



Lecture 1:

An Overview of Computers and Programming Languages

Objectives

- In this lecture, 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

Objectives (cont'd.)

- Discover what a compiler is and what it does
- 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++, and C++11

Introduction

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

A Brief Overview of the History of Computers

- 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 (cont'd.)

- Early computer-like machines
 - Mark I
 - ENIAC
 - Von Neumann architecture
 - UNIVAC
 - Transistors and microprocessors

A Brief Overview of the History of Computers (cont'd.)

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

Elements of a Computer System

- Hardware
 - CPU
 - Main memory
 - Secondary storage
 - Input/Output devices
- Software

Hardware

- CPU
- Main memory: RAM
- Input/output devices
- Secondary storage

Central Processing Unit and Main Memory

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

Central Processing Unit and Main Memory (cont'd.)

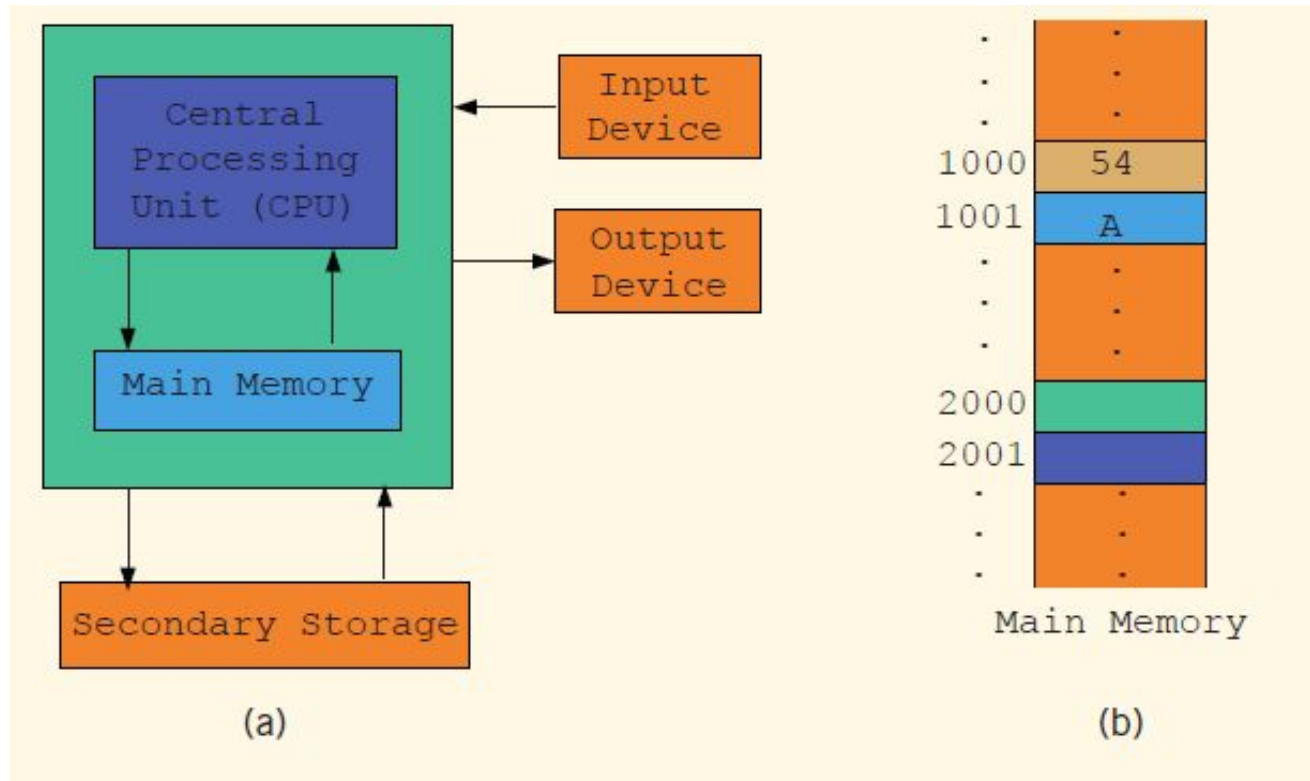


FIGURE 1-1 Hardware components of a computer and main memory

Central Processing Unit and Main Memory (cont'd.)

- Random access memory
 - 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 (cont'd.)

- 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
 - Floppy disks
 - Zip disks
 - CD-ROMs
 - Tapes

Input/Output Devices

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

Software

- Software: programs that do 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

- Analog signals: 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 (cont'd.)

- 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)
 - 3 is encoded as 0110011

The Language of a Computer (cont'd.)

TABLE 1-1 Binary Units

Unit	Symbol	Bits/Bytes
Byte		8 bits
Kilobyte	KB	2^{10} bytes = 1024 bytes
Megabyte	MB	$1024 \text{ KB} = 2^{10} \text{ KB} = 2^{20} \text{ bytes} = 1,048,576 \text{ bytes}$
Gigabyte	GB	$1024 \text{ MB} = 2^{10} \text{ MB} = 2^{30} \text{ bytes} = 1,073,741,824 \text{ bytes}$
Terabyte	TB	$1024 \text{ GB} = 2^{10} \text{ GB} = 2^{40} \text{ bytes} = 1,099,511,627,776 \text{ bytes}$
Petabyte	PB	$1024 \text{ TB} = 2^{10} \text{ TB} = 2^{50} \text{ bytes} = 1,125,899,906,842,624 \text{ bytes}$
Exabyte	EB	$1024 \text{ PB} = 2^{10} \text{ PB} = 2^{60} \text{ bytes} = 1,152,921,504,606,846,976 \text{ bytes}$
Zettabyte	ZB	$1024 \text{ EB} = 2^{10} \text{ EB} = 2^{70} \text{ bytes} = 1,180,591,620,717,411,303,424 \text{ bytes}$

The Language of a Computer (cont'd.)

- EBCDIC
 - Used by IBM
 - 256 characters
- Unicode
 - 65536 characters
 - Two bytes are needed to store a character
 - E.g. 1 and 8 are represented as 0001 and 0008

The Evolution of Programming Languages

- 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 (cont'd.)

- Assembly language instructions are mnemonic
- Assembler: translates a program written in assembly language into machine language
- Using assembly language instructions, $\text{wages} = \text{rate} \cdot \text{hours}$ can be written as:

LOAD rate

MULT hour

STOR wages

The Evolution of Programming Languages (cont'd.)

- High-level languages include Basic, FORTRAN, COBOL, Pascal, C, C++, C#, and Java
- Compiler: translates a program written in a high-level language into machine language
- The equation $wages = rate \cdot hours$ can be written in C++ as:

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

Processing a C++ Program

```
#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 (cont'd.)

- To execute a C++ program:
 - Use an editor to create a source program in C++
 - E.g . Codeblocks,
 - Preprocessor directives begin with # and are processed by the preprocessor
 - Use the compiler to:
 - Check that the program obeys the language rules
 - Translate into machine language (object program)

Processing a C++ Program (cont'd.)

- To execute a C++ program (cont'd.):
 - Linker:
 - Combines object program with other programs provided by the SDK to create executable code
 - Library: contains prewritten code you can use
 - Loader:
 - Loads executable program into main memory
 - The last step is to execute the program
- Some IDEs do all this with a Build or Rebuild command

Processing a C++ Program (cont'd.)

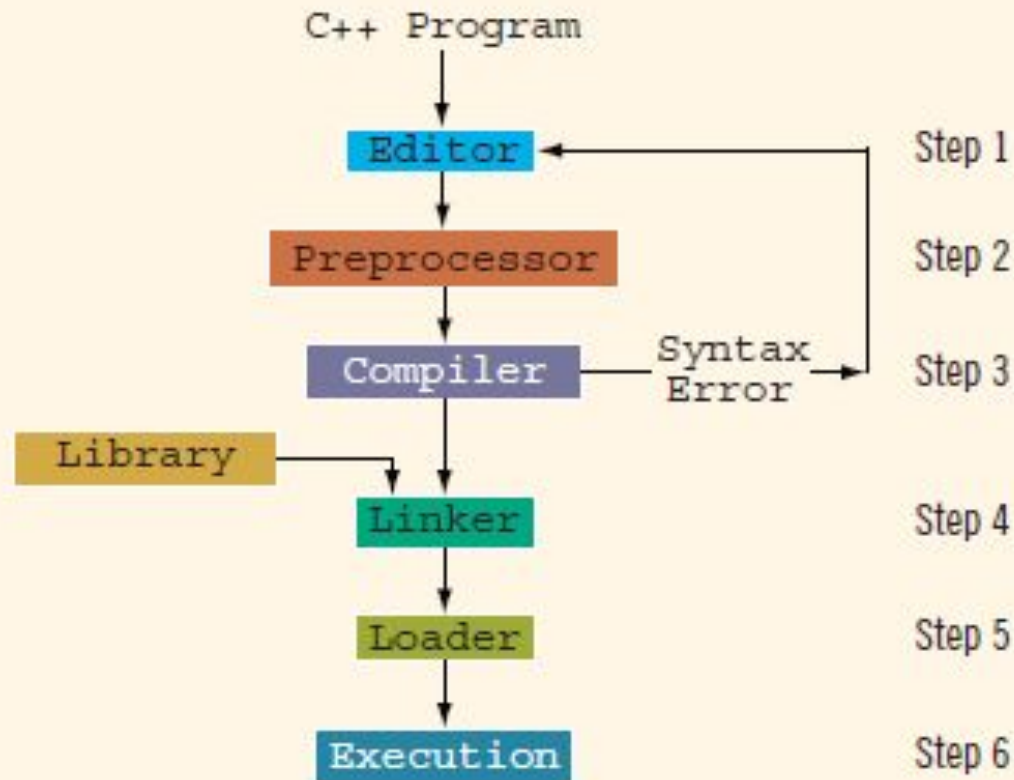


FIGURE 1-2 Processing a C++ program

Programming with the Problem Analysis–Coding–Execution Cycle

- Algorithm:
 - Step-by-step problem-solving process
 - Solution achieved in finite amount of time
- Programming is a process of problem solving

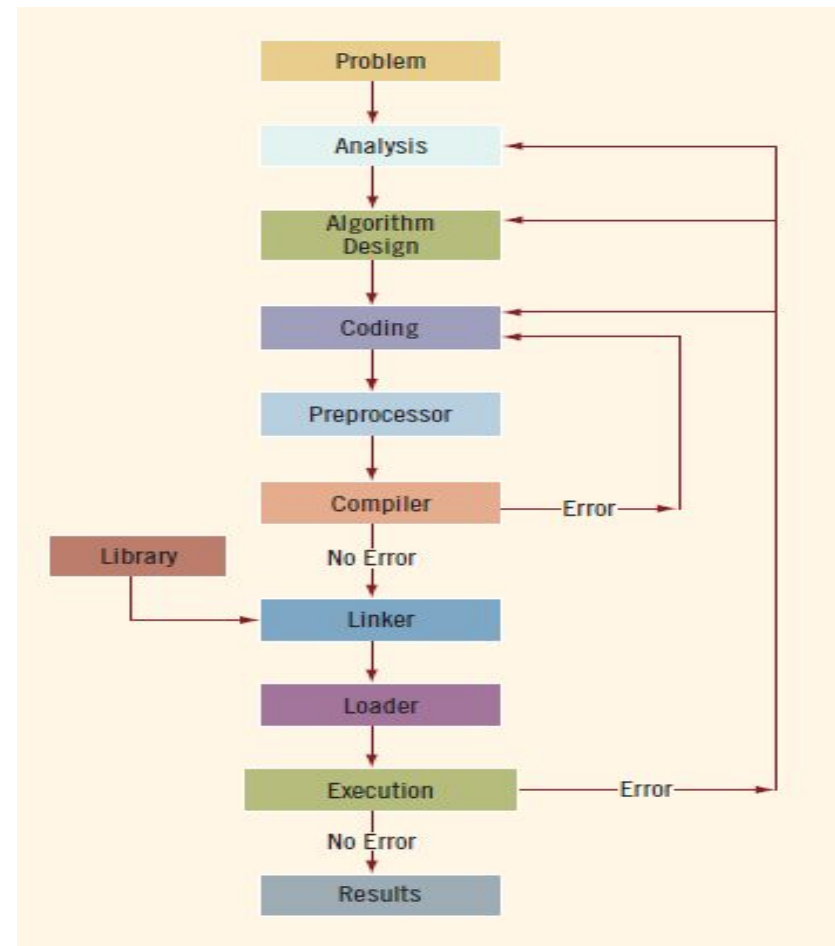


FIGURE 1-3 Problem analysis–coding–execution cycle

Programming with the Problem Analysis–Coding–Execution Cycle

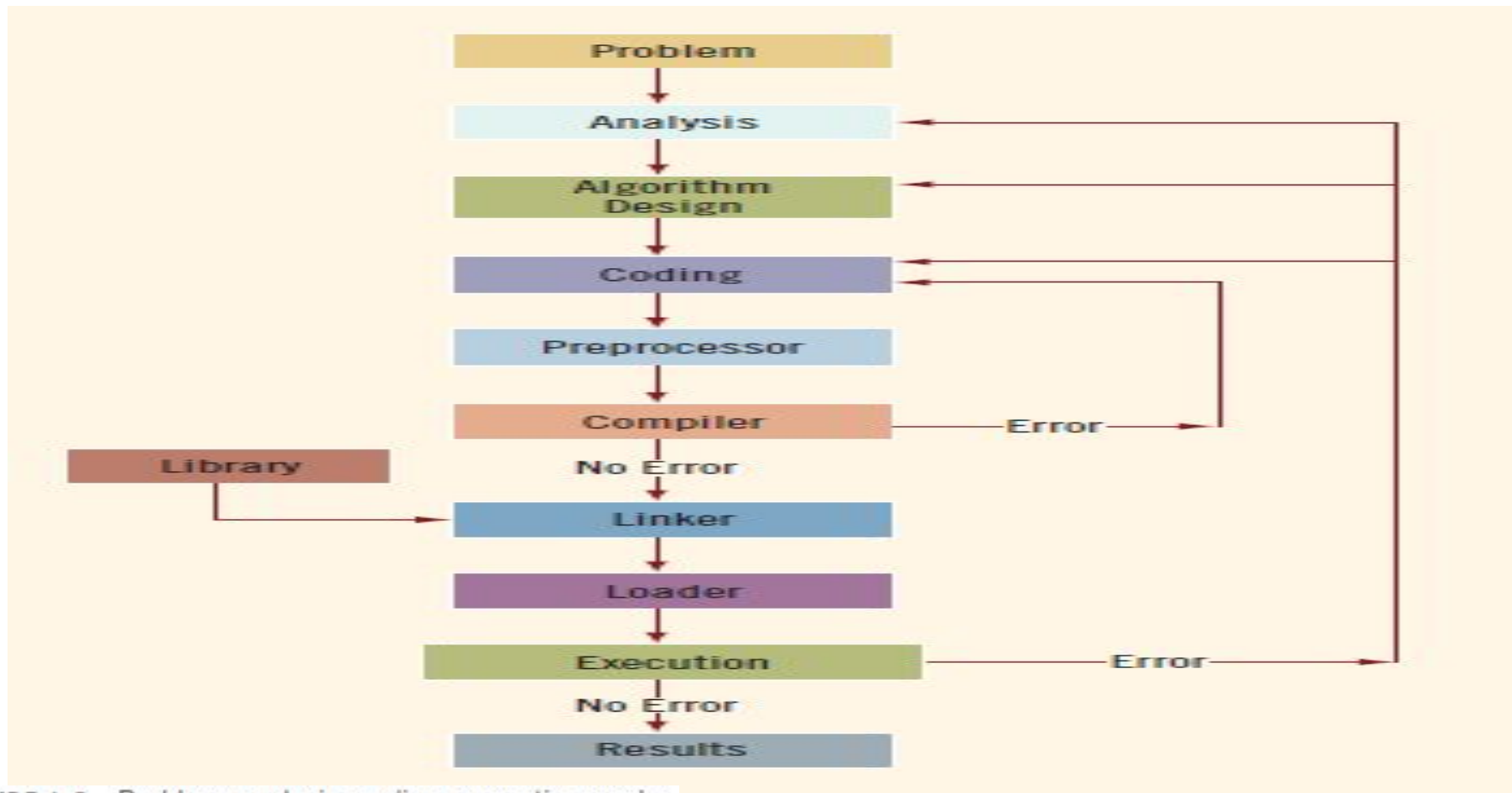


FIGURE 1-3 Problem analysis–coding–execution cycle

The Problem

Analysis–Coding–Execution Cycle

(cont'd)

- 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
 - Use and modify the program if the problem domain changes

The Problem

Analysis–Coding–Execution Cycle

(cont'd)

- Thoroughly understand the problem and all requirements
 - Does program require user interaction?
 - Does program manipulate data?
 - What is the output?
- If the problem is complex, divide it into subproblems
 - Analyze and design algorithms for each subproblem
- Check the correctness of algorithm
 - Can test using sample data
 - Some mathematical analysis might be required

The Problem

Analysis–Coding–Execution Cycle

(cont'd)

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

The Problem

Analysis–Coding–Execution Cycle

(cont'd)

- Run code through 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
- Linker links machine code with system resources

The Problem

Analysis–Coding–Execution Cycle

(cont'd)

- Once compiled and linked, 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

- 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 (cont'd.)

- Algorithm:
 - Get length of the rectangle
 - Get width of the rectangle
 - Find the perimeter using the following equation:
$$\text{perimeter} = 2 * (\text{length} + \text{width})$$
 - Find the area using the following equation:
$$\text{area} = \text{length} * \text{width}$$

Example 1-5

- Calculate each student's grade
 - 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
- Data: students' names; test scores

Example 1-5 (cont'd.)

- Algorithm to determine the average test score:
 - Get the five test scores
 - Add the five test scores
 - Suppose `sum` stands for the sum of the test scores
 - Suppose `average` stands for the average test score:
 - `average = sum / 5;`

Example 1-5 (cont'd.)

- 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 (cont'd.)

- Main algorithm is as follows:
 - `totalAverage = 0;`
 - 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
 - Update `totalAverage` by adding current student's average test score
 - Determine the class average as follows:
 - `classAverage = totalAverage / 10`

Programming Methodologies

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

Structured Programming

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

Object-Oriented Programming

- 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 (cont'd.)

- 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
- Must learn how to represent data in computer memory, how to manipulate data, and how to implement operations

Object-Oriented Programming (cont'd.)

- 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 an object
- 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 approved in 2011

Summary

- Computer: electronic device that can perform arithmetic and logical operations
- Computer system has hardware/software
 - Central processing unit (CPU): brain
 - Primary storage (MM) is volatile; secondary storage (e.g., disk) is permanent
 - Operating system monitors overall activity of the computer and provides services
 - Various kinds of languages

Summary (cont'd.)

- Compiler: translates high-level language into machine code
- Algorithm: step-by-step problem-solving process; solution in finite amount of time
- Problem-solving process has three steps:
 - Analyze problem and design an algorithm
 - Implement the algorithm in code
 - Maintain the program

Summary (cont'd.)

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