1. Introduction to Computers and Programming

This chapter provides the student a brief introduction to the basics of hardware components of a computer, and the functions of these components. Further on, the chapter provides an understanding of language of the computer, the ideas of computer program, programming language, and compiler.

1.1. Basic Computer Concepts

Computers and Computer Programming

A **computer** is a machine that **is capable of accepting, storing and acting on data and/or a series of instructions given to it by the user**, at a great speed with a high degree of accuracy.

A computer is a machine which can accept, store and manipulate data accurately and extremely quickly. Most computer-users have the wrong perception that the computers are intelligent machines, and they create wonders. Any apparent intelligence of a computer is in fact the intelligence of the programmer who programmed it. **The computer can only carry out instructions given by the programmer**. It doesn't have any intelligence, and it cannot think.

If we want to use the computer to solve a problem, then first **we have to come up with a method to solve the problem**. That is not all; **we have to break up this method of solution into simple steps, and then we have to write step by step instructions that the computer can carry out and arrive at the solution**.

The **set of instructions written for the computer** to guide it through the process of achieving a specific task is called a ***computer program*** (or software).

Professional programs are often referred to as software. Computer system itself is called the hardware. Writing programs is called computer programming. **Breaking up a method of solution into simple step by step instructions is the most difficult part of computer programming**. What makes it so difficult? Imagine giving instructions to a brainless individual, and trying to get things done through him. We, the programmers, have to do all the thinking so that the computer doesn't have to think. Sometimes it is easier to solve the problem by ourselves than writing a computer program to solve it. Then why do we bother to write programs? Why not solve the problems ourselves? We have to know how to do them either way.

There are many **reasons to prefer a computer over humans**. The following are some.

1. A computer carries out instructions **perfectly accurately**, whereas humans make errors.
2. Computer carries out instructions at an **extremely high speed**. Computations that would take months for a human to complete can be finished in seconds.
3. Computer **never gets tired** or bored.
4. Once a program is written, it is good forever **for everyone in the world to solve similar problems**.

Computer Systems

To be able to use a computer, in addition to the computer we need various devices that enable us to communicate with the computer. The term "computer system" is used to refer to the complete system that consists of all the necessary modules, or devices connected to the computer.

A computer system can be thought of as a combination of **three main hardware components** :

1. **Input device** : a device that allows a person to communicate data to the computer. Examples: keyboard, hard disk, cd, mouse, scanner.
2. **Output device** : a device that allows the computer to communicate information to the user. Examples: monitor, printer.

3. **Central Processing Unit** (CPU): This is the computer. It controls the operation of the entire system by carrying out instructions given in the operating system program. It also carries out the instructions in the program when we execute a program.

Let us now understand what these components are and what their functions are.

The Central Processing Unit

The central processing unit is designed to carry out computing tasks in the way similar to the way human brain does. When we process information, we store them in the memory part of our brain first. Then the processor part of our brain processes the information in the memory. To work the same way, the CPU part of the computer consists of a **processor** and a **memory**. The processor in turn consists of **control unit** and **arithmetic-logic unit** (ALU). The control unit fetches instructions from memory, decodes them, and directs the system to execute the operations indicated by the instructions. The ALU carries out the operations that are arithmetical or logical in nature by using special registers and circuits. The **memory unit is also identified by the names internal memory, primary memory, main memory and random access memory (RAM).** It is used to store the instructions (programs) and data of the programs being executed. In other words, the memory is for the computer to temporarily store all its instructions that are to be carried out at the time, and the data that are to be used at the time.

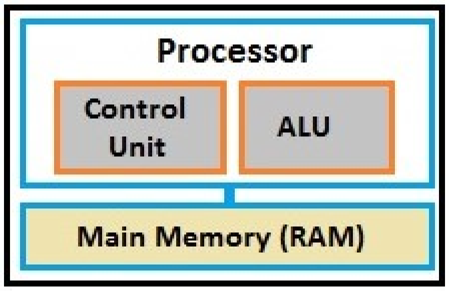

Figure 1.1. The central processing unit

Computer users often mix-up the term memory to permanant storage-media. **Examples of permanent storage media are hard-disk, usb media, cd and dvd. A permanent storage-media is external to CPU, and it is for permanently storing data and programs**. **Internal memory, on the other hand, is only for the computer to store the data and programs temporarily while processing them**. Internal memory is also known as Random Access Memory (RAM), and it is simply an ordered sequence of memory cells (which are integrated circuits) contained on small silicon blocks called chips. Internal memory needs power to hold the data, and therefore, is erased when the computer is turned off. Student will learn more details of how data are stored in the memory in Section 1.2.

The CPU is the computer. All other components of a computer system are used to either extend the capabilities of the computer system or to provide the user a means to communicate with the computer. These units are also called ***peripherals***. Peripheral devices are input devices, output devices, and secondary storage devices such as external hard disks..

Input / Output and Data Processing

Input is what we provide to the computer and output is what we get out of the computer after processing the input. Usually **the input would consist of the data (characters or numbers most of the time) and a program**. When the user executes a program, the program is first copied into the memory. After that if the user supplies the data, they also are stored in the memory. Processor then caries out the instructions found in the program which guide it through processing of the data that are stored in the memory, in the manner intended by the programmer. The output is the results produced by the computer after processing the data by following the instructions in the program.

Input to a computer usually consists of two parts:
1. program.
2. data.

The process of providing some input (data and program) to the computer and processing and receiving output is called data processing.

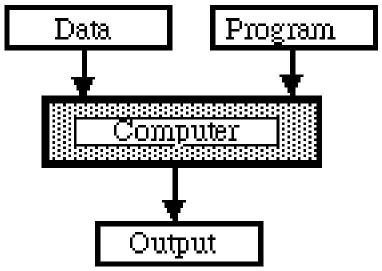

Figure 1.2. Data processing

Operating System

As we have noted earlier all that a computer can do is carry out instructions. It cannot think and therefore cannot do anything by itself. All the basic functions of the computer, such as recognizing a disk in the disk drive, and recognizing and acting on various commands issued by the user including mouse-movements, are controlled by a set of instructions given in a program called Operating System.

Operating system is a program (software) that **controls the overall operation of the computer**

It is due to the operating system program that the computer displays a prompt, and responds to various commands we type using keyboard. Without the operating system program the computer would not understand what your commands mean, and therefore would not function at all. The operating system program must be provided to the computer before giving any other program or data. In microcomputers the operating system program is usually stored in disks. The disk that contains the operating system program is usually called the *system disk* or *startup disk*.. If there is a hard disk in the computer, the operating system is usually stored on the hard disk. If there is no hard disk, after turning a microcomputer on, the startup disk (usually a cd or dvd) must be inserted first. **The computer will then copy the operating system program from the disk or hard disk into the memory**. This process is called booting up.

It is the operating system that determines the style in which "user-computer interaction" (known as interface) is done in a specific computer. All the computers have the same basic structure, the same components, and use the same principle. However, different operating systems make them interact with the user differently. For example, Apple Macintosh and PC with Windows appear to work differently because of the two different operating systems: Mac OS, and Windows.

Exercises 1.1

1. What is a computer?

2. What is a computer program?

3. What are the three main components of a computer system?

4. What are the two components of the central processing unit?

5. What are the components of the processor?

6. What are the two parts of input usually given to a computer?

7. What is an operating system?

8. What software must be installed in the computer before we can start using it?

9. Describe the key operation the computer performs before it is ready for the user after the user turns it on.

10. What makes the Macintosh and PC appear to work differently?

11. List three input devices.

12. List two output devices.

1.2. Machine Language and Data Representation

The only language the computer can understand is called ***Machine Language*** or **American Standard Code for Information Interchange** (abbreviated as **ASCII** code). In this language each character of the character set (characters and numerals and special symbols) are represented by a set of eight ones and zeros (called bits). **Eight bits arranged in various orders would represent various characters of the character set**. For example the character A is represented in this language by 01000001 and B is represented by 01000010. Instructions in the program written for the computer also have to be coded in this language (using bits). Early computer scientists had to design such a coding system because the character sets cannot be represented inside the computer in the form of the symbols that we use to identify them. Only the bits can be represented inside the computer because computers can only distinguish two different things. We shall learn some details of this language in this section, and then we will see how these codes can be represented inside the computer. In the ASCII language, numerals are coded by their binary representations. We shall begin by learning the binary representation of numerals.

Binary Coding of Numerals

Any data requires some coding system to represent it. To represent quantities we humans use numbers that are coded in the coding system called ***decimal coding system***. In decimal coding system, we use the symbol 5, for example, to represent the count five, and we write the numbers 5 and 2 side by side as 52 to represent fifty-two. The actual value of the number that is written decimal code as 52 is described as follows:

$$52 = 5\times10+2 \ = 5\times10^1+2\times10^0.$$

Similarly, the actual value of the decimal code 2354 is

$$\mathbf{2354} = \mathbf{2}\times1000 + \mathbf{3}\times100 + \mathbf{5}\times10 + \mathbf{4}\times1$$
$$\mathbf{2354} = \mathbf{2}\times10^3 + \mathbf{3}\times10^2 + \mathbf{5}\times10^1 + \mathbf{4}\times10^0$$

As these examples illustrate, in the decimal coding system each digit represents how many thousands, hundreds, tens, and ones are there in that number. That is, how many 10^3 s, 10^2 s, 10^1 s, and 10^0 s are there in that number. Number **10 is said to be the base of this system**. In the decimal coding system, to represent a number we use the digits that appear in front of 10^3, 10^2,10^1, and 10^0 in their expansions (as shown in the right sides of the above equations). Note that the necessary digits to represent any number in this coding system are the ten symbols 0,1,2, . . . ,9.

Why should we use 10 as the base? There is no specific reason, except that 10 seems to be a convenient base because powers of 10 are easy to compute. Using 10 as base calls for using 10 symbols in the code. Instead of 10 if we use 2 as the base of the number coding system, then we get what is known as binary coding system. **To**

determine the binary coding of a number, we need to express the number as a sum of multiples of powers of 2 (that is, . . ., 2^3, 2^2,2^1, and 2^0). Then the numbers that appear in front of . . . ,2^3, 2^2,2^1, and 2^0 would make up the binary code for the given number. When 2 is the base, all the numbers that go in front of . . . ,2^3, 2^2,2^1, and 2^0 would only be two: 0 and 1. The following two examples illustrate.

Example 1.2.1. To determine the binary code of the decimal number 13, express it as

$$13 = \underset{\uparrow}{\mathbf{1}}\times2^3 + \underset{\uparrow}{\mathbf{1}}\times2^2 + \underset{\uparrow}{\mathbf{0}}\times2^1 + \underset{\uparrow}{\mathbf{1}}\times2^0.$$

These digits make up the binary code

Thus the binary code of 13 is **1101**.

Example 1.2.2. Let us determine the binary code of the decimal number 37.

The highest power of 2 that has a value smaller than 37 is 2^5 (which is equal to 32). Therefore our expression for 37 would begin with 2^5 and decrease in power down to 2^0. You begin by writing a 1 in front of 2^5 and then determine which powers of 2 should have 1 in front of them and which ones should have 0 so that the entire expression simplifies to equal 37.

$$37 = \mathbf{1}\times2^5 + \mathbf{0}\times2^4 + \mathbf{0}\times2^3 + \mathbf{1}\times2^2 + \mathbf{0}\times2^1 + \mathbf{1}\times2^0.$$

Therefore the binary code of 37 is **100101**.

Warm-up Exercise 1. Determine the binary code of 19 and 65 using the above approach.

Shorter Method for Binary Code

A shorter approach to finding the binary code is to repeatedly divide the given number by 2, writing the answers and remainders as shown in the following example. Terminate the repeated division when the answer becomes zero. Then the remainders, written in the reverse order (that is, from bottom up) would make up the binary code of the number.

Example 1.2.3. Let us find the binary code of 37 using the short cut approach

```
          remainders
   2 | 37
    2 |18  -  1
     2 | 9  -  0
      2 |4  -  1
      2 |2  -  0
      2 |1  -  0
          0  -  1
```

Writing the remainders starting with the bottom one first, we get the binary code of 37: **0100101**. Note that 0s in the front may or may not be included.

Finding Decimal Equivalent of a Binary Coded Number

To find the decimal equivalent of a binary coded number, you do the above process in the opposite way. For example, to find the decimal equivalent of the binary code **101101**, you would begin by writing 1×2^0 as the decimal value of the last bit, and add 0×2^1 as the decimal value of the bit **0** in front of it and so on, arriving at the expression:

$$\mathbf{1} \times 2^5 + \mathbf{0} \times 2^4 + \mathbf{1} \times 2^3 + \mathbf{1} \times 2^2 + \mathbf{0} \times 2^1 + \mathbf{1} \times 2^0.$$

Simplifying this expression yields 45 as the decimal equivalent of the binary number 101101.

Warm-up Exercise 2.

(a) Find binary code of 17, 35, and 86

(b) Find decimal codes of 10010, 110101, and 0010101

The ASCII codes

ASCII codes for numbers are same as their binary codes discussed above. For the characters, a coding system was developed using eight 1s and 0s for each character. These 1s and 0s in this coding system are referred to as **bits**.

Table 1.1 below shows some examples of characters, their ASCII codes, and the decimal equivalents of these ASCII codes (that is, the decimal numbers whose binary codes are the same as the specific ASCII codes) which are called the decimal codes of these characters.

Table 1.1. ASCII and Decimal Codes

Character	ASCII code	Decimal code
A	01000001	65
B	01000010	66
C	01000011	67
D	01000100	68
E	01000101	69
Z	01011010	90
a	01100001	97
b	01100010	98
c	01100011	99
d	01100100	100
e	01100101	101
z	01111010	122

A complete list of all the characters and commonly used key combinations and their decimal codes is provided in Appendix B.

Representing Character Data in The Computer Memory

The computer is an electrical equipment just like a television or a VCR. How does it understand numbers, characters and the instructions given in a program? **The idea is to represent each character and number by using a set of electric circuits**. This is because since a machine can recognize electricity, it can distinguish a circuit that has electricity flowing through (an on-circuit) and another one that has no electricity flowing through (an off-circuit).

A ***circuit*** is a loop of wire through which electricity can flow. Electricity passes through a circuit, only when it is complete and connected to the power source. Such a circuit is said to be **turned on**. If any part of the circuit is disconnected, no electricity would pass through it, and the circuit is said to be **turned off**.

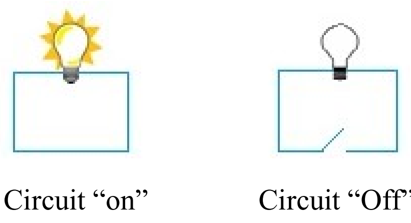

Figure 1.3. "On" and "off" settings of circuits

Inside the computer (more precisely inside the RAM) thousands of circuits are networked together in pieces of silicon (a substance that does not conduct electricity). These pieces are called **silicon chips**, and they are about one inch squares. These chips make up the RAM.

To represent a character in the RAM, 8 circuits are turned on or off according to what the ASCII code of the character is. To represent each 1 in the ASCII code, a circuit gets turned "on", and to represent each 0 in the ASCII, a circuit gets turned "off".

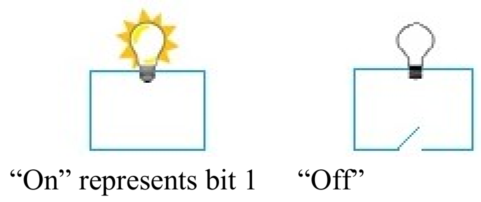

Figure 1.4. Circuit settings that represent 1 and 0

For example, to represent the character B, whose ASCII code is 01000010, a network of eight circuits are set in the manner shown in Figure 1.5 below.

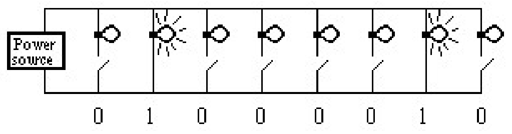

Figure 1.5. The character data 'B' stored in the memory

A byte of the memory

A byte represents the amount of memory needed to store one character data. As noted earlier, to represent one character, we need a group of 8 circuits. Therefore, a byte consists of 8 circuits (or 8 bits). Therefore,

1 byte = 8 bits.

When we input a single character data (by typing it in the keyboard, for example), inside the RAM a network of 8 circuits would be selected and each circuit would be set "on" or "off" according to the ASCII code of the character.

The job of each key on the keyboard is turning the circuits in a byte on and off according to the ASCII code of the character it represents whenever it is pressed. The following example illustrates this.

Example 1.2.4. When we type the character 'a' by pressing the key labeled "a" in the keyboard, it sends a signal to the computer to set the next 8 available memory circuits as shown in Figure 1.6. That is, 8 circuits of memory get set to represent the ASCII code 01100001 of the character 'a'.

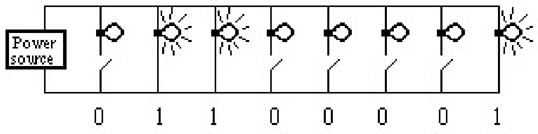

Figure 1.6. The character 'a' stored in the memory

Millions of circuit-networks that constitute bytes are put together in memory chips that make up the RAM.

It is now clear why RAM is erased when the power is turned off. When the power is turned off all the circuits become off, and therefore no data is represented inside the RAM anymore.

Secondary storage devices such as hard-disks and CDs use a different method to permanently store data. They do not need electric power to hold the data. However, their storage capacities are also measured using the same units. For any data storage medium (memory or secondary storage device) a **byte of memory is the space needed to store one character of data**. Since computer memory should be sufficient for storing data being processed and the programs that are being executed, it needs to be very large. The measure “byte” is too small to measure large memory or secondary storage. Bigger units to measure memory and secondary storage spaces are defined as follows.

One **kilo byte** (1K)	= approximately 1000 bytes.
One **mega byte** (1MB)	= approximately 1000 kilobytes
	= approximately 1 million bytes.
One **giga byte** (1GB)	= 1000 mega bytes.
One **terra byte** (1TB)	= 1000 giga bytes

Storing Numeric Data in the Computer Memory

As noted earlier, each character data has 8 bits in its ASCII code, and therefore each character can be stored in one byte of memory (or one byte of any storage-media). Machine language code of a number is its binary code, and therefore, when a number is stored in the memory its binary code gets stored by setting circuits "on" and "off" to represent the 1s and 0s in the code. **Depending on the size, numbers may require more than one byte to store them.** Only the numbers whose binary codes have 8 bits or less can be stored in one byte of memory. If the binary code of a number has more than 8 bits but less than 16 bits, then two bytes will be necessary to store that number. Similarly for larger numbers more than two bytes have to be used.

We will see later how to tell the computer in a program to allow more than one byte for a number.

For a number that has 15 bits in its binary code, for example, two bytes of memory will be used, and the binary form of the number will be stored by storing **the last eight bits in the second one of the two bytes and the remaining bits will be stored in the first byte** padding with 0s at its front if necessary.

Example 1.2.5. Find the binary code of 1279 and determine how many bytes of memory is required to store that number, and show how it will be stored.

The binary code of 1279 is 10011111111, which has 11 bits. Therefore, two bytes of memory will be needed. So, the number will be stored as follows.

0000 0001	0011111111
First byte	Second byte

Example 1.2.6. Determine the largest number that can be stored in one byte of memory.

One byte of memory has 8 bits. The largest number that can be stored in one byte of memory is the number that has only eight bits in its binary code with all bits equal to 1. This is because when any bit is 0, the value of the number is lower. Hence, the largest number we can store in one byte is the number whose binary code is 11111111, whose decimal equivalent is

$$1\times2^7 + 1\times2^6 + 1\times2^5 + 1\times2^4 + 1\times2^3 + 1\times2^2 + 1\times2^1 + 1\times2^0$$

which is equal to 255.

From Example 5, it is clear that

Any number larger than 255 requires more than 1 byte of computer memory.

The details of how larger numbers are stored are presented later. **Quicker way to find the largest number that fits in a memory byte** is the following**.** Recall from Algebra that the sum of the series $2^0 + 2^1 + 2^2 + \ldots + 2^n$ is $2^{n+1} - 1$. Therefore, the largest number that can be stored in one byte of memory = $2^0 + 2^1 + 2^2 + \ldots + 2^7$ which is equal to $2^{7+1} - 1 = 255$.

Warm-up Exercise 3. What is the largest number that can be stored in two bytes of memory?

Other Coding Systems

We noted earlier that when the base of the number system is 10, we call it decimal coding system, and when the base is 2 we call it binary coding system. Two other coding systems are important to computer scientists. The ***octal* coding system is one in which the base is 8**, and the ***hexadecimal* coding system is one in which the base is 16**.

Octal Coding

To find the octal code of a number, you need to express it as a sum of powers of 8. The digits that appear in front of each power of 8 make up the octal code. Only symbols that would appear in an octal code would be 0, 1, 2, 3, 4, 5, 6, and 7.

Example 1.2.7. Find the octal code of 459.

Expressing 459 as a sum of powers of 8, we have

$$459 = \mathbf{7}\times 8^2 + \mathbf{1}\times 8^1 + \mathbf{3}\times 8^0,$$

and therefore the ***octal code of 459 is*** *713*.

Quick Method for Octal Code

Just like the quick method to finding the binary code, the quick method to finding the octal code is to divide the number continuously by 8 until we get an answer less than 8.

```
8 |459
   8 |57 - 3
       7 - 1
```

The remainders written bottom-up, 713, is the octal code.

Since the base of the octal coding system is 8, the digits that will appear in the octal code of a number are 0, 1, 2, 3, 4, 5, 6, 7 (that is, all the numbers less than 8).

Hexadecimal Coding

Similar to the octal coding system, **to find the hexadecimal code of a number, we will have to express the numbers as a sum of powers of 16**. The digits that will appear in the hexadecimal code of a number would be 0, 1, ……, 9, 10, 11, 12, 13, 14, 15. The two digit numbers 10, 11, …, 15 clearly cannot be used as digits in the hexadecimal codes because if we use them one would not be able to determine whether to identify, for example, the hexadecimal coded number 1115 as $11\times 16^1 + 15\times 16^0$, or as $1\times 16^3 + 1\times 16^2$

$+ 1\times16^1 + 5\times16^0$. For this reason in hexadecimal coding system the **alphabets A, B, C, D, E and F are used in place of 10, 11, 12, 13, 14, and 15 respectively**.

Example 1.2.8. Find the hexadecimal code of the number 1279.

Expressing 1279 as sum of powers of 16, we get

$$1279 = \mathbf{4}\times16^2 + \mathbf{15}\times16^1 + \mathbf{15}\times16^0,$$

and therefore the hexadecimal code of 1279 is **4FF**.

Just as in the other coding systems, the quick method for finding the hexadecimal code is to repeatedly divide the number by 16.

A list of all the characters and their decimal, octal and hexadecimal codes are provided in Appendix B.

Exercises 1.2

1. Find the binary code of the integer 97.

2. Find the binary code of the integer 89.

3. Find the decimal code of the binary number 01011010

4. Find the decimal code of the number 10010111.

5. Find the decimal code of the number 10101100.

6. ASCII code of the character D is 01000100. What is its decimal code?

7. ASCII code of the character f is 01100110. What is its decimal code?

8. Find the binary code of the number 18. If this number is stored in the memory of the computer, show the on/off settings of the byte of the memory where it is stored.

9. Find the binary code of the number 161 and determine how many bytes are needed to store it.

10. Find the binary code of the number 265 and determine how many bytes are necessary to store this number.

11. How many bytes of memory is needed to store the word SUNSHINE?

12. How many bytes of memory is needed to store the word EASTERN?

13. How many bytes are in a megabyte?

14. A diskette has 1.44 mega byte of storage. How many characters can be stored in this diskette.

15. Determine the largest number that can be stored in three bytes of memory.

16. Determine the largest number that can be stored in two bytes of memory.

17. Find the octal code and hexadecimal code of 3245.

18. Find the octal code of 7259.

19. Find the hexadecimal code of 3342

20. Find the hexadecimal code of 2834.

21. Find the decimal equivalent of the hexadecimal number 2C8.

22. Find the decimal equivalent of the hexadecimal number E7B.

1.3. Programming Languages and Compilers

Programming Languages

As we have seen earlier, data and instructions for a computer must all be given in the machine language. Therefore early programmers had to learn the machine language to be able to write computer programs. But this language is very difficult to work with. Programs almost always have errors after writing them the first time. We have to go through it, find errors, and fix them before they will work. A program written is machine language is too difficult to read through again and figure out errors. Therefore, computer scientists designed *programming languages*.

> *A* ***programming language*** *is a language that is* ***closer to English than the machine language****, and therefore* ***convenient for humans to write programs for the computer****.*

Programming languages are also called high level languages. There are many such languages. The languages FORTRAN, BASIC, Pascal, COBOL, BASIC, Ada, C, C^{++}, C# (pronounced C-sharp) Python, and JAVA are some of them. Computers do not understand these languages. We need the aid of ***compilers*** (discussed later) to be able to execute programs written in these languages. Java is a relatively new language evolved from C++. The language C# is also a relatively new programming language, designed by Microsoft for a wide range of enterprise applications that run on the .NET Framework. The C# *language* is simple, modern, type safe and object oriented

Why not English?

Why can't we communicate with the computer in English or other natural languages? No computer can be designed to understand natural languages, because the **natural languages have too many ambiguities**.

Compiler

Asking the computer to carry out the instructions given in a program is called "executing the program" or "running" a program. **Only a program written in the machine language can be executed** because the computer understands only this language. Computer scientists have developed computer programs called ***Compilers*** (one for each programming language) to translate programs written in any programming language into one in machine language.

The **C++ Compiler is a computer program which**, when executed, **takes a program written in C++ as input and writes the same program in machine language** as output.

Compilers of other programming languages work the same way.

Executing a Program

To execute a C++ program on the computer, first the program must be translated to one in the machine language by accessing and using the ***compiler***. The process of translating a program written in a programming language into a program in machine language is called ***compiling***.

When we issue the command to execute the program, the machine follows the instructions given in the machine language version of the program.

C and C++ Languages

C++ is one of today's most popular software development languages. It is an extension of the C language. That is, it includes all of the C language features and has much more. It is the ***object oriented programming*** capabilities of C++ that make it much more powerful and desirable than C. A C program can be compiled using a C++ compiler but not the other way around.

Programming Environments

Computes are configured in wide variety of ways. They can be grouped into four kinds based on their sizes and capabilities. They are presented here in the increasing order of capabilities:

1. **Micro** computers
2. **Mini** computers
3. **Main frame** computers
4. **Super** computers.

Smallest kind of computers called **micro computers,** which include lap tops are dedicated to a single user. Examples are PC and Macintosh. **Mini computers** are more powerful and faster than micro computers. They can serve multiple users simultaneously. Larger computers, called **main frames**, can also serve numerous users simultaneously. The fourth kind of computers called **super computers**, have "parallel processing" capability and allow multiple users. That is, super computers can process many jobs simultaneously. Since mini computers and main frame computers also permit multiple users, they appear to be processing jobs in a parallel manner. But they actually process jobs one at a time. For our purposes all these computers serve the same function and behave similarly.

Exercises 1.3

1. What is the language that the computers understand?

2. Name four high level languages?

3. Can the computer understand high level programming languages? If not, why do we choose to write programs in these languages?

4. After typing a program written in C++ language into the computer, can we execute it right away? If not what do we have to do first?

5. Name the four categories of computers.

6. Name a micro computer. Name a mainframe computer.

7. Which group of computers fall in between micro and mainframe?

8. Why can't English be a programming language?

Programming Projects 1.3

(Compiling, executing, and understanding syntax errors)

Project 1.3.1. The program given below is written to compute the product of the two numbers 32.5 and 124.8. Note that any line that begins with double slash in a C++ program is a comment to be read by any human reader of the program, and it is ignored by the compiler. Your name must be written as a comment in the program as indicated in the program.

Program:

```
#include <iostream>
using namespace std;
//  Programmer:Type your full name here.
// This program computes the product of the numbers 32.5 and 124.7.
  main()
      {
        float  n1=32.5, n2=124.7, product;
        int  num;
        product = n1  * n2;
        cout << "Product is : " << product << "\n";
        system("pause");
       }
```

Part 1 of the project: Type this program using the computer, save it on your disk, compile it and execute it. When the program works correctly, it should produce the following output on the screen.

```
Product is : 4052.75
```

The **system("pause")** statement is only necessary when using some compilers to make the system hold the output window open. If you type anything and press enter, the output window will close. This window cannot be printed since it doesn't have a print command associated with it. To print the display, first move the two windows, if necessary, to make sure that the program and the output are fully displayed. Then press the ***PrintScreen*** button on the keyboard. This key takes a picture of the entire screen and saves it in the clip-board of the computer. Open a word processor program and select *Paste* command from the *Edit* menu to paste the picture. Then print the picture by selecting the *Print* command from *File* menu of the word processor.

Now close the word processor, activate the output window by clicking on it, type a number, and press Enter. The output window closes.

Part 2 of the project: Change the line

```
cout << "Product is : " << product << "\n";
```

found in the program to have

```
cout << " Product is : " << product << \n;
```

(that is, remove the double quotes arround the escape sequence: \n) and compile the program again. You should get a syntax error message during compilation the program. Print the new version of the program and the window that shows the error message by following the instructions given above.

Project 1.3.2. The program given below is written to compute the *quotient* (integer component of the answer) and the *remainder* of dividing the number 349 by 8. Since 8 goes into 349 forty three times with a remainder of 5, the output of executing this program will be

```
Quotient = 43
Remainder = 5
```

```
#include <iostream>
using namespace std;
/*  Programmer:Type your full name here.
This program computes the quotient and the remainder of dividing any
given integer data by 8.  */
main()
   {
     int  n1=349, quotient, remainder, dummy;
     float decimal_answer;
     decimal_answer = n1/8.0; // Compute floating point answer of division
     quotient = decimal_answer;  //Quotient is integer component of answer
     remainder = n1 - quotient * 8;
           // Now write the results on the screen.
     cout << "Quotient = " << quotient << "\n"
                     << "Remainder = " << remainder;
     cout << "Enter a number when finished reading:";
     cin >> dummy;
     return 0;
   }
```

Part 1 of the project: Type this program using the computer, save it on your disk, compile it and execute it. When the program works and produces the correct answer, you need to print the display. To print the display, first move the two windows, if necessary, to make sure that the program and the output are fully displayed. Then press the *PrintScreen* button on the keyboard. This key takes a picture of the entire screen and saves it in the clip-board of the computer. Open a word processor program and select

Paste command from the *Edit* menu to paste the picture. Then print the picture by selecting the *Print* command from *File* menu of the word processor.

Now close the word processor, activate the output window by clicking on it, type a number, and press Enter. The output window closes.

Part 2 of the project: After printing the program, change the line

```
remainder = n1 - quotient * 8;
```

found in the program to have

```
remainder = n1 - quotient * 8
```

(that is, remove the semicolon found at the end of the line) and compile the program again. You should get a syntax error message from the compiler. Print the new version of the program and the window that displays the error message by following the same instructions given above.

Turn in the original program and the output, and the modified program with the error message.