Chapter 1 Introduction to Computers and Programming

1.1 Why Program?

Concept:

Computers can do many different jobs because they are programmable.

- Computers are versatile tools used in various fields like education, business, and entertainment because they are **programmable**.
- A **program**, or **software**, is a set of instructions that a computer follows to perform a specific task.
- **Programmers** (or software developers) are skilled individuals who design, create, and test these programs.
- Programming is both an art and a science, requiring careful design in areas such as logical flow, user interface, and documentation. It also demands a scientific approach involving testing, correction, and adherence to the strict rules of programming languages like C++.

1.2 Computer Systems: Hardware and Software

Concept:

All computer systems consist of similar hardware devices and software components. This section provides an overview of standard computer hardware and software organization.

Hardware

- **Hardware** refers to the physical components that make up a computer system.
- A typical system includes the **central processing unit (CPU)**, **main memory**, **secondary storage devices**, **input devices**, and **output devices**.

The CPU

- The **central processing unit (CPU)** is the core component that runs or executes programs.
- Modern CPUs are powerful **microprocessors**, a significant advancement from the large electromechanical CPUs of early computers like the ENIAC.
- The CPU comprises two main parts: the **control unit**, which coordinates operations, and the **arithmetic and logic unit (ALU)**, which performs mathematical calculations.
- It operates on a **fetch/decode/execute cycle**: fetching an instruction from memory, decoding it, and executing the resulting operation.

Main Memory

- **Main memory**, or **Random-Access Memory (RAM)**, is the computer's temporary work area where programs and their data are stored while running.
- RAM is **volatile**, meaning its contents are erased when the computer is turned off.
- Memory is organized into **bytes**, with each byte consisting of eight **bits** (binary digits).
- Each byte has a unique **address** to identify its location in memory.

Secondary Storage

- **Secondary storage** provides long-term, non-volatile data storage, even without power.
- Common types include traditional **disk drives**, faster **solid-state drives (SSDs)**, portable **USB drives** and **SD cards**, and optical devices like **CDs** and **DVDs**.

Input Devices

- **Input** is any data the computer receives from the outside world.
- **Input devices** collect this data; examples include keyboards, mice, touchscreens, and scanners.

Output Devices

- **Output** is any information the computer sends to the outside world.
- **Output devices** present this information; examples include screens, printers, and speakers.

Software

- Software is essential for a computer's operation and is broadly divided into two categories: system software and application software.

System Software

- **System software** manages the fundamental operations of a computer.
- It includes **Operating Systems** (control hardware and run other programs), **Utility Programs** (perform specialized tasks like virus scanning), and **Software Development Tools** (used by programmers to create software).

Application Software

- **Application software** consists of programs that make the computer useful for everyday tasks, such as word processors, web browsers, and games.

Checkpoint

1.1 Why is the computer used by so many different people, in so many different professions?

1.2 List the five major hardware components of a computer system.

1.3 Internally, the CPU consists of what two units?

1.4 Describe the steps in the fetch/decode/execute cycle.

1.5 What is a memory address? What is its purpose?

1.6 Explain why computers have both main memory and secondary storage.

1.7 What are the two general categories of software?

1.8 What fundamental set of programs control the internal operations of the computer's hardware?

1.9 What do you call a program that performs a specialized task, such as a virus scanner, a file-compression program, or a data-backup program?

1.10 Word processing programs, spreadsheet programs, e-mail programs, web browsers, and game programs belong to what category of software?

1.3 Programs and Programming Languages

Concept:

A program is a set of instructions a computer follows in order to perform a task. A programming language is a special language used to write computer programs.

What Is a Program?

- A computer program is a set of instructions that directs a computer to solve a problem or perform a task.
- These instructions form an **algorithm**, which is a set of well-defined, sequential steps for a task.
- A CPU only understands **machine language**, a sequence of binary numbers (1s and 0s).
- **Programming languages**, like C++, were developed to make programming easier by using words instead of numbers. These are then converted to machine language.

Program 1-1

```
1   // This program calculates the user's pay.
2   #include <iostream>
3   using namespace std;
4
5   int main()
6   {
7       double hours, rate, pay;
8
9       // Get the number of hours worked.
10      cout << "How many hours did you work? ";
11      cin >> hours;
12
13      // Get the hourly pay rate.
14      cout << "How much do you get paid per hour? ";
15      cin >> rate;
16
17      // Calculate the pay.
18      pay = hours * rate;
19
20      // Display the pay.
```

```
21        cout << "You have earned $" << pay << endl;
22        return 0;
23    }
```

Program Output

```
How many hours did you work? 10 Enter
How much do you get paid per hour? 15 Enter
You have earned $150
```

Note:

The line numbers that are shown in Program 1-1 are *not* part of the program. This book shows line numbers in all program listings to help point out specific parts of the program.

Programming Languages

- Programming languages fall into two main categories: **low-level** and **high-level**.
- Low-level languages are close to machine language, while high-level languages are closer to human language and easier to learn.
- C++ is a popular language that combines both high-level and low-level features, offering significant **portability** (the ability to run on different computer systems with minimal changes).

Table 1-1 Programming Languages

LANGUAGE	DESCRIPTION
BASIC	Beginners All-purpose Symbolic Instruction Code. A general programming language originally designed to be simple enough for beginners to learn.
FORTRAN	Formula Translator. A language designed for programming complex mathematical algorithms.
COBOL	Common Business-Oriented Language. A language designed for business applications.

LANGUAGE	DESCRIPTION
Pascal	A structured, general-purpose language designed primarily for teaching programming.
C	A structured, general-purpose language developed at Bell Laboratories. C offers both high-level and low-level features.
C++	Based on the C language, C++ offers object-oriented features not found in C. Also invented at Bell Laboratories.
C#	Pronounced “C sharp.” A language invented by Microsoft for developing applications based on the Microsoft .NET platform.
Java	An object-oriented language that may be used to develop programs that run on many different types of devices.
JavaScript	JavaScript can be used to write small programs that run in webpages. Despite its name, JavaScript is not related to Java.
Python	Python is a general-purpose language created in the early 1990s. It has become popular in both business and academic applications.
Ruby	Ruby is a general-purpose language that was created in the 1990s. It is increasingly becoming a popular language for programs that run on web servers.
Visual Basic	A Microsoft programming language and software development environment that allows programmers to quickly create Windows-based applications.

> **Note:**
>
> Programs written for specific graphical environments often require significant changes when moved to a different type of system. Examples of such graphical environments are Windows, the X-Window System, and the macOS operating system.

Source Code, Object Code, and Executable Code

- A programmer writes **source code** in a text editor and saves it in a **source file**.
- The translation process involves several steps:
- **Preprocessor**: Reads the source code and processes lines beginning with `#`.
- **Compiler**: Translates the preprocessed code into machine language instructions called **object code**, stored in an **object file**. It also checks for **syntax errors**.
- **Linker**: Combines the object file with necessary routines from a **runtime library** to create a final **executable file** containing **executable code**.
- **Integrated Development Environments (IDEs)**, like Microsoft Visual Studio, bundle a text editor, compiler, and other tools into a single package to simplify this entire process.

```
g++ -o hello hello.cpp
```

Checkpoint

1.11 What is an algorithm?

1.12 Why were computer programming languages invented?

1.13 What is the difference between a high-level language and a low-level language?

1.14 What does *portability* mean?

1.15 Explain the operations carried out by the preprocessor, compiler, and linker.

1.16 Explain what is stored in a source file, an object file, and an executable file.

1.17 What is an integrated development environment?

1.4 What Is a Program Made of?

Concept:

There are certain elements that are common to all programming languages.

Language Elements

Table 1-2 Language Elements

LANGUAGE ELEMENT	DESCRIPTION
Key Words	Words that have a special meaning. Key words may only be used for their intended purpose. Key words are also known as reserved words.
Programmer-Defined Identifiers	Words or names defined by the programmer. They are symbolic names that refer to variables or programming routines.
Operators	Operators perform operations on one or more operands. An operand is usually a piece of data, like a number.
Punctuation	Punctuation characters that mark the beginning or end of a statement, or separate items in a list.
Syntax	Rules that must be followed when constructing a program. Syntax dictates how key words and operators may be used, and where punctuation symbols must appear.

Program 1-1

```
1   // This program calculates the user's pay.
2   #include <iostream>
3   using namespace std;
4
5   int main()
6   {
```

```
 7       double hours, rate, pay;
 8
 9       // Get the number of hours worked.
10       cout << "How many hours did you work? ";
11       cin >> hours;
12
13       // Get the hourly pay rate.
14       cout << "How much do you get paid per hour? ";
15       cin >> rate;
16
17       // Calculate the pay.
18       pay = hours * rate;
19
20       // Display the pay.
21       cout << "You have earned $" << pay << endl;
22       return 0;
23   }
```

Key Words (Reserved Words)

- **Key words** (or reserved words) have a special meaning in the programming language and can only be used for their intended purpose.
- Examples in the program include `using`, `namespace`, `int`, and `double`. In C++, key words are always lowercase.

Note:

The `#include <iostream>` statement in line 2 is a preprocessor directive.

Note:

In C++, key words are written in all lowercase.

Programmer-Defined Identifiers

- These are names created by the programmer for things like variables.
- In the example program, `hours`, `rate`, and `pay` are programmer-defined identifiers for variables.

Operators

- **Operators** are symbols that perform operations on data (operands).
- The program uses the multiplication (`*`) and assignment (`=`) operators.

Punctuation

- Punctuation symbols serve specific purposes, like marking the end of a statement.
- In C++, a semicolon (`;`) is used to end a complete statement, similar to a period in English.

Lines and Statements

- A **line** is a single line of text in the program's source code.
- A **statement** is a complete instruction that causes the computer to perform an action. It may span multiple lines.

Variables

- A **variable** is a named storage location in the computer's memory (RAM) for holding data that can change while the program runs.
- The program uses variables named `hours`, `rate`, and `pay` to store user input and calculation results.

Variable Definitions

- Data types are generally categorized as numbers (integers, floating-point) and characters.
- A **variable definition** is a statement that specifies a variable's name and the type of data it will hold.
- All variables must be defined before they can be used. The definition `double hours, rate, pay;` creates three variables to hold floating-point numbers.

> **Note:**
>
> Programmers often use the term "variable declaration" to mean the same thing as "variable definition." Strictly speaking, there is a difference between the two terms. A definition statement always causes a variable to be created in memory. Some types of declaration

statements, however, do not cause a variable to be created in memory. You will learn more about declarations later in this book.

1.5 Input, Processing, and Output

Concept:

The three primary activities of a program are input, processing, and output.

- Programs typically follow a three-step sequence: input, processing, and output.
- **Input**: Information a program gathers from the outside world. The program uses `cin` to get the hours worked and pay rate from the user.
- **Processing**: The program processes the input data. In the example, it multiplies the hours by the rate to calculate the pay.
- **Output**: Information the program sends to the outside world. The program uses `cout` to display messages and the final result on the screen.

Checkpoint

1.18 Describe the difference between a key word and a programmer-defined identifier.

1.19 Describe the difference between operators and punctuation symbols.

1.20 Describe the difference between a program line and a statement.

1.21 Why are variables called “variable”?

1.22 What happens to a variable’s current contents when a new value is stored there?

1.23 What must take place in a program before a variable is used?

1.24 What are the three primary activities of a program?

1.6 The Programming Process

Concept:

The programming process consists of several steps, which include design, creation, testing, and debugging activities.

Designing and Creating a Program

- **Clearly define what the program is to do.** This involves identifying its purpose, required input, the processing to be performed, and the desired output.
- **Visualize the program running on the computer.** Imagine the user's experience, including on-screen messages and prompts.
- **Use design tools to create a model.** Common tools include:
    - **Hierarchy charts**, which graphically depict the program's structure using a top-down design approach.
    - **Flowcharts**, which show the logical flow of the program.
    - **Pseudocode**, a mix of human language and programming language used to outline the program's algorithm.
- **Check the model for logical errors.** Review the design to catch mistakes before coding begins.
- **Type the code, save it, and compile it.** Write the source code and use the compiler to translate it, which will identify any syntax errors.
- **Correct any errors found during compilation.** Repeat the compile-and-correct cycle until the program is free of syntax errors.
- **Run the program with test data for input.** Test for runtime errors (logical errors that occur while the program is running) by using sample data with predictable outcomes.
- **Correct any runtime errors found.** Use methods like desk-checking (manually stepping through the code) to find and fix logical mistakes, then recompile and retest.
- **Validate the results.** Ensure the corrected program solves the original problem as intended.

What Is Software Engineering?

- **Software engineering** is the comprehensive process of creating software, including design, writing, testing, debugging, and maintenance.

- It utilizes various tools like program specifications, diagrams, and pseudocode.
- Large software projects are often developed by teams, where the work is divided into modules based on a top-down design.

1.7 Procedural and Object-Oriented Programming

Concept:

Procedural programming and object-oriented programming are two ways of thinking about software development and program design.

- C++ supports two primary programming paradigms: procedural and object-oriented.
- **Procedural programming** focuses on creating procedures (or functions), which are collections of statements that perform a specific task.
- **Object-Oriented Programming (OOP)** is centered on the **object**, a self-contained unit that bundles data and the procedures (or methods) that operate on that data.

Checkpoint

1.25 What four items should you identify when defining what a program is to do?

1.26 What does it mean to "visualize a program running"? What is the value of such an activity?

1.27 What is a hierarchy chart?

1.28 Describe the process of desk-checking.

1.29 Describe what a compiler does with a program's source code.

1.30 What is a runtime error?

1.31 Is a syntax error (such as misspelling a key word) found by the compiler or when the program is running?

1.32 What is the purpose of testing a program with sample data or input?

1.33 Briefly describe the difference between procedural and object-oriented programming.

Review Questions and Exercises

Short Answer

1. Both main memory and secondary storage are types of memory. Describe the difference between the two.
2. What is the difference between system software and application software?
3. What type of software controls the internal operations of the computer's hardware?
4. Why must programs written in a high-level language be translated into machine language before they can be run?
5. Why is it easier to write a program in a high-level language than in machine language?
6. Explain the difference between an object file and an executable file.
7. What is the difference between a syntax error and a logical error?

Fill-in-the-Blank

1. Computers can do many different jobs because they can be _____.
2. The job of the _____ is to fetch instructions, carry out the operations commanded by the instructions, and produce some outcome or resultant information.
3. Internally, the CPU consists of the _____ and the _____.
4. A(n) _____ is an example of a secondary storage device.
5. The two general categories of software are _____ and _____.
6. A program is a set of _____.
7. Since computers can't be programmed in natural human language, algorithms must be written in a(n) _____ language.
8. _____ is the only language computers really process.
9. _____ languages are close to the level of humans in terms of readability.
10. _____ languages are close to the level of the computer.

11. A program's ability to run on several different types of computer systems is called _____.

12. Words that have special meaning in a programming language are called _____.

13. Words or names defined by the programmer are called _____.

14. _____ are characters or symbols that perform operations on one or more operands.

15. _____ characters or symbols mark the beginning or end of programming statements, or separate items in a list.

16. The rules that must be followed when constructing a program are called _____.

17. A(n) _____ is a named storage location.

18. A variable must be _____ before it can be used in a program.

19. The three primary activities of a program are _____, _____, and _____.

20. _____ is information a program gathers from the outside world.

21. _____ is information a program sends to the outside world.

22. A(n) _____ is a diagram that graphically illustrates the structure of a program.

Algorithm Workbench

Draw hierarchy charts or flowcharts that depict the programs described below. (See Appendix C for instructions on creating flowcharts.)

1. Available Credit

   The following steps should be followed in a program that calculates a customer's available credit:

   1. Display the message "Enter the customer's maximum credit."
   2. Wait for the user to enter the customer's maximum credit.
   3. Display the message "Enter the amount of credit used by the customer."
   4. Wait for the user to enter the customer's credit used.

5. Subtract the used credit from the maximum credit to get the customer's available credit.

6. Display a message that shows the customer's available credit.

2. Sales Tax

   Design a hierarchy chart or flowchart for a program that calculates the total of a retail sale. The program should ask the user for:

   - The retail price of the item being purchased
   - The sales tax rate

   Once these items have been entered, the program should calculate and display:

   - The sales tax for the purchase
   - The total of the sale

3. Account Balance

   Design a hierarchy chart or flowchart for a program that calculates the current balance in a savings account. The program must ask the user for:

   Designing the Account Balance Program

   - The starting balance
   - The total dollar amount of deposits made
   - The total dollar amount of withdrawals made
   - The monthly interest rate

   Once the program calculates the current balance, it should be displayed on the screen.

Predict the Result

Questions 33–35 are programs expressed as English statements. What would each display on the screen if they were actual programs?

Predicting the Result of Problem 33

1. The variable `x` starts with the value 0.

   The variable `y` starts with the value 5.

   Add 1 to `x`.

   Add 1 to `y`.

   Add `x` and `y`, and store the result in `y`.

   Display the value in `y` on the screen.

2. The variable `j` starts with the value 10.

   The variable `k` starts with the value 2.

   The variable `l` starts with the value 4.

   Store the value of `j` times `k` in `j`.

   Store the value of `k` times `l` in `l`.

   Add `j` and `l`, and store the result in `k`.

   Display the value in `k` on the screen.

3. The variable `a` starts with the value 1.

   The variable `b` starts with the value 10.

   The variable `c` starts with the value 100.

   The variable `x` starts with the value 0.

   Store the value of `c` times 3 in `x`.

   Add the value of `b` times 6 to the value already in `x`.

   Add the value of `a` times 5 to the value already in `x`.

Display the value in `x` on the screen.

Find the Error

1. The following *pseudocode algorithm* has an error. The program is supposed to ask the user for the length and width of a rectangular room, then display the room's area. The program must multiply the width by the length in order to determine the area. Find the error.
    - *area = width × length.*
    - *Display "What is the room's width?".*
    - *Input width.*
    - *Display "What is the room's length?".*
    - *Input length.*
    - *Display area.*