TOPIC 1: INTRODUCTION

:: Learning Objectives ::

At the end of the lesson, the students should be able to:

- ❒ explain the basic history of computer and programming languages
- ❒ describe what the computer is
- ❒ describe what a computer program is
- ❒ appreciate the importance of good programs
- ❒ explain the relationship between compilers, interpreters and programs
- ❒ recognize the program errors
- ❒ become familiar with the program design process

:: Subtopics ::

1. A brief history of computer and programming languages
2. Introduction to programming
    a. What is a computer program
    b. Importance of good programs
    c. Relationship between compilers, interpreters, assemblers and programs
3. Program development life cycle
    a. Problem analysis, design, implementation (coding), testing and maintenance

1.1 A Brief of Computer History

Table 1.1 The early history of computers

Year	Machine	Description
	Abacus	The abacus was invented in Asia but was used in ancient Babylon, China, and throughout Europe until the late middle ages.
1642	Pascaline	A calculating device invented by French philosopher and mathematician Blaise Pascal.
1819	Jacquard Loom	A weaver that could rearrange the cards (these cards contain weaving instructions) and change the pattern being woven.
1800s	Difference Engine	The difference engine could perform complex operations such as squaring numbers automatically. Charles Babbage built a prototype of the difference engine, but the actual device was never produced.
	Analytical Engine	The analytical engine's design included input device, data storage, a control unit that allowed processing instructions in any sequence, and output devices. However, the designs remained in blueprint stage.

Ada Augusta, Countess of Lovelace is considered the first computer programmer.

The computers that we know today use the design rules given by John von Neumann in the late 1940s. His design included components such as an arithmetic logic unit, a control unit, memory, and input/output devices. Von Neumann's computer design makes it possible to store the programming instructions and the data in the same memory space. In 1951, the UNIVAC (Universal Automatic Computer) was built.

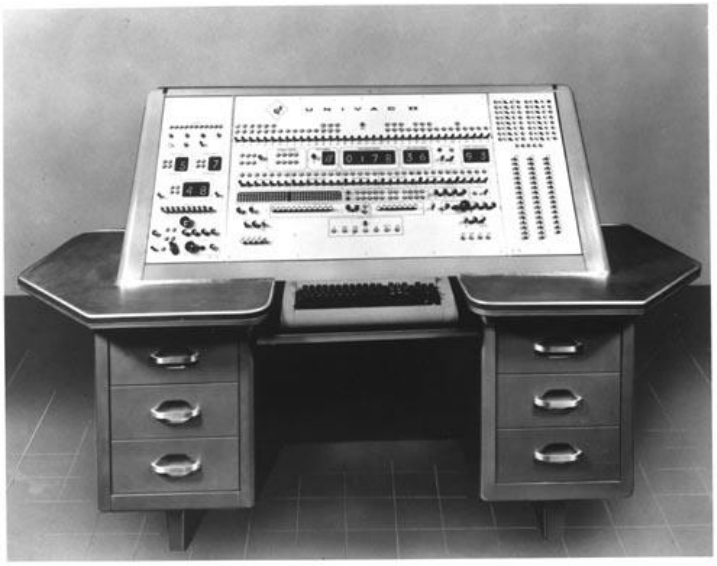

Figure 1.1 UNIVAC II Console

1.2 A Brief Overview of Programming Languages

❖ Machine Language

Machine language is the most basic language of a computer. Program instructions in machine language are written in bits.

```
100100 010001
100110 010010
100010 010011
```

Figure 1.2 Example of instructions in machine language

To represent the mathematical equation in machine language, the programmer had to remember the machine language codes for various operations. Also, to manipulate data, the programmer had to remember the locations of the data in the main memory. This need to remember specific codes made programming not only very difficult, but also error-prone.

❖ Assembly Language

Assembly languages were developed to make the programmer's job easier. In assembly language, an instruction is an easy-to-remember form called a mnemonic.

Table 1.2 Examples of instructions in assembly language and its equivalent machine language

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

It is much easier to write instructions in assembly language. However, a computer cannot execute assembly language instructions directly. The instructions first have to be translated into machine language. A program called an assembler translates the assembly language instructions into machine language.

❖ High-level languages

The next step toward making programming easier was to devise high-level languages that were closer to natural languages (examples are English, Spanish and French). Basic, FORTRAN, COBOL, Pascal, C, C++, C#, and Java are all high-level programming languages.

The instruction written in any high-level programming language is much easier to understand and is self-explanatory even to a novice user who is familiar with basic arithmetic. As in the case of assembly language, however, the computer cannot directly execute instructions written in a high-level programming language. To run on a computer, these instructions first need to be translated into machine language. A program called a compiler translates instructions written in high-level programming languages into machine code.

Assembler:
A program that translates a program written in assembly language into an equivalent program in machine language.

Compiler:
A program that translates instructions written in a high-level language into the equivalent machine language.

1.3 Elements of A Computer System

A computer is an electronic device capable of performing commands. The basic commands that a computer performs are input (get data), output (display result), storage, and performance of arithmetic and logical operations.

There are TWO (2) basic components of a computer system are:

(i) Hardware

Hardware consists of all the physical devices. Major hardware components include the:

- central processing unit (CPU)
- main memory (MM), also called random access memory (RAM)
- input/output devices - Some examples of input devices are the keyboard, mouse, and secondary storage. Examples of output devices are the screen and printer.

- secondary storage.

The central processing unit is the "brain" of the computer and the single most expensive piece of hardware in a computer. The more powerful the CPU, the faster the computer becomes. Arithmetic and logical operations are carried out inside the CPU.

Main memory, or random access memory, is connected directly to the CPU. All programs must be loaded into main memory before they can be executed. Similarly, all data must be brought into main memory before a program can manipulate it. When the computer is turned off, everything in main memory is lost.

Main memory is an ordered sequence of cells, called memory cells. Each cell has a unique location in main memory, called the address of the cell. These addresses help you access the information stored in the cell.

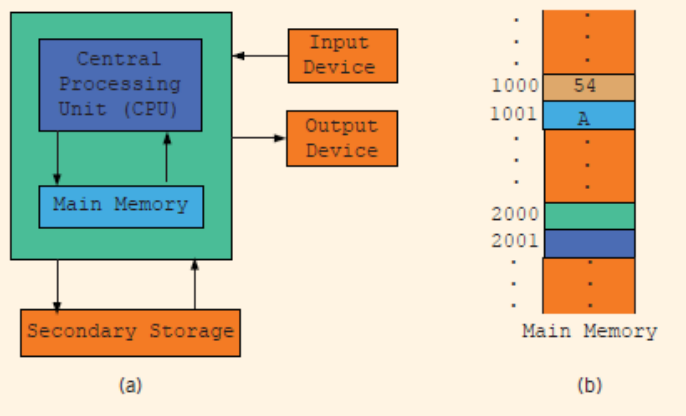

Figure 1.3 Hardware components of a computer and main memory

Because programs and data must be stored in main memory before processing and because everything in main memory is lost when the computer is turned off, information stored in main memory must be transferred to some other device for permanent storage. The device that stores information permanently (unless the device becomes unusable or you change the information by rewriting it) is called secondary storage. To be able to transfer information from main memory to secondary storage, these components must be directly connected to each other. Examples of secondary storage are hard disks, flash drives, floppy disks, ZIP disks, CD-ROMs, and tapes.

For a computer to perform a useful task, it must be able to take in data and programs and display the results of calculations. The devices that feed data and programs into computers are called input devices. The keyboard, mouse, and secondary storage are examples of input devices. The devices that the computer uses to display results are called output devices. A monitor, printer, and secondary storage are examples of output devices.

(ii) Software

Software are programs written to perform specific tasks. For example, word processors are programs that you use to write letters, papers, and even books. All software is written in programming languages. There are two types of programs: system programs and application programs.

System programs control the computer. The system program that loads first when you turn on your PC is called the operating system. Without an operating system, the computer is useless. The operating system monitors the overall activity of the computer and provides

services. Some of these services include memory management, input/output activities, and storage management. The operating system has a special program that organizes secondary storage so that you can conveniently access information.

Application programs perform a specific task. Word processors, spreadsheets, and games are examples of application programs. The operating system is the program that runs application programs.

1.4 Computer Programs

A program is a set of instructions that directs the computer to accomplish specific tasks. Another term commonly used for computer program is software.

Every program is written in some programming languages that contain a set of instructions, data and rules that are used to construct a program.

What are the examples of programming languages that are already mentioned previously?

The process of writing or coding programs is called programming and the individuals who write programs are called programmers. Users, on the other hand, are individuals who use the program.

In order for a program to be considered as a good program, it must fulfill the following criteria:

- Reliability of output

  A good program must be able to produce correct output. For that, a different set of input data is used to ensure the reliability of the output.

- Program's efficiency

  A good program must be designed to be efficient and reliable in the sense that it produces no errors during execution process. Also, the program must achieve its purpose so that the final result can be trusted. Thus, it is important that the program is outlined first using the pseudo code or flowchart tool.

- Interactivity

  The interaction process between the user and the program must be well defined. The interactivity is important so that the user knows the processing status. Programs that are user-friendly allow the users to respond to instructions correctly and this will help the users to key in valid input thus minimizing the errors resulted from invalid data.

  For example, users should be given clear instruction on how to complete the data entry task. Without the instruction, users might give invalid data input.

- Program readability

  Readability is concerned with how other person views one's program. For programmers, the use of indentation and comments are common to improve the program readability.

Indentation

Indentation helps in making the structure of the program clearer and easier to read. A statement within a statement should be indented to show the user which statements are subordinate of the other.

Comments

Some explanatory notes or comments (sometimes referred to as internal documentation) should be placed in the program coding to improve its readability. In other words, comments are there for the convenience of anyone reading the program.

Comments can be placed anywhere within a program and they will not be executed by the compiler. Commenting out a section of code is useful in a debugging process.

C++ supports two types of comments:

- Line comment

  A line comment begins with two slashes (//) and continues to the end of line.

- Block comment

  Block comments begin with the symbol /* and end with the symbols */. Block comments are conveniently used for statements than span across two or more lines.

```
if (score > 90)
{ highest = highest + 1;
cout<<"Highest score";}
else if (score < 40)
{ lowest = lowest + 1;
cout<<"Lowest score";}
```

(a) Without indentations

```
if (score > 90)
{
      highest = highest + 1;
      cout<<"Highest score";
}
else if (score < 40)
{
      lowest = lowest + 1;
      cout<<"Lowest score";
}
```

(b) With indentations

Coding 1.1 Indentations

```
#include<iostream.h> //preprocessor directive

int main() //this is the main function
{
      cout<<"Hello user.";
      cout<<"\nWelcome to C++";
      return 0; //return statement
}
```

Coding 1.2 Example of single line comment

```
/*
This is my first program in C++.
This program will display a welcome message to the user. */

#include<iostream.h>

int main()
{
      cout<<"Hello user.";
      cout<<"\nWelcome to C++";
      return 0;
}
```

Coding 1.3 Example of block comment

1.5 Program errors

Although the principle in programming is to efficiently produce readable and error free programs that works correctly, errors or sometimes are called bugs can occur at any time.

There are THREE (3) different types of errors:

- Run time errors

  Run time errors occur during the execution of a program and are detected by the compiler. Example of run-time error is division by zero.

> The method for detecting errors after a program has been executed is called debugging process. A debugger program is available in C++ for detecting errors while a program is being executed.

- Syntax errors

  A syntax error is an error in the structure or spelling of a statement. This error can be detected by the compiler during compilation of the program.

```
cout<<"/nThis is a statement in C++" //Line 1
cot>>This is the second line."; //Line 2
```

Coding 1.4 Example of syntax errors

There are FIVE (5) syntax errors that can be detected in Coding 1.4:

1 - Invalid use of backslash (/) in Line 1
2 - Missing semicolon (;) in Line 1
3 - Keyword `cout` is misspelled in Line 2
4 - Invalid use of insertion symbol (>>) in Line 2
5 - Missing aN opening quote (") in Line 2

A C++ statement with correct syntax will be accepted by the compiler without any error messages.

Nevertheless, a statement or a program can be syntactically correct but still be logically incorrect. Consequently, incorrect result would be produced.

❖ Logic errors

Logic errors refer to the unexpected or unintentional errors resulted from some flaws in the program's logic. Logic error is very difficult to detect since compilers cannot detect the errors. The only way to detect the error is by testing the program thoroughly and comparing its output against manually calculated results.

Many errors can be eliminated simply by checking a copy of the program before it is compiled or executed. Writing down each variable as it is encountered in the program and listing the value that should be stored in the variable is called program tracing.

Program tracing sharpens one's programming skill because it requires that the person who writes the program really understands what each statement in the program does.

1.6 Compiling a C++ Program

Programs written in other languages than machine language need to go through a process called compiling.

What is compiling?

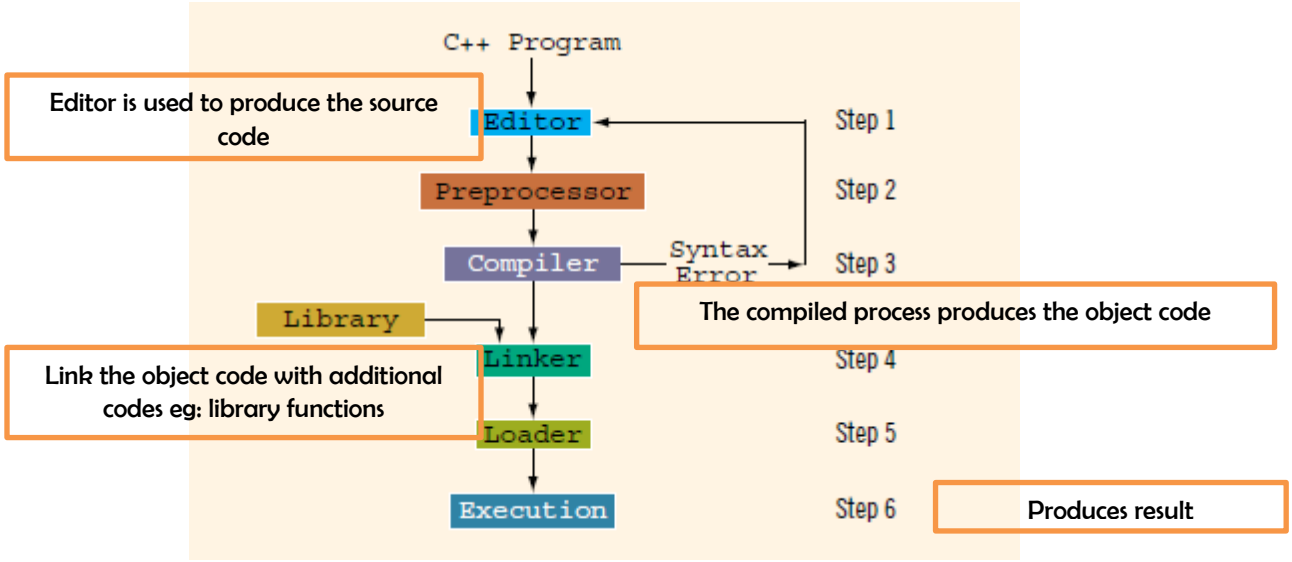

Figure 1.4 How a typical C++ is processed

Figure 1.4 illustrates the translation process for a C++ program to be successfully executed.

During compilation of program, a compiler will convert the entire program into machine language. Interpreter is also used to convert a program into machine language. But, it will translate one program statement at a time, instead of the entire program.

- Source code

  Source code is the original program written in assembly or high-level languages. The file containing the source code is called the source file, and in C++, it uses the filename extension .cpp (cpp stands for C plus plus).

- Object code

  Object code is created from the compilation process in machine-readable form. The file containing the object code is called the object file, and it uses the file name extension .obj (obj stands for object).

  After the compiler creates the object file, it then invokes another program called a linker. The linker combines the object file with other machine code necessary for the C++ program to run correctly. The linker the produces an executable file that has an extension of .exe (exe stands for executable).

1.7 Program Design Process

In designing a program, there is no complete set of rules or specific algorithm to follow. However, software developers try to use a reasonably consistent problem solving approach for constructing computer programs.

Programming is a process of problem solving.

The phases in the problem solving approach are outlined as follows:

Phase 1 – Problem definition (analysis)
Phase 2 – Algorithm design
Phase 3 – Algorithm implementation
Phase 4 – Program testing
Phase 5 – Program maintenance

The phases can be further divided into two:

1 - Problem solving phase – Phase 1 and 2
2 - Implementation phase – Phase 3, 4 and 5

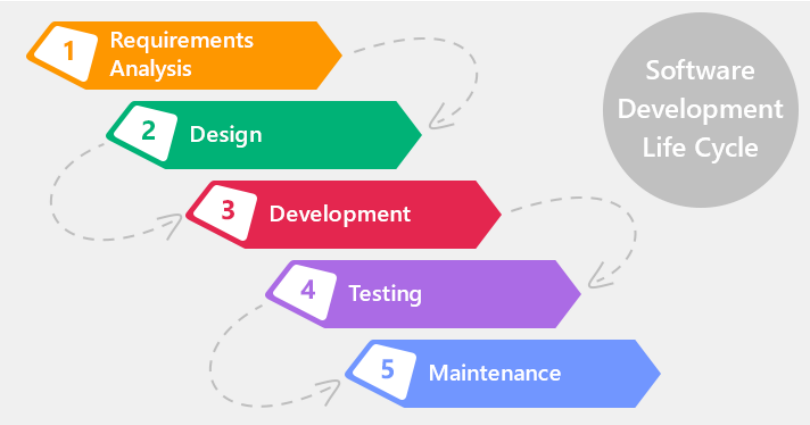

Figure 1.5 5 phases of software development life cycle

❖ Problem definition phase

Problem definition phase is also known as analysis phase. The problem is defined to obtain a clear understanding of the problem requirements.

The following questions should be asked to get a complete problem specification:

I. What are the input data?
II. What are the output (desired) data?
III. What formula is to be used?
IV. What other assumptions or constraints can be made?
V. What is the expected output screen?

❖ Algorithm design phase

The specifications derived earlier in the analysis phase are translated into the algorithm.

An algorithm is step-by-step sequence of precise instructions that must terminate and describes how the data is to be processed to produce the desired outputs. The instructions may be expressed in a human language.

An algorithm must satisfy some requirements:

a) Input and output – It must have input and must produce at least one output.
b) Unambiguous – Every step in an algorithm must be clear as to what it is supposed to do and how many times it is expected to be executed.
c) Correct and efficient – It must be correct and efficiently solved the problem for which it is designed.
d) Finite – It must execute its instructions and terminate in finite time.

An algorithm can be written or described using several tools. Listed below in Table 1.3 are two tools to write an algorithm.

Table 1.3 Tools to describe an algorithm

Pseudo code	Flowchart
Use English-like phrases to describe the processing process. It is not standardized since every programmer has his or her own way of planning the algorithm.	Use standardized symbols to show the steps the computer needs to take to accomplish the program's objective.

```
1. START
2. READ R
3. SET PI=3.142
4. CALCULATE AREA OF CIRCLE
   4.1.    FORMULA:
           Area= PI * R * R
5. DISPLAY Area
6. END
```

Figure 1.6 Example of pseudo code

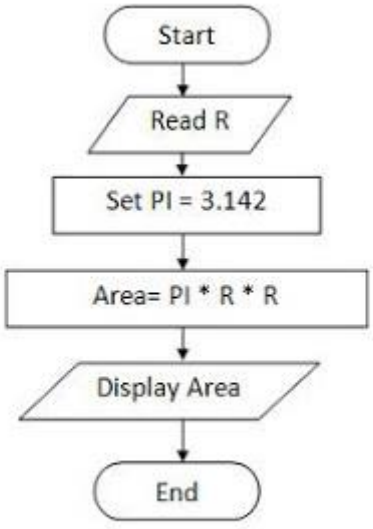

Figure 1.7 Example of a flowchart

❖ Algorithm implementation

The algorithm is translated into a computer program by using a specific programming language. The process is called coding, which involves editing, compiling and debugging.

- **Program testing**

  Program testing requires testing the completed program to verify that it produces expected output. A different set of testing data is normally used to verify that the program works properly and that it is indeed solving the given problem.

- **Program maintenance**

  Often, there may be new requirements to be added into the current program. Making revisions to meet the changing needs with ongoing correction of problems are the major efforts in the program maintenance. As a result, the program codes may be modified, added or deleted accordingly.

  Thus, it is very important that a program is well documented for future development.

Backup:

Although backup is not part of the program design process, it is critical to make and keep the backup copies of the program at each step of the programming process.

Backup copies allow the recovery of the last stage of work with a minimum effort.