Hard disks
  - Flash drives
  - CD-ROMs

Input/Output Devices

- Input devices feed data and programs into computers
  - Keyboard
  - Mouse
  - Scanner
  - Camera
  - Secondary storage
- Output devices display results
  - Monitor
  - Printer
  - Secondary storage

Software

- Software are programs written to perform specific tasks
- System programs control the computer
  - Operating system monitors the overall activity of the computer and provides services such as:
    - Memory management
    - Input/output activities
    - Storage management
- Application programs perform a specific task
  - Word processors
  - Spreadsheets
  - Games

The Language of a Computer (1 of 4)

- Analog signals: continuously varying continuous wave forms
- Digital signals: sequences of 0s and 1s
- Machine language: language of a computer
  - A sequence of 0s and 1s
- Binary digit (bit): the digit 0 or 1
- Binary code (binary number): a sequence of 0s and 1s

The Language of a Computer (2 of 4)

- Byte: a sequence of eight bits
- Kilobyte (KB): 2^{10} bytes = 1024 bytes
- ASCII (American Standard Code for Information Interchange)
  - 128 characters
  - **A** is encoded as 1000001 (66th character)
  - The character **3** is encoded as 0110011 (51st character)
- Number systems
  - The decimal system (base 10) is used in our daily life
  - The computer uses the binary (or base 2) number system

The Language of a Computer (3 of 4)

TABLE 1-1 Binary Units

Unit	Symbol	Bits/Bytes
Byte		8 bits
Kilobyte	KB	2^{10} bytes = 1024 bytes
Megabyte	MB	10^{24} KB = 2^{10} KB = 2^{20} bytes = 1,048,576 bytes
Gigabyte	GB	10^{24}MB = 2^{10} MB = 2^{30} bytes = 1,073,741,824 bytes
Terabyte	TB	10^{24} GB = 2^{10} GB = 2^{40} bytes = 1,099,511,627,776 bytes
Petabyte	PB	10^{24} TB = 2^{10} TB = 2^{50} bytes = 1,125,899,906,842,624 bytes
Exabyte	EB	10^{24} PB = 2^{10} PB = 2^{60} bytes = 1,152,921,504,606,846,976 bytes
Zettabyte	ZB	10^{24} EB5 2^{10} EB = 270 bytes = 1,180,591,620,717,411,303,424 bytes

The Language of a Computer (4 of 4)

- Unicode is another coding scheme
  - 65,536 characters
  - Two bytes (16 bits) to store a character

The Evolution of Programming Languages (1 of 3)

- Early computers were programmed in machine language
- To calculate wages = rate * hours in machine language:

```
100100 010001    //Load
100110 010010    //Multiply
100010 010011    //Store
```

The Evolution of Programming Languages (2 of 3)

- Assembly language instructions are mnemonic
  - Instructions are written in an easy-to-remember form
- An assembler translates a program written in assembly language into machine language
- Using assembly language instructions, **`wages = rate * hours`** can be written as:

```
LOAD rate
MULT hours
STOR wages
```

The Evolution of Programming Languages (3 of 3)

- High-level languages include Basic, FORTRAN, COBOL, C, C++, C#, Java, and Python
- Compiler: translates a program written in a high-level language into machine language
- In C++, the weekly wages equation can be written as:

```
wages = rate * hours;
```

Processing a C++ Program (1 of 4)

```
#include <iostream>
using namespace std;
int main()
{
    cout << "My first C++ program." << endl;
    return 0;
}
```

Sample Run:

```
My first C++ program.
```

Processing a C++ Program (2 of 4)

- Steps needed to process a C++ program
  1. Use a text editor to create the source code (source program) in C++
  2. Include preprocessor directives
     - Begin with the symbol # and are processed by the preprocessor
  3. Use the compiler to:
     - Check that the program obeys the language rules
     - Translate the program into machine language (**object program**)
  4. Use an integrated development environment (IDE) to develop programs in a high-level language
     - Programs such as mathematical functions are available
     - The library contains prewritten code you can use
     - A linker combines object program with other programs in the library to create executable code
  5. The loader loads executable program into main memory
  6. The last step is to execute the program

Processing a C++ Program (3 of 4)

- IDEs are quite user friendly
  - Compiler identifies the syntax errors and also suggests how to correct them
  - Build or Rebuild is a simple command that links the object code with the resources used from the IDE

Processing a C++ Program (4 of 4)

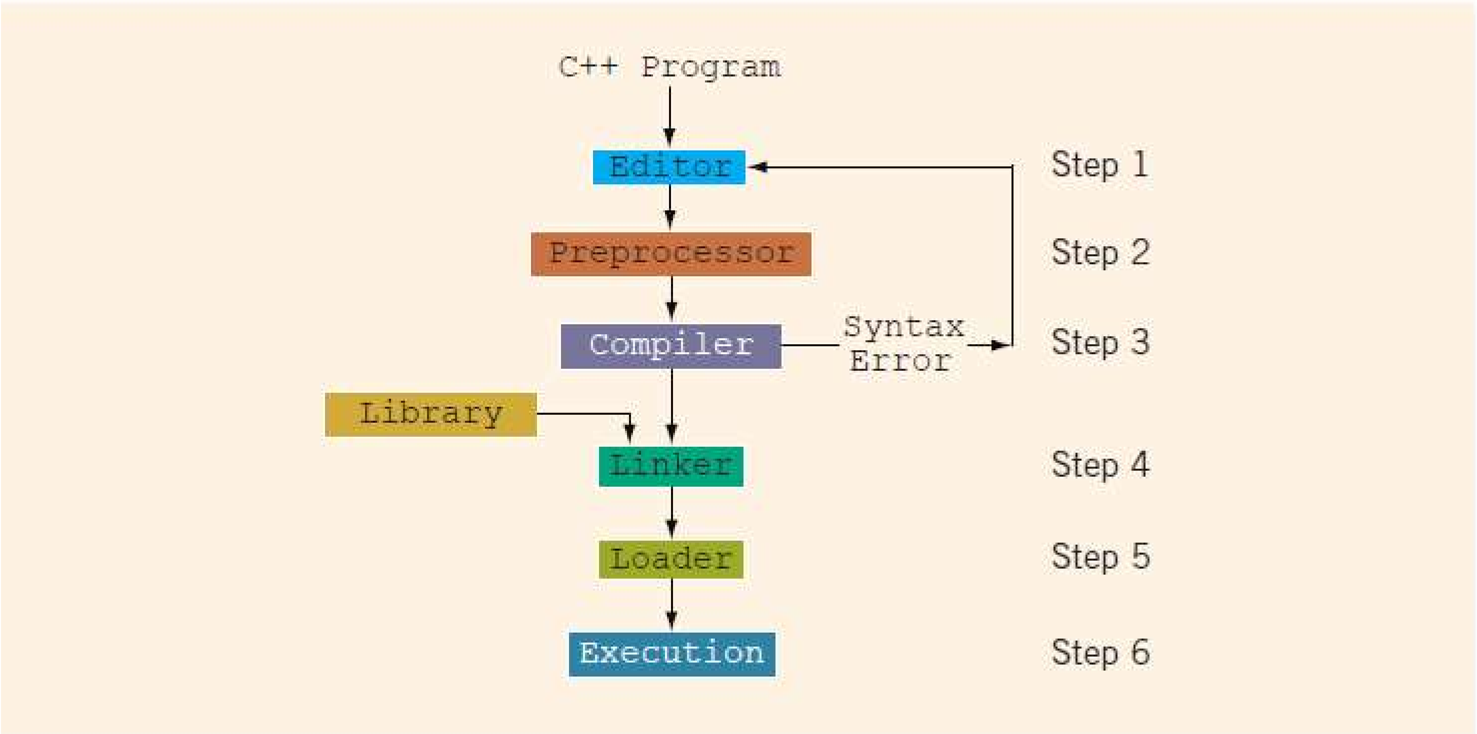


FIGURE 1-2 Processing a C++ program

Programming with the Problem Analysis–Coding–Execution Cycle

- Programming is a process of problem solving
- An <u>algorithm</u> is a step-by-step problem-solving process
  - A solution is achieved in a finite amount of time

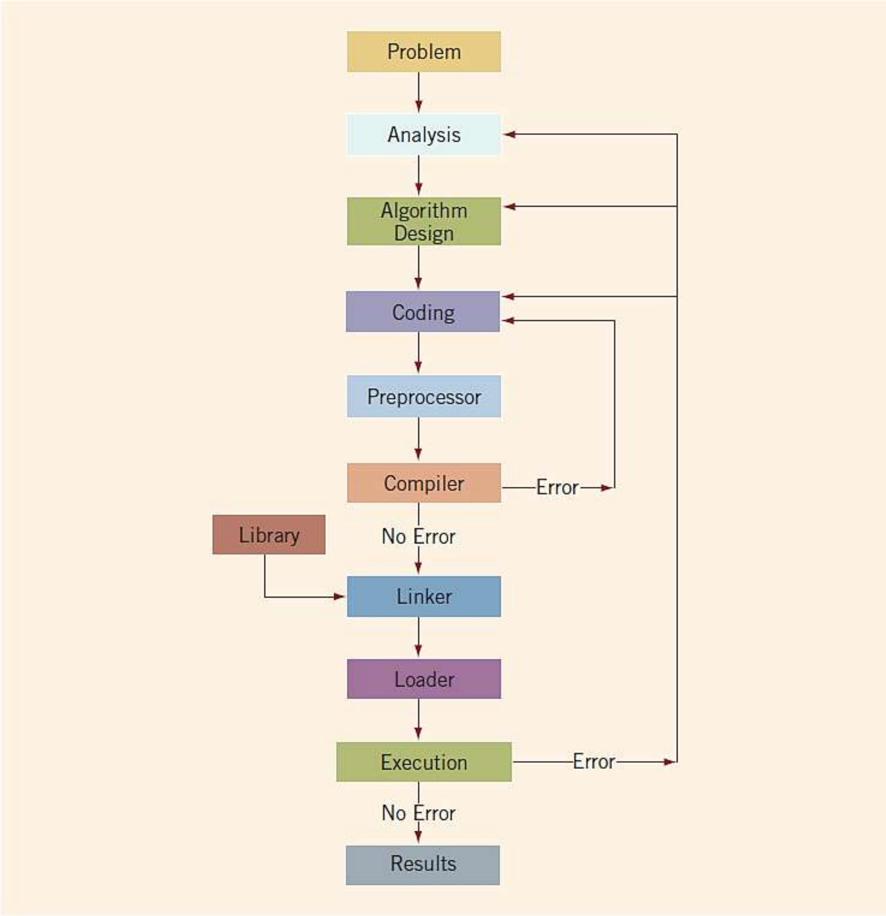


FIGURE 1-3 Problem analysis–coding–execution cycle

The Problem Analysis–Coding–Execution Cycle (1 of 5)

- Step 1: Analyze the problem
  - Outline the problem and its requirements
  - Design steps (algorithm) to solve the problem
- Step 2: Implement the algorithm
  - Implement the algorithm in code
  - Verify that the algorithm works
- Step 3: Maintain the program
  - Use and modify the program if the problem domain changes

The Problem Analysis–Coding–Execution Cycle (2 of 5)

- Analyze the problem using these steps:
  - Step 1: Thoroughly understand the problem and all requirements
  - Step 2: Understand the problem requirements
    - Does program require user interaction?
    - Does program manipulate data?
    - What is the output?
  - Step 3: If complex, divide the problem into subproblems
    - Analyze and design algorithms for each subproblem
- Check the correctness of algorithm
  - Test the algorithm using sample data
  - Some mathematical analysis might be required

The Problem Analysis–Coding–Execution Cycle (3 of 5)

- Once the algorithm is designed and correctness is verified
  - Write the equivalent code in high-level language
- Enter the program using a text editor

The Problem Analysis–Coding–Execution Cycle (4 of 5)

- Run code through the compiler
- If compiler generates errors
  - Look at code and remove errors
  - Run code again through compiler
- If there are no syntax errors
  - Compiler generates equivalent machine code
- Link machine code with the system's resources
  - Performed by the linker

The Problem Analysis–Coding–Execution Cycle (5 of 5)

- Once compiled and linked, the loader can place program into main memory for execution
- The final step is to execute the program
- Compiler guarantees that the program follows the rules of the language
  - Does not guarantee that the program will run correctly

Example 1-1 (1 of 2)

- Design an algorithm to find the perimeter and area of a rectangle
- The perimeter and area of the rectangle are given by the following formulas:

```
perimeter = 2 * (length + width)
area = length * width
```

Example 1-1 (2 of 2)

- Algorithm
  - Get the length of the rectangle
  - Get the width of the rectangle
  - Find the perimeter with this equation:

    ```
    perimeter = 2 * (length + width)
    ```
  - Find the area with this equation:

    ```
    area = length * width
    ```

Example 1-5 (1 of 4)

- Calculate each student's grade
  - There are 10 students in a class
  - Each student has taken five tests
  - Each test is worth 100 points
- Design algorithms to:
  - Calculate the grade for each student and class average
  - Find the average test score
  - Determine the grade
- Use the provided data: students' names and test scores

Example 1-5 (2 of 4)

- Algorithm to determine the average test score
  - Get the five test scores
  - Add the five test scores
    - The sum of the test scores is represented by **`sum`**
    - Suppose **`average`** stands for the average test score:

```
average = sum / 5;
```

Example 1-5 (3 of 4)

- Algorithm to determine the grade:

```
if average is greater than or equal to 90
    grade = A
otherwise
    if average is greater than or equal to 80 and less than 90
       grade = B
otherwise
    if average is greater than or equal to 70 and less than 80
        grade = C
otherwise
    if average is greater than or equal to 60 and less than 70
       grade = D
otherwise
    grade = F
```

Example 1-5 (4 of 4)

- Main algorithm is presented below:

  1. **`totalAverage = 0;`**
  2. Repeat the following for each student:
     - Get student's name
     - Use the algorithm to find the average test score
     - Use the algorithm to find the grade
  3. Update **`totalAverage`** by adding current student's average test score
  4. Determine the class average as follows:
     `classAverage = totalAverage / 10`

Programming Methodologies

- Two popular approaches to programming design
  - Structured
  - Object-oriented

Structured Programming

- Structured design
  - Involves dividing a problem into smaller subproblems
- Structured programming
  - Involves implementing a structured design
- The structured design approach is also called:
  - Top-down (or bottom-up) design
  - Stepwise refinement
  - Modular programming

Object-Oriented Programming (1 of 3)

- Object-oriented design (OOD)
  - Identify components called objects
  - Determine how objects interact with each other
- Specify relevant data and possible operations to be performed on that data
- Each object consists of data and operations on that data

Object-Oriented Programming (2 of 3)

- An object combines data and operations on the data into a single unit
- A programming language that implements OOD is called an object-oriented programming (OOP) language
- To design and use objects, you must learn how to:
  - Represent data in computer memory
  - Manipulate data
  - Implement operations

Object-Oriented Programming (3 of 3)

- To create operations:
  - Write algorithms and implement them in a programming language
  - Use functions to implement algorithms
- Learn how to combine data and operations on the data into a single unit called a class
- C++ was designed to implement OOD
- OOD is used with structured design

ANSI/ISO Standard C++

- C++ evolved from C
- C++ designed by Bjarne Stroustrup at Bell Laboratories in early 1980s
  - Many different C++ compilers were available
- C++ programs were not always portable from one compiler to another
- In mid-1998, ANSI/ISO C++ language standards were approved
- Second standard, called C++11, was approved in 2011

Quick Review (1 of 3)

- A computer is an electronic device that can perform arithmetic and logical operations
- A computer system has hardware and software components
  - The central processing unit (CPU) is the brain
  - Primary storage (MM) is volatile; secondary storage (e.g., disk) is permanent
  - The operating system monitors overall activity of the computer and provides services
  - There are various kinds of languages

Quick Review (2 of 3)

- Compiler: translates high-level language into machine code
- Algorithm:
  - Step-by-step problem-solving process
  - Arrives at a solution in a finite amount of time
- Problem-solving process
  1. Analyze the problem and design an algorithm
  2. Implement the algorithm in code
  3. Maintain the program

Quick Review (3 of 3)

- Structured design
  - Problem is divided into smaller subproblems
  - Each subproblem is solved
  - Combine solutions to all subproblems
- Object-oriented design (OOD) program: a collection of interacting objects
  - Object: data and operations on those data