The Zen of C++

2nd Edition

Adnan Zejnilovic

Chapter 1: Intoduction

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.)

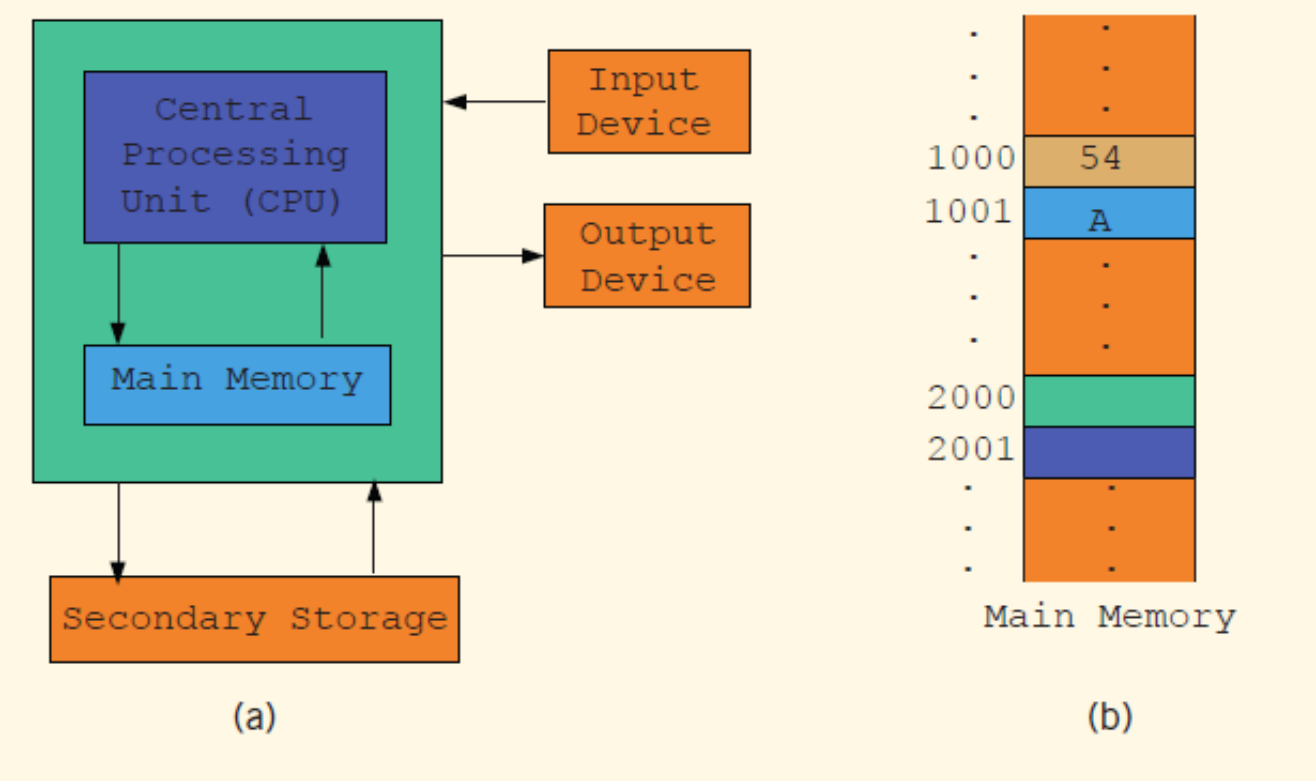


FIGURE 1-1 Hardware components of a computer and main memory

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

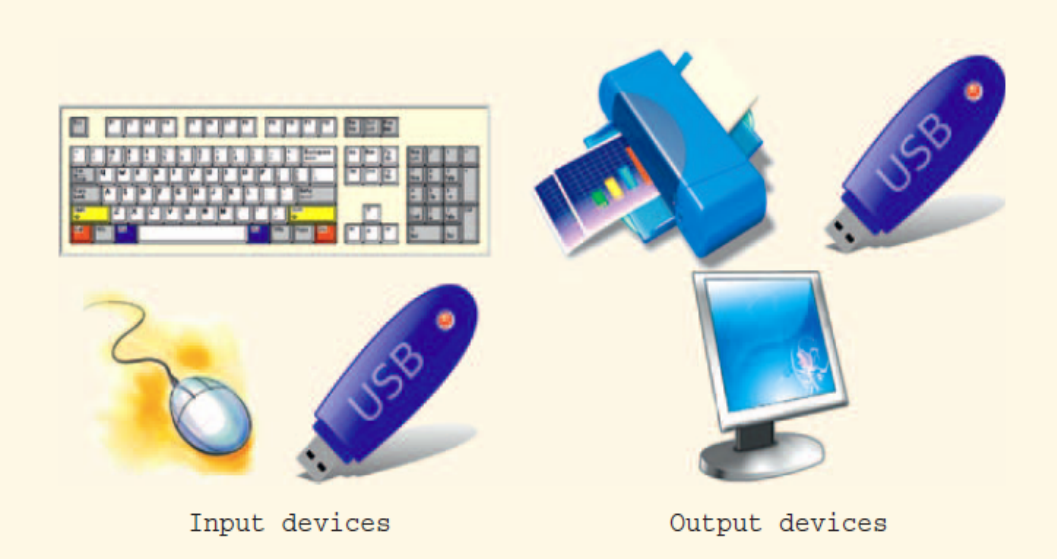


FIGURE 1-2 Some input and output devices

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.)

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

TABLE 1-1 Binary Units

The Language of a Computer (cont'd.)

- EBCDIC
  - Used by IBM
  - 256 characters
- Unicode
  - 65536 characters
  - Two bytes are needed to store a character

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

Assembly Language	Machine Language
LOAD	100100
STOR	100010
MULT	100110
ADD	100101
SUB	100011

TABLE 1-2 Examples of Instructions in Assembly Language and Machine Language

The Evolution of Programming Languages (cont'd.)

- Using assembly language instructions, `wages = rate • 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 • hours` can be written in C++ as:

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

C++ Programming Process

- Write the source code
  - Notepad
  - IDEs
- Compile
  - Command Line
  - IDEs
- Execute

Compilation

Command Line Compilers

- GNU C++ (GCC)
- Pros: Free
- Cons:
  - Need to remember commands
  - Error prone due to typing

IDEs

- MS Visual Studio
- Code Blocks
- DevC++
- Pros:
  - No need to memorize commands – only a few buttons
- Cons:
  - Each IDE is specific
  - Learning curve

Your 1st C++ Program

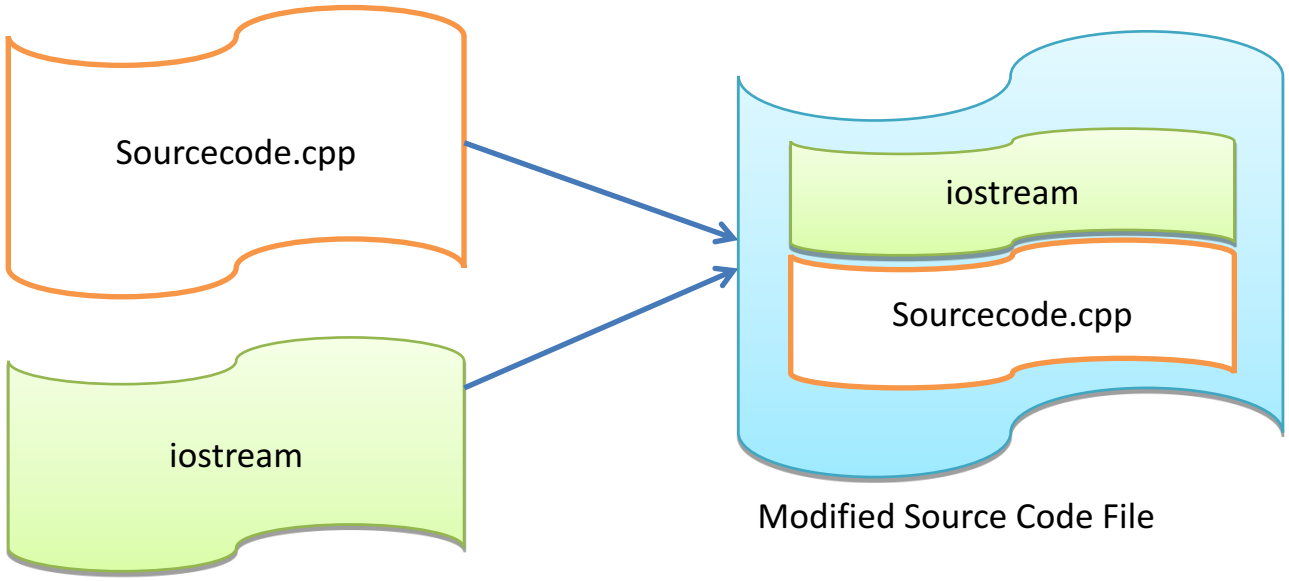
```
1   // my first C++ program -Hello World
2   #include <iostream>
3
4   using namespace std;
5
6   int main()
7   {
8       cout << "Hello world!" << endl;
9       return 0;
10  }
```

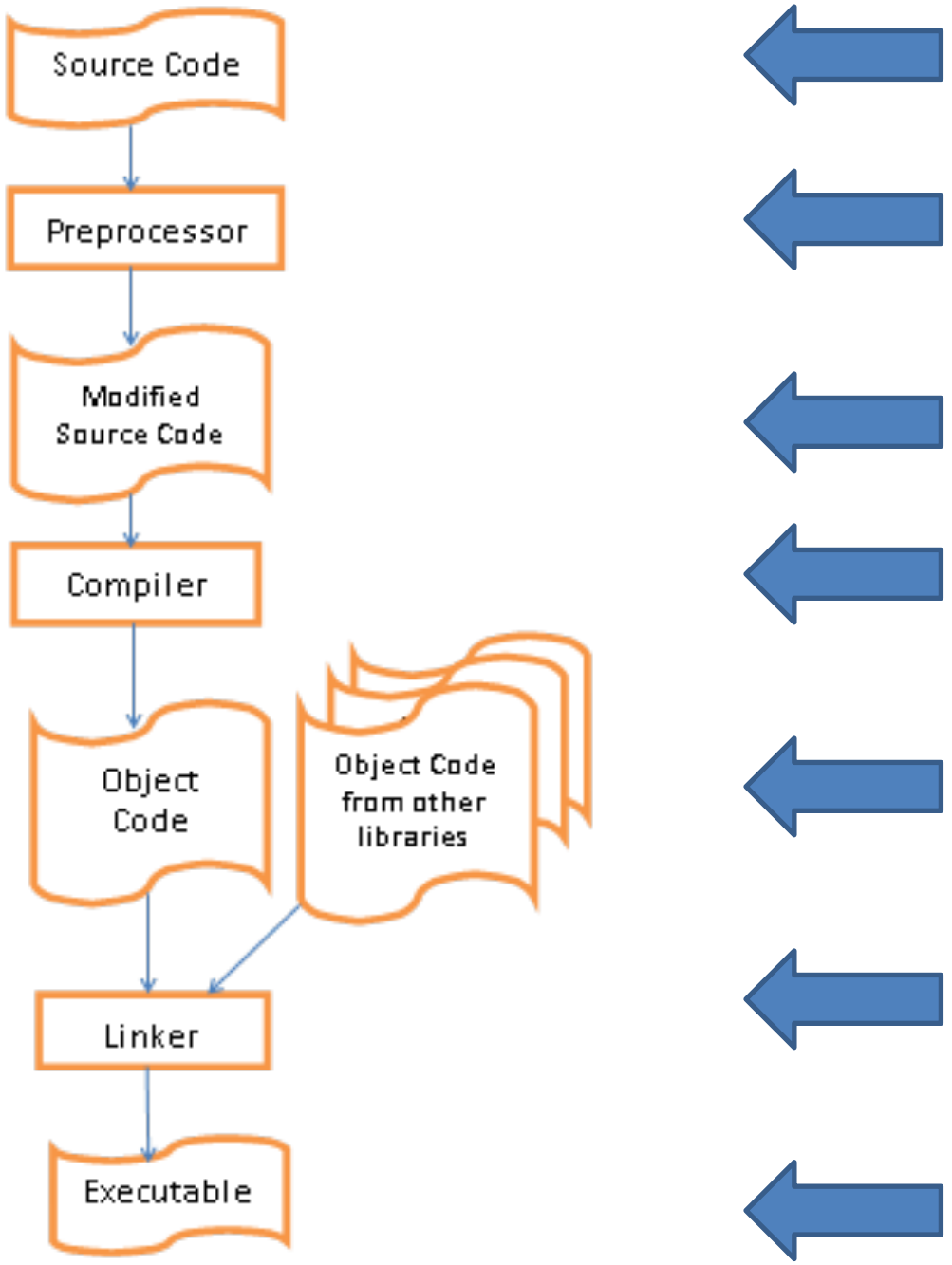
```
1   /*
2       Programmer    : Adi Zejnilovic
3       Date          : 07/28/2013
4       Description   : This program will display the following
    message:

5                       Hello world!
6   */
7   #include <iostream>
8
9   using namespace std;
10
11  int main()
12  {
13      cout << "Hello world!" << endl;
14      return 0;
15  }
```

Preprocessor Directive

```
1   // my first C++ program -Hello World
2   #include <iostream>
3
4   using namespace std;
5
6   int main()
7   {
8       cout << "Hello world!" << endl;
9       return 0;
10  }
```




Source Code
Preprocessor
Modified Source Code
Compiler
Object Code
Object Code from other libraries
Linker
Executable

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

- To execute a C++ program:
  - Use an editor to create a source program in C++
  - 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.)

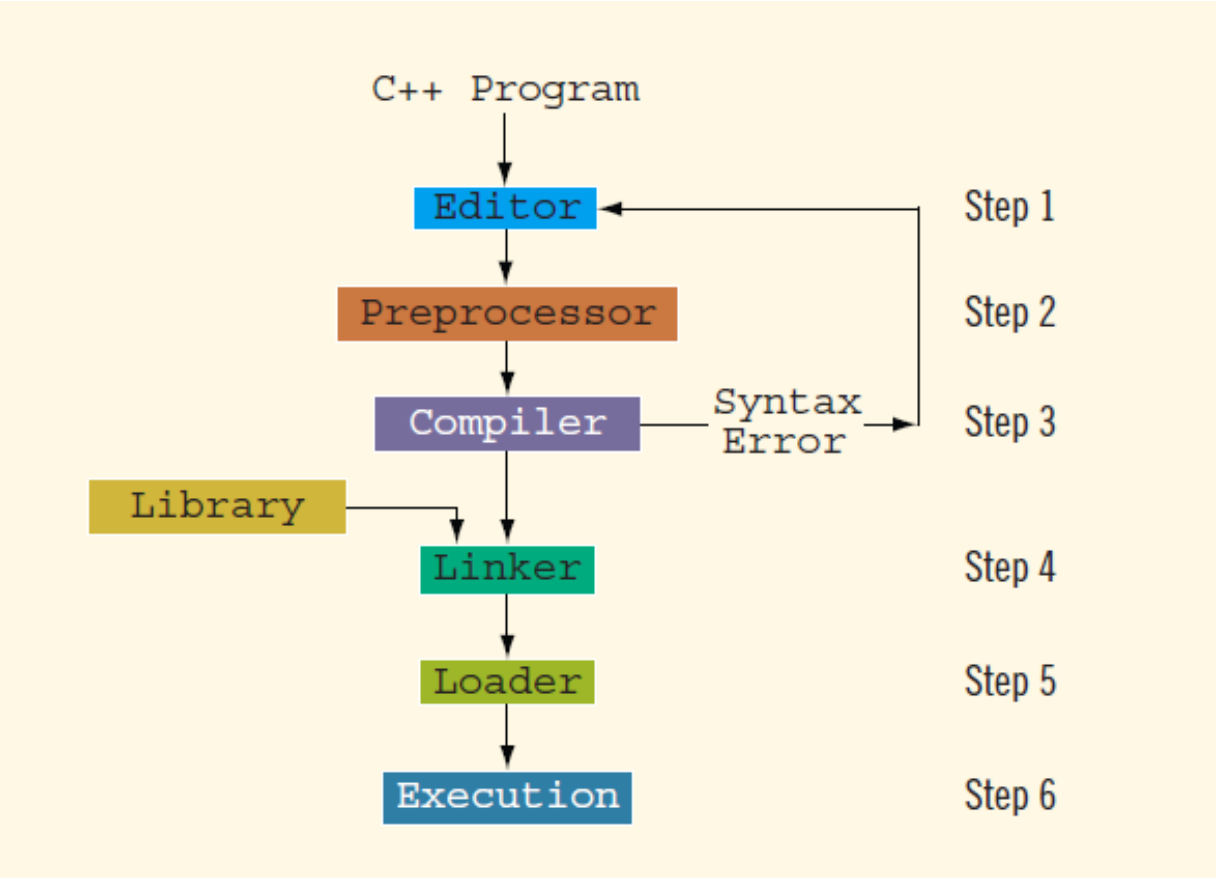


FIGURE 1-3 Processing a C++ program

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

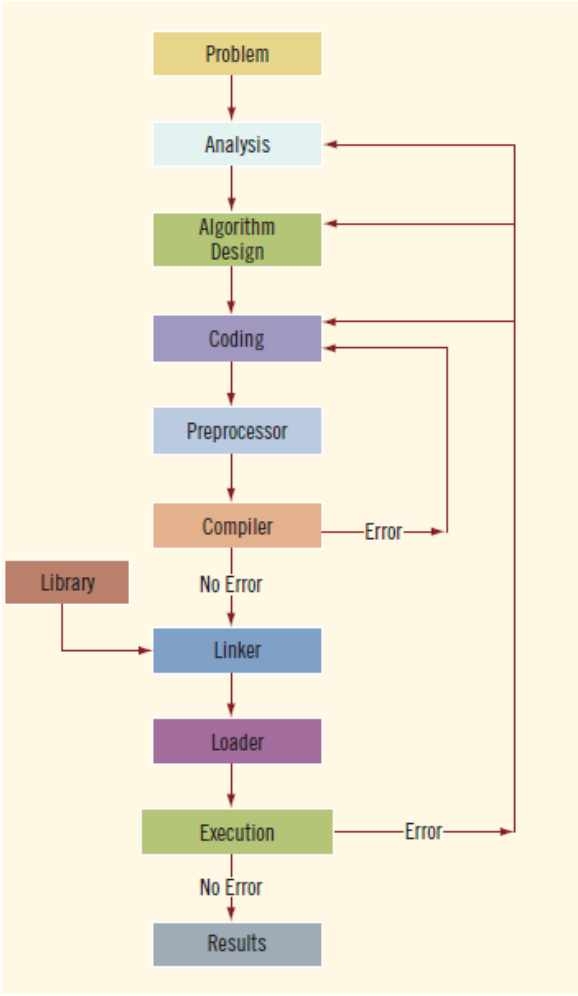


FIGURE 1-4 Problem analysis–coding–execution cycle

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: Maintenance
  - 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:

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

  - Find the area using the following equation:

```
area = length * 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

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