Contents

DATA REPRESENTATION IN A COMPUTER

Chapter outline

Introduction

In Book 1 of this series, we learnt that computers are classified according to *functionality, physical size* and *purpose.* We saw that when classified according to functionality, computers can be *analog, digital* or *hybrid.* Digital computers process data that is in discrete form while analog computers process data that is continuous in nature. Hybrid computers, on the other hand can process both discrete and continuous data.

In digital computers, the user input is first converted and transmitted as electrical pulses that can be represented by two distinct digits' l' and '0' before processing. These two digits are referred to as *binary* digits or in short *bits.*

Although two graphs can look different in their appearance, they may repeat themselves at equal time intervals. Electronic signals or waveforms of this nature are said to be *periodic.* Generally, a periodic wave representing a signal can be described using the following parameters.

1. Amplitude (A)
2. Frequency (f)
3. Periodic time (T)

Amplitude (A): Amplitude is the maximum value a wave can attain. For example, the amplitude of waves in Figure 1.1 is 1.

Frequency (f): Frequency of a wave is the number of cycles made by the wave in one second. It is measured in units called hertz (Hz). 1Hz is equivalent to 1 cycle/second.

Periodic time (T): The time taken by a signal to complete one cycle is called periodic time. Periodic time, T, is given by the formula $T = 1/f$ where f is the frequency of the wave.

When a digital signal is to be sent over analog telephone lines e.g. e-mail, it has to be converted to analog signal. This is done by connecting a device called a *modem* to the digital computer. This process of converting a digital signal to an analog signal is known as *modulation.* On the receiving end, the incoming analog signal is converted back to digital form in a process known as *demodulation.*

Concepts of data representation in digital computers

Since digital computers are the most widely used, this book seeks to explain in details how data is represented in digital form.

Data and instructions cannot be entered and processed directly into computers using human language. Any type of data be it numbers, letters, special symbols, sound or pictures must first be converted into *machine readable form* i.e. binary form. Due to this reason, it is important to understand how a computer together with its peripheral devices handle data in its electronic circuits, on magnetic media arid in optical devices.

Data representation in electronic circuits

Electronics components, such as the microprocessor, are made up of millions of electronic circuits. The availability of a *high voltage* (on) in these circuits is interpreted as 'I' while a *low voltage* (off) is interpreted as a '0'. This concept can be compared to switching on and off of an electric circuit. (Figure 1.3). When the switch is closed, (Figure 1.3 (a)), the high voltage in the circuit causes the bulb to light (' l' state). On the other hand, when the switch is open (Figure 1.3 (b)), the bulb goes off ('0' state).

Data representation on magnetic media

The presence of a magnetic field in one direction on magnetic media is interpreted as 'I', while the field in the opposite direction is interpreted as '0'. Magnetic technology is mostly used on storage devices which are coated with special magnetic materials such as *iron oxide.* Data is written on the media by arranging the *magnetic dipoles* of some iron oxide particles to face in the same direction and some others in the opposite direction. Figure 1.4 shows how data is recorded on the surface of a magnetic disk. Note that the dipoles on the track are arranged in groups facing opposite directions.

Data representation on optical media

In optical devices, the presence of light is interpreted as '1' while its absence is interpreted as '0'. Optical devices use this technology to read or store data. Take an example of a CD-ROM. If the shiny surface is placed under a powerful microscope, the surface can be observed to have very tiny holes called *pits.* The areas that do not have pits are called *land* (Figure 1.5).

In Figure 1.5 (a) the laser beam reflects from the land which is interpreted as '1' while in Figure 1.5 (b) the laser beam enters a 'pit' and is not reflected. This is interpreted as '0'. The reflected pattern of light from the rotating disk falls on a receiving photoelectric detector that transforms the patterns into digital form.

Reason for use of binary system in computers

It has proved difficult to develop devices that can understand or process natural language directly due to the complexity of natural languages. It is, however, possible to develop devices that can understand binary language. Devices that read, process and output data in digital form are used in computers and other digital devices such as calculators. Binary logic has therefore simplified the technology needed to develop both hardware and software systems. Other reasons for the use of binary are that digital devices are more reliable, small in size and use less energy as compared to analog devices.

Bits, bytes, nibble and word

The terms *bits, bytes, nibble* and *word* are used widely in reference to computer memory and data size. Let us explain each term.

Bits: A bit can be defined as binary digits that can either be 0 or 1. It is the basic unit of data or information in digital computers.

Byte: A group of bits (often 8) used to represent a character is called a byte. A byte is considered as the basic unit of measuring memory size in computers.

A nibble: Half a byte, which is usually a grouping of 4 bits is called a nibble.

Word: Two or more bytes make a word. The term *word length* is used as a measure of the number of bits in each word. For example a word can have a length of 16 bits, 32 bits, 64 bits etc.

Types of data representation

Computers not only process numbers, letters and special symbols but also complex types of data such as sound and pictures. However these complex types of data take a lot of memory and processor time when coded in binary form. This limitation necessitates the need to develop better ways of handling long streams of binary digits. Higher number systems are used in computing to reduce these streams of binary into manageable form. This helps to improve the processing speed and optimise memory usage.

Number systems and their representation

As far as computers are concerned, number systems can be classified into four major categories:

1. Decimal number system.

2, Binary number system

3. Octal number system.

4. Hexadecimal number systems.

Let us now consider each number system and its representation.

Decimal number system

The term decimal is derived from a Latin prefix *deci* which means ten. Decimal number system has ten digits ranging from 0-9. Because this system has ten digits, it is also called a *base ten number system* or *denary number system,*

A decimal number should always be written with a subscript 10 e.g. X_{IO}

But since this is the most widely used number system in the world, the subscript is usually understood and ignored in written work. However, when many number systems are considered together, the subscript must always be put so as to differentiate the number systems.

The magnitude of a number can be considered using three parameters.

1. Absolute value.
2. Place value or positional value.
3. Base value.

The *absolute value* is the magnitude of a digit in a number. For example, the digit 5 in 7458 has an absolute value of 5 according to its value in the number line as shown in the Figure 1.6.

The *place value* of a digit in an number refers to the position of the digit in that number i.e. whether "tens", "hundreds", "thousands" etc. as shown in Table 1.1.

Table 1.1

Thousands, 103	Hundreds, 102	Tens, 101	Ones, 10°
7	4	5	8

The total value of a number is the sum of the place value of each digit making the number. For example, the total value of the digits in Table 1.1 can be worked out by as shown below:

```
7 x 1000 = 7 000
4x 100     = 400
5 x 10    =        50
8  x   l     =         8
Total =           7458
```

The *base value* of a number also known as the radix, depends on the type of number system that is being used. The value of any number depends on the radix. For example the number 100_{10} is not equivalent to $100_{2,}$

Binary number system

Binary number system uses two digits namely, 1 and 0 to represent numbers. Unlike in decimal numbers where the place values go up in factors of ten, in binary system, the place values increase by factors of two. Binary numbers are written as X_2. Consider a binary number such as 1011_2. The right most digit has a place value of 1 x 2° while the left most has a place value of 1 x 2^3 as shown in Table 1.2.

Table 1.2

Place values (2n)	Eights 2^3 =8	Fours 2^2 =4	Twos 2^1 = 2	Ones 2° = 1
Binary digit	1	0	1	1

The decimal equivalent of 1011_2 can be worked out as shown below.

1 x 8 = 8   Ox4 = 0   1 x 2 = 2   l x l =1   Total = 11

;Octal number system

The octal number system consists of eight digits running from 0 - 7. The place value of octal numbers go up in factors of eight from right to left as shown in Table 1.3. For example to represent an octal number such as 724\, we proceed as follows:

Table 1.3

Place values	8^3 =512	8^2 =64	8^1 = 8	8° = 1
Octal digit	7	2	4	5

The decimal equivalent can be worked out as follows:

7 x 512 = 3 584   2 x 64 = 128   4 x 8=32 5xl = 5   Total = 3749

Hexadecimal number system

This is a base sixteen number system that consist of sixteen digits ranging from 0 - 9 and letters A - F where A is equivalent to 10, B to 11 up to F which is equivalent to 15 in base ten system. The place value of hexadecimal numbers goes up in factors of sixteen as shown in Table 1.4. Table 1.5 gives digits for base 10 and base 16.

Table 1.4

Place value	16^2= 256	16^1 = 16	16° = 1
Hexadecimal digit	9	4	6

Table 1.5

Base 10	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Base 16	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

A hexadecimal number is usually denoted using 16 as a subscript or capital letter H to the right of the number. For example, 94B can be written as 94BI6 or 94BH.

The decimal equivalent of94BI6 can be worked out as shown below.

9 x 256 =2304   4 x 16 = 64   11xl =11 total = 2379

Further conversion of numbers from one number system to another

So far, we have looked at the four types of number systems and introduced their basic concepts in a general and limited way. However, in this section, we shall have a detailed look at how to convert numbers from one system to another. The following conversions will be considered.

1. Conversion between binary and decimal numbers.
2. Converting octal numbers to decimal and binary form.
3. Converting hexadecimal numbers to decimal and binary form.

Conversion between binary and decimal numbers

Converting binary numbers to decimal numbers

To convert a binary number to decimal number, we proceed as follows:

1. First write the place values starting from the right hand side.
2. Write each digit under its place value.
3. Multiply each digit by its corresponding place value.
4. Add up the products. The answer will be the decimal number in base 10.

Converting decimal numbers to binary

To convert a decimal number to binary, there are two possible methods, the long division method and the place value method.

In *long division method,* the decimal number is continuously divided by 2. However, at each level of the division, the remainder which is either a 1 or 0 is written to the right of the quotient. Starting from bottom upwards, read the series of the remainder digits. The series of 1 's and O's obtained represent the binary equivalent of the number.

To convert a decimal number to a binary number using place value method proceed as follows:

Write down the place values in factors of 2 up to the value immediately larger or equal to the number being considered. For example, to convert 247_{10} into binary, we write" down the place values up to 28 i.e.256. Similarly to convert 258_{10}' write down the place values up to 29 i.e. 512. If the number being considered is itself a factor of 2 such as 64, 128, 256 etc., then place values should be written up to the number itself.

Let us now convert 247_{10} to binary. Starting from the left as shown in Table 1.6, subtract the place value from the number being converted. If the difference is a positive number or a 0, place a 1 in the binary digit row. If the difference is negative, place a Zero.

In Table 1.6, a 0 is placed in the binary digits row of the first column because 247 - 256 gives a negative value. The number 247 is then carried forward to the next lower place value i.e. 128.

Table 1.6

Place value	28	27	26	25	24	23	22	21	2°
Difference	247 - 256	247 - 128							
Binary digit	0								

Since 247 - 128 gives a positive difference of 119, the digit 1 is placed in the second column of the binary digit row and the difference is carried forward to the next lower place value as shown in Table 1.7.

Table 1.7

256	128	64	32	16	8	4	2	1
247 - 256	247 - 128	119 - 64						
0	1							

Table 1.8 shows the completed operation. **Table** 1.8

256	128	64	32
247 - 256	247 - 128	119 - 64	55 - 32
0	1	1	1

16	8	4	2	1
23 - 16	7-8	7-4	3-2	1 - 1=0
1	0	1	1	1

Converting a binary fraction to decimal number

A decimal number which has both an integral and fractional part is called a real number. The weight of the integral part of a real number increases from right to left in factors of I 0 while that of the fractional part decreases from left to right in factors of $10^{-x.}$ Table 1.9 shows how a real number 87.537 can be represented using the place values.

Table 1.9

Place value	10^1	10°	.	10^{-1}	10^{-2}	10^{-3}
Decimal digit	8	7	.	5	3	7
Value	80	7	.	0.5	0.03	0.007

For a binary number, the same approach as in Table 1.9 can be used, only that the place values (weight) are based on factors of 2. For example, the binary number 11.11011_2 can be represented as shown in Table 1.10.

Table 1.10

Place value	21	2°	.	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
Binary digit	1	1	.	1	1	0	1	1
Value in base ten	2	1	.	0.5	0.25	0	0.0625	0.03125

NB: When converting a real number from binary to decimal, work out the integral and fractional parts separately then combine them.

Converting a decimal fraction to binary

Remember that to convert a decimal integer to its binary equivalent we continuously divide the number by 2. In real decimal numbers, we do the same for the integral part. However to convert the fractional part to its binary equivalent, we proceed as follows:

1. Multiply the fractional part by 2 and note down the product.
2. Take the fractional part of the immediate product and multiply it by 2 again.
3. Continue this process until the fractional part of the subsequent product is 0 or starts repeating the value of the original fractional part of the number being converted:

4. The binary equivalent of the fractional part is extracted from the products by reading the respective integral digits from the top downwards as shown by the arrow in
5. Combine the two parts together to get the binary equivalent.

Converting octal numbers to decimal and binary numbers

Converting octal numbers to decimal numbers
To convert a base 8 number to its decimal equivalent we use the same method as we did with binary numbers. However, it is important to note that the maximum absolute value of an octal digit is 7. For example 982 is not a valid octal number because digits 8 and 9 are not octal digits, but 736_8 is valid because all the digits are in the range of 0 - 7. Example 1.13 and 1.14 show how to convert an octal number to a decimal number.

Converting octal numbers to binary numbers
To convert an octal number to binary, each digit is represented by 3 binary digits because the maximum octal digit i.e. 7 can be represented with a maximum of 3 digits. See Table 1.11.

Table 1.11

Octal digit	Binary equivalents
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Examples 1.15, 1.16 and 1.17 show how to convert octal numbers to binary numbers.

Converting hexadecimal numbers to decimal and binary numbers

Converting hexadecimal numbers to decimal number
To convert a hexadecimal number to its base ten equivalents, we proceed as follows:
1. First write the place values starting from the right hand side.
2. If a digit is a letter such as an 'A' write its decimal equivalent.
3. Multiply each hexadecimal digit with its corresponding place value and then add the products.
The following examples illustrate how to convert a hexadecimal number to a decimal number.
Converting hexadecimal numbers into binary numbers
Since F is equivalent to a binary number 1111_2, the hexadecimal numbers are represented using 4 digits as shown in Table 1.12.

Table 1.12

Decimal. equivalent	Hexadecimal digit	Binary equivalent
00	00	0000
01	01	0001
02	02	0010
03	03	0011
04	04	0100
05	05	0101
06	06	0110
07	07	0111
08	08	1000
09	09	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

The simplest method of converting a hexadecimal number to binary is to express each hexadecimal digit as a four bit binary number and then arranging the groups according to their corresponding positions as shown in Example 1.21.

Symbolic representation using coding schemes
In computing, a single character such as a letter, a number or a symbol is represented by a group of bits, the number of bits per character depends on the *coding* scheme used.

The most common coding schemes are the Binary Coded Decimal (BCD), Extended Binary Coded Decimal Interchange Code (EBCDIC) and American Standard Code for Information Interchange (ASCII).

Binary Coded Decimal
Binary Coded Decimal is a 4-bit code used to represent numeric data only. For example, a number like 9 can be represented using Binary Coded Decimal as 1001_2, Binary Coded Decimal system is mostly used in simple electronic devices like calculators and microwaves. This is because it makes it easier to process and display individual numbers on their Liquid Crystal Display (LCD) screens.

A standard Binary Coded Decimal, an enhanced format of Binary Coded Decimal, is a 6-bit representation scheme which can represent nonnumeric characters. This allows 64 characters to be represented. For example, letter A can be represented as 110001_2 using the standard Binary

Coded Decimal. A set of Binary Coded Decimal and standard Binary Coded Decimal code are provided in Appendix II.

Extended Binary Coded Decimal Interchange **Code (EBCDIC)**
Extended Binary Coded Decimal Interchange Code (EBCDIC) is an 8bit character coding scheme used primarily on IBM computers. A total of256 (2^8) characters can be coded using this scheme. For example, the symbolic representation of letter A using Extended Binary Coded Decimal Interchange Code is 11000001_2, See Appendix II for a detailed scheme.

American Standard Code for Information Interchange (ASCII)
American Standard Code for Information Interchange (ASCII) is a 7-bit code, which means that only 128 characters i.e. 2^7 can be represented. However manufacturers have added an eighth bit to this coding scheme, which can now provide for 256 characters. This 8-bit coding scheme is referred to as an 8-bit American Standard Code for Information Interchange. The symbolic representation of letter A using this scheme is 1000001_2, See Appendix II for more details,

»**Binary arithmetic operations**
In mathematics, the four basic arithmetic operations applied on numbers are addition, subtraction, multiplication and division. In computers the same operations are performed inside the central processing unit by the arithmetic and logic unit (ALU). However the arithmetic and logic unit cannot perform binary subtraction directly. It performs binary subtraction using a process known as *complementation.* For multiplication and division, the arithmetic and logic unit uses a method called *shifting* before adding the bits; however, because the treatment of this method is beyond the scope of this book, we shall only explain how the computer performs binary addition and subtraction.

Representation of signed binary numbers
In computer technology there are three common ways of representing a signed binary number.
1. Prefixing an extra sign bit to a binary number.
2. Using ones complement.
3. using twos complement.

Prefixing an extra sign bit to a binary number
In decimal numbers, a signed number has a prefix "+" for a positive number e.g. $+27_{10}$ and "-" for a negative number e.g. -27_{10} However in binary, a negative number may be represented by prefixing a digit 1 to the number while a positive number may be represented by prefixing a digit O. For example, the 7-bit binary equivalent of 127 is 1111111_2, To indicate that it is positive, we add an extra bit (0) to the left of the number i.e. (0) 1111111_2, To indicate that it is a negative number we add an extra bit (1) i.e. (1) 1111111_2, The problem of using this method is that the zero can be represented in two ways i.e. $(0)0000000_2$ and $(1\ )0000000_2$,

Ones complement

. The term complement refers to *a part* which together with another makes up a *whole.* For example in geometry two complementary angles add up to one right angle (90°). The idea of complement is used to address the problem of signed numbers i.e., positive and negative.
In decimal numbers (0 to 9), we talk of nine's complement. For example the nines complement of 9 is 0 that of 5 is 4 while that of 3 is 6. However, in binary numbers, the ones complement is the *bitwise NOT* applied to the number. Bitwise NOT is a unary operator (operation on only one operand) that performs logical negation on each bit. For example the bitwise NOT of 1100_2 is 0011_2 i.e. Os are negated to Is while I's are negated to O's. Likewise the bitwise NOT of 00 1 0 11 0 1 is 11010010_2 which represents -45_{10}' The bitwise NOT of 8-bit zero 00000000_2 is 11111111_2, Looking at the two numbers, the most significant digit shows that the number has a sign bit "0" for "+0" and" 1" for "-0". Like in the method of using an extra sign bit, in ones complement, there are two ways of representing a zero.

Twos complement

Twos complement, equivalent to tens complement in decimal numbers, is the most popular way of representing negative numbers in computer systems. The advantages of using this method are:

1. There are no two ways of representing a zero, as is the case with the other two methods.
2. Effective addition and subtraction can be done even with numbers that are represented with a sign bit without a need for extra circuitries to examine the sign of an operand.

The twos complement of a number is obtained by getting the ones complement then adding a 1. For example, to get the twos complement of a decimal number 45_{10}' first convert it to its binary equivalent then find its ones complement. Add a 1 to the ones complement i.e.

$45_{10} = 00101101_2$

Bitwise NOT (00101101) = 11010010

Two's complement = 11010011_2

Binary addition

The five possible additions in binary are:

1. 0+0=0
2. $0 + 1_2 = 1_2$
3. $1_2 + 0 = 1_2$
4. $1_2 + 1_2 = 10_2$ (read as 0, carry 1)
5. $1_2 + 1_2 + 1_2 = 11_2$ (read as 1, carry 1)

Binary subtraction

Direct subtraction

The four possible subtractions in binary are:

1. 0-0=0

2. 1_2-0= 1

3. 1_2 - 1_2 = 0

4. 10_2 - 1_2 = 1_2 (Borrow 1 from the next most significant digit to make 0 become 10_2, hence 10_2 - 1_2 = 1_2)

The following examples illustrate binary. Subtraction using the direct. method.

Subtraction using ones complements

The main purpose of using the ones complement in computers is to perform binary subtractions. For example to get the difference in 5 - 3, using the ones complement, we proceed as follows:

1. Rewrite the problem as 5 + (-3) to show that the computer performs binary subtraction by adding the binary equivalent of 5 to the ones complement of 3.
2. Convert the absolute value of 3 into 8-bit equivalent i.e. 00000011_2,
3. Take the ones complement of 000000 11_2 i.e. 11111100_2 which is the binary representation of -3_{10}'
4. Add the binary equivalent of5 to the one's complement of3 i.e.

$$\begin{array}{r} 000000101 \\ +\ \underline{111111000} \\ (1)00000001 \end{array}$$

Looking at the difference of the two binary numbers, you will observe that:

1. It has a ninth bit. The ninth bit is known as an *overflow* bit.
2. The result show that the difference between the two numbers is 00000001. This is not true! We know that it should be 00000010.

To address this problem in a system that uses ones complement, the overflow digit is added back to the magnitude of the 8-bit difference. Therefore the difference becomes 00000001 + 1 = 00000010, which is the correct answer.

Subtraction using twos complements

Like in ones complement, the twos complement of a number is obtained by negating a positive number to its negative counterpart. For example to get the difference in 5 - 3, using the two's complement, we proceed as follows:

1. Rewrite the problem as 5 + (-3).
2. Convert the absolute value of 3 into 8-bit binary equivalent i.e. 00000011.
3. Take the ones complement of 000000 11 i.e. 11111100.
4. Add a 1 to the ones complement i.e. 11111100 to get 11111101
5. Add the binary equivalent of 5 to the twos complement of 3 i.e.

$$\begin{array}{r} 000000101 \\ +\ \underline{111111001} \\ \underline{(1\ )000000\ 1\ 0} \end{array}$$

6. Ignoring the overflow bit, the resulting number is 00000010 which is directly read as a binary equivalent of +2.

DATA PROCESSING

Chapter outline

Introduction

Data refers to the raw facts that do not have much meaning to the user and may include numbers, letters, symbols, sound or images. Information, on the other hand, refers to the meaningful output obtained after processing the data.

Therefore the *data processing* refers to the process of transforming raw data into meaningful output i.e. information. Data processing can be done *manually* using pen and paper, *mechanically* using simple devices like typewriters or *electronically* using modem data processing tools such as computers. Electronic data processing has become so popular that manual and mechanical methods are being pushed to obsolescence.

Data processing cycle

Data processing cycle refers to input-process-output stages that data goes through to be transformed into information. It is often referred to as a cycle because the output obtained can be stored after processing and may be used in future as input. The four main stages of data processing cycle are:

1. Data collection
2. Data input
3. Processing
4. Output

Data collection

Data collection is also referred to as data gathering or fact-finding. It involves looking for crucial facts needed for processing.

Methods of data collection
Some methods of data collection include interviews, use of questionnaires, observation etc. In most cases, the data is collected after *sampling.* Sampling is the process of selecting representative elements (e.g. people, organisations) from an entire group (population) of interest. Some of the tools that help in the data collection include source documents such as forms, data capture devices such as a digital camera etc.

Stages of data collection
The process of data collection may involve a number of stages depending on the method used. These include:

Data creation: This is the process of putting together facts in an organised format. This may be in form of manually prepared document or captured from the source using a data capture device such as a bar code reader.

Data transmission: This will depend on whether data need to be transmitted via communication media to the central office.

Data preparation: This is *transcription* (conversion) of data from source document to machine-readable form. This may not be the case for all input devices. Data collected using devices that directly capture data in digital form do not require transcription.

. *Media conversion:* Data may need to be converted from one medium to another e.g. from a floppy disk to hard disk for faster input.

Input validation: Data entered into the computer is subjected to validity checks by a computer program before being processed to reduce errors at the input.

Sorting: In case the data needs to be arranged in a predefined order, it is first sorted before processing.

Data input
Data input refers to a process where the collected data is converted from human readable form to machine-readable form (binary form). The conversion takes place in the input device.

Processing
This is the transformation of input data by the central processing unit (CPU) to a more meaningful output (information). Some of the operations performed on data include calculations, comparing values and sorting.

Output
The final activity in data processing cycle is producing the desired output also referred to as information. The information can then be *distributed* to the target group or stored for future use. Distribution is making the information available to those who need it and is sometimes called

information *dissemination.* This process of dissemination may involve electronic presentation over radio or television, distribution of hard copies, broadcasting messages over the Internet or mobile phones etc.

Description of errors in data processing

The accuracy of computer output is very critical. As the saying goes, garbage in, garbage out (GIGO), the accuracy of the data entered in the computer directly determines the accuracy of the information given out.

Some of the errors that influence the accuracy of data input and information output include *transcription, computation* and *algorithm errors.*

Transcription errors

Transcription errors occur during data entry. Such errors include misreading and transposition errors.

Misreading errors

Incorrect reading of the source document by the user and hence entering wrong values bring about misreading errors. For example, a user may misread a hand written figure such as 589 and type S86 instead i.e. confusing 5 for S.

Transposition errors

Transposition errors results from incorrect arrangement of characters i.e. putting characters in the wrong order. For example, the user may enter 396 instead of369.

Transcription errors can be avoided by using modem data capture devices such as bar code readers, optical character readers, and digital cameras etc., which enter data with minimum user intervention.

Computational errors

Computational errors occur when an arithmetic operation does not produce the expected results. The most common computation errors include *overflow, truncation* and *rounding errors.*

Overflow errors

An *overflow* occurs if the result from a calculation is too large to be stored in the allocated memory space. For example if a byte is represented using 8 bits, an overflow will occur if the result of a calculation gives a 9-bit number.

Truncation errors

Truncation errors result from having real numbers that have a long fractional part that cannot fit in the allocated memory space. The computer would truncate or cut off the extra characters from the fractional part. For example, a number like 0.784969 can be truncated to four digits to become 0.784. The resulting number is not rounded off.

Rounding errors

Rounding errors results from raising or lowering a digit in a real number to the required rounded number. For example, to round off 30 666 to one decimal place, we raise the first digit after the decimal point if its successor is more than 5. In this case, the successor is 6 therefore 30.666 rounded up to one decimal place is 30.7. If the successor is below 5, e.g. 30.635, we round down the number to 30.6.

Algorithm or logical errors

An algorithm is a set of procedural steps followed to solve a given problem. Algorithms are used as design tools when writing programs. Wrongly designed programs would result in a program that runs but gives erroneous output. Such errors that result from wrong algorithm design are referred to as *algorithm* or *logical errors.*

Data integrity

Data integrity refers to the accuracy and completeness of data entered in a computer or received from the information system. Integrity is measured in terms of *accuracy, timeliness* and *relevance* of data.

Accuracy

Accuracy refers to how close an approximation is to an actual value. As long as the correct instructions and data are entered, computers produce accurate results efficiently. In numbers, the accuracy of a real number depends on the number. For example 72.1264 is more accurate than 72.13.

Timeliness

Timeliness of data and information is important because data and information have a time value attached to them. If received late, information may have become meaningless to the user. For example, information on the newspaper that is meant to invite people for a meeting or occasion must be printed prior to the event and not later.

Relevance

Data entered into the computer must be relevant in order to get the expected output. In this case, *relevance* means that the data entered must be pertinent to the processing needs at hand and must meet the requirements of the processing cycle. The user also needs relevant information for daily operations or decision making.

Threat to data integrity

Threats to data integrity can be minimized through the following ways:

1. Backup data preferably on external storage media.
2. Control access to data by enforcing security measures.

3. Design user interfaces that minimize chances of invalid data entry.
4. Using error detection and correction software when transmitting data.
5. Using devices that directly capture data from the source such as bar code readers, digital cameras, optical character readers etc.

Data processing methods.

As mentioned earlier, data can be processed manually, mechanically or electronically.

Manual data processing

In manual data processing, most tasks are done manually with a pen and a paper. For example in a busy office, incoming tasks (input) are stacked in the *"in tray"*. The processed tasks are then put in the *"out tray"* (output). The processing of each task involves a person using the brain in order to respond to queries. The processed information from the out tray is then distributed to the people who need it or stored in a file cabinet.

Mechanical data processing

Manual processing is cumbersome and boring especially when processing repetitive tasks. Mechanical devices were developed to help in automation of manual tasks. Examples of mechanical devices include the typewriter, printing press and weaving looms. Initially, these devices did not have any electronic intelligence.

Electronic data processing

For a long time, scientists have researched on how to develop machines or devices that would *simulate* some form of human intelligence during data and information processing. This was made possible to some extent with the development of electronic programmable devices such as computers.

The advent of microprocessor technology has greatly enhanced data processing efficiency and capability. Some of the microprocessor-controlled devices include computers, cellular (mobile) phones, calculators, fuel pumps, modem television sets, washing machines etc.

Computer files

A file can be defined as a collection of related records that give a complete set of information about a certain item or entity. A file can be stored manually in a file cabinet or electronically in computer storage devices. Computerized storage offers a much better way of holding information than the manual filing systems, which heavily rely on the concept of the file cabinet.

Some of the advantages of computerized filing system include:

1. Information takes up much less space than the manual filing.
2. It is much easier to update or modify information.
3. It offers faster access and retrieval of data.
4. It enhances data integrity and reduces duplication.

Elements of a computer file

A computer file is made up of three elements namely: *characters*, *fields* and *records.*

Characters A *character* is the smallest element in a computer file and refers to a letter, number or symbol that can be entered, stored and output by a computer. A character is made up of a set of seven or eight bits depending on the character-coding scheme used.

Fields

A field is a single character or collection of characters that represents a single piece of data. For example, in a student's record, the student's admission number is an example of a field.

Records

A *record* is a collection of related fields that represent a single entity. For example, in a class score sheet, details of each student in a row such as admission number, name, total marks and position make up a record.

Logical and physical files

Computer files are classified as either logical or physical.

Logical files

A *logical file* is a type of file viewed in terms of *what* data items it contains and details of what processing operations may be performed on the data items. It does not have implementation specific information like field, data types, size and file type. Logical files are discussed in system design later in the book.

Physical files

As opposed to a logical file, a *physical file* is one that is viewed in terms of *how* data is stored on a storage media and how the processing operations are made possible. Physical files have implementation specific details such as characters per field and data type for each field. Physical files are discussed later in system implementation and operation in this book.

Types of computer processing files

There are numerous types of files used for storing data needed for processing, reference or backup. The main common types of processing files include master files, transaction, reference, backup, report and sort file.

Master file

A master file is the main file that contains relatively permanent records about particular items or entries. For example a customer file will contain details of a customer such as *customer ID, name* and *contact address.*

Transaction (movement) file

A *transaction file* is used to hold input data during transaction processing. The file is later used to update the master file and audit daily, weekly or monthly transactions. For example in a busy supermarket, daily sales are recorded on a transaction file and later used to update the stock file. The file is also used by the management to check on the daily or periodic transactions.

Reference file

A *reference file* is mainly used for reference or look-up purposes. Lookup information is that information which is stored in a separate file but is required during processing. For example, in a point of sale terminal, the item code entered either manually or using a bar code reader looks up the item description and price from a *reference file* stored on a storage device.

Backup file

A *backup file* is used to hold copies (backups) of data or information from the computers fixed storage (hard disk). Since a file held on the hard disk may be corrupted, lost or changed accidentally, it is necessary to keep copies of the recently updated files. In case of the hard disk failure, a backup file can be used to reconstruct the original file.

Report file

A *report file* is used to store relatively permanent records extracted from the master file or generated after processing. For example you may obtain a stock levels report generated from an inventory system while a copy of the report will be stored in the report file.

Sort file

A *sort file* is mainly used where data is to be processed sequentially. In sequential processing, data or records are first sorted and held on a magnetic tape before updating the maste file.

File organization methods

File organization refers to the way data is stored in a file. File organization is very important because it determines the method of access, efficiency, flexibility and storage devices to be used. There are four methods of organizing files on a storage media. This includes: sequential, random, serial and indexed-sequential

Sequential file organisation

In *sequential file* organisation, records are stored and accessed in a particular order sorted using a key field. Retrieval requires searching sequentially through the entire file record by record from the beginning to the end. Because the records in the file are sorted in a particular order, better file searching methods like the *binary search* technique can be used to reduce the time used for searching a file. Since the records are sorted, it is possible to know in which half of the file a particular record being searched is located. Hence this method repeatedly divides the set of records in the file into two halves and searches only the half in which the record is found. For example, if the file has records with key fields 20, 30, 40, 50, 60 and the computer is searching for a record with key field 50, it starts at 40 upwards in its search, ignoring the first half of the set.

Random or direct file organisation

In *random or direct file organisation*, records are stored randomly but accessed directly. To access a file stored randomly, a record key is used to determine where a record is stored on the storage media. Magnetic and optical disks allow data to be stored and accessed randomly.

Serial file organisation

With *serial file organisation*, records in a file are stored and accessed one after another. The records are not sorted in any way on the storage medium. This type of organisation is mostly used on magnetic tapes.

Indexed-sequential file organisation method

This method is almost similar to sequential method, only that an index is used to enable the computer to locate individual records on the storage media. For example, on an magnetic drum, records are stored sequentially on the tracks. However, each record is assigned an index that can be used to access it directly.

Electronic data processing modes

There are several ways in which a computer, under the influence of an operating system is designed to process data. Examples of processing modes are:

1. Online processing
2. Real-time processing
3. Distributed processing
4. Time-sharing.
5. Batch processing
6. Multiprocessing
7. Multitasking
8. Interactive processing

On-line processing

In online data processing data is processed immediately it is received the computer is connected directly to the data input unit via a communication link. The data input may be a network terminal or an online input device attached to the computer.

Real-time processing

In a *real-time data processing*, computer processes the incom111g data as soon as it occurs, up-dates the transaction file and gives an immediate response that would affect the events as they happen. This is different from online in that for the latter an immediate response may not be required. The main purpose of a real-time processing is to provide accurate, up-to-date information hence better services based on a true (real) situation. An example of real-time processing is making a reservation for airline seats. A customer may request for an airline booking information through a remote terminal and the requested information will be given out within no time by the reservation system. If a booking is made, the system immediately updates the reservations file to avoid double booking and sends the response back to the customer immediately.

Distributed data processing

Distributed data processing refers to dividing *(distributing)* processing tasks to two or more computers that are located on physically separate sites but connected by data transmission media. For example, a distributed database will have different tables of the same database residing on separate computers and processed there as need arises. The users of the distributed database will be completely unaware of the distribution and will interact with the database as if all of it was on their computer.. This distribution of processing power increases efficiency and speed of processing. An example is in the banking industry where customers' accounts are operated on servers in the branches but all the branch accounts can be administered centrally from the main server as if they resided on it. In this case, we say that the distributed database is *transparent* to the user because the distribution is hidden from the user's point of view.

Time-sharing

In a *time-sharing processing,* many terminals connected to a central computer are given access to the central processing unit apparently at the same time. However in actual sense, each user is allocated a *time slice* of the CPU in sequence. The amount of time allocated to each user is controlled by a multi-user operating system. If a user's task is not completed during the allocated time slice, he/she is allocated another time slice later in a round robin manner.

Batch processing

In *batch processing,* data is accumulated as a group (batch) over a specified period of time e.g. daily, weekly or monthly. The batch is then processed at once. For example in a payroll processing system, employees' details concerning number of hours worked, rate of pay, and other details are collected for a period of time, say one month. These details are then used to process the payment for the duration worked. Most printing systems use the batch processing to print documents.

Multiprocessing

Multiprocessing refers to the processing of more than one task at the same time on different processors of the same computer. This is possible in computers such as mainframes and network servers. In such systems, a computer may contain more than one independent central processing unit, which works together in a coordinated way. At a given time, the processors may execute instructions from two or more different programs or from different parts of one program simultaneously. This coordination is made possible by a multiprocessing *operating system* that enables different processors to operate together and share the same memory.

Multiprogramming

Multiprogramming, also referred to as *multi-tasking* refers to a type of processing where more than one programs are executed apparently at the same time by a *single central processing unit.* It is important to note that, as opposed to multiprocessing. In multiprogramming, a computer has only one central processing unit. The operating system allocates *each program a time slice* and decides what order they will be executed. This scheduling is done so quickly that the user gets the impression that all programs are being executed at the same time.

Interactive processing

In *interactive data processing*, there is continuous dialogue between the user and the computer. As the program executes, it keeps on prompting the user to provide input or respond to prompts displayed on the screen.

ELEMENTARY PROGRAMMING PRINCIPLES

Chapter outline

Introduction

Human beings have evolved from the Stone Age to a highly sophisticated and advanced society by inventing things like the wheel, fire, transistors and today's ultra modem devices like computers. The idea of computers started way back in the nineteenth century.
The first generation computers called *Electronic Numeric integrator and Calculator* (ENIAC 1) were operated by plugging wires into a control panel that resembles the old telephone switchboards

Computer programming

A computer works by executing a set of instructions known as a program. The term *programming* refers to the process of developing computer instructions (programs) used to solve a particular task. It involves use of special characters, signs and symbols found in a particular *programming language* to create computer instructions. A programming language is a special set of symbols that can be translated into machine-readable form by the computer when arranged in a particular sequence or order. Each language has a special sequence or order of writing characters usually referred to as *syntax.*
It was John von Neumann of Princeton University (USA) who first came up with the proposition to store programs in the computer memory. Perhaps, this was one of the most dramatic developments in the computer history. Why? Because not only did the new *stored-program* way of computing increase-processing speed but also allowed easy and flexible methods of editing and updating the program.

Description of terms used in programming

Before we go further with programming, it is important to define some common terms used in computer programming.

Source program

The term source program refers to the program code that the programmer enters in the program editor window that is not yet translated into machine-readable form. The source program is usually created using a particular programming language as discussed later.

Object code

The term object code refers to the program code that is in machine-readable. A source code that is not in machine-readable form must be translated into object code.

Translators

The term translator is used to refer to language processors such as *assemblers*, *interpreters* and *compilers* that convert the source program into object code.

Assembler

An assembler translates *assembly language* into machine language that the computer can understand and execute.

Interpreter

An *interpreter* translates the source program line-by-line, allowing the CPU to execute one line before translating the next. The translated line is not stored in the computer memory. It means that every time the program is needed for execution, it has to be translated. This method of translating programs was very common in early computers that did not have enough memory to store the object code as a file that can be executed later

Compiler

A *compiler* translates the entire source program into object code: The object code file can be made into a fully executable program by carrying out another process known as *linking* which joins the object code to all the other files that are needed for the execution of the program. After the linking process, an executable file (application file) is generated. This file is stored on a storage media such as a disk with a name that has a unique extension *(.EXE)*. Examples of executable files are WINWORD.EXE and PM70.EXE used to start Microsoft Word and Adobe PageMaker 7.0 respectively.

The difference between the interpreters and compilers are summarised below:

Interpreters	*Compilers*
1. Translates the source program one statement at a time	1. Translates the entire source code at once before execution
2. Translates the program each Time it is run hence slower than Compiling	2. Compiled program (object code) can be saved on a storage media and run as required, hence Executes faster than interpreted Programs.

3. Interpreted object code takes Less memory compared to Compiled program.	3. Compiled programs require More memory as the object file Are larger.

Levels of programming languages

Many programming languages have been developed over the years. These languages are classified into two major levels namely:

1. Low-level languages
2. High-level languages

These levels are further subdivided into five generations. The first and second generations consist of low-level languages while the third to the fifth generation consist of high-level languages.

Low-Level languages

Low-level languages are classified as low because the computer can easily understand them directly or they require little effort to translate into computer understandable form. These languages are hardware oriented and therefore they are not portable i.e. a program written for one computer cannot be installed and used on another. Two types of low level languages are the *machine* languages and *assembly* languages.

Machine languages (First generation languages)

In machine languages, instructions are written using binary logic. Given that data and instructions are in binary form, many lines of code are needed to accomplish even a simple task like adding two numbers. A program written in machine language might look like this:
It is evident from the above code that it is hard for a person to guess what the program is all about unless they have special knowledge in machine level programming. Furthermore, different CPU's have different machine codes e.g. those for the *Intel Pentium* processors may differ from *Motorola or Cyrix processors.* Therefore before decoding the meaning, a programmer has to know for which CPU the program was written. Obviously, such programs are hard to understand from the programmer's point of view, but very easy to execute from the computer's perspective.

Assembly languages (Second generation languages)

Assembly languages were developed in order to overcome the difficulties of understanding and using machine languages. These languages represented the first successful attempt to make computer languages readable. The languages allowed programmers to write programs as a set of symbolic operation codes called *mnemonics.* Mnemonics are basically shortened two or three letter words. Programs written in assembly language require an *assembler* in order to convert them into machine language that the computer can understand. Just like the machine languages, assembly languages are also machine dependent and therefore a program written for one computer cannot be used on another.

High-level languages

High-level languages are very close to the human language (English like) and they can be read and understood even by people who are not experts in programming. There are many types of high-level languages and each of them was developed to address a particular problem-solving domain while others came about due to advancement in technology. These languages are *machine independent.* This means that a programmer concentrates on problem solving during a programming session rather than how a machine operates.

High-level languages can be classified into five groups:

1. Third generation languages. (3 GLs)
2. Fourth generation languages. (4 GLs)
3. Fifth generation languages. (5 GLs)
4. Object oriented languages. (OOPs)
5. Web scripting languages.

Third generation languages (3 GLs)

Third generation languages (3 GLs) are also called *structured* or *procedural* languages. A procedural language makes it possible to break a program into components called *modules* each performing a particular task. This is referred to as structured programming. The structured programming approach emphasizes the following:

1. Large programs can be broken down into smaller sub programs each performing a single task.
2. Use of a few simple control structures in problem solving. These control structures include *sequence, selection* and *iteration* as covered later in this book.

Structured programming offers many benefits because it is flexible, easier to read and modify.
Examples of third generation programming languages include:
Pascal: Pascal was initially developed as an academic language, to help in the teaching and learning of structured programming.

FORTRAN: (FORmula TRANslator): This language was developed for mathematicians, scientists and engineers. It enables writing of programs with mathematical expressions.
COBOL: (Common Business Oriented Language): This language is designed for developing programs that solve business problems e.g. developing data processing applications such as computer-based inventory control systems.

BASIC: (Beginners All-purpose Symbolic Instructional Code): This language was mainly developed to enable students to easily learn programming. Basic is a simple general-purpose language used for developing business and educational applications. Because of its simplicity, it is a powerful tool for students who wish to learn programming. It was the first high-level language that was available for microcomputer users.
C: This is a programming language mainly used for developing system software such as the operating system. It is one of the most popular and powerful high-level languages in the business world because of its ability to provide the programmer with powerful features of low-level languages and at the same time easily understandable as a high level language.

Ada: This language was named after the first lady programmer, Ada Lovelace. Ada is suitable for developing *military, industrial* and *real time* systems. .

A sample program written in Pascal language

Imagine the task of developing a program that would solve the equation of a straight line given by the algebraic expression:

Y = MX + C.

To enter the program code below in Pascal, proceed as follows:

1. From Windows explorer, locate a folder called TP and open it.
2. From the TP folder select BIN
3. From BIN window, double click a file named turbo.EXE. Pascal program window is displayed on the screen.
4. Enter the program code exactly as it is but ignore the numbering.

```
1. Program Straight Line (Input, Output);
2. Var
3. y, m, x, c: Integer;
4. Begin
5. Writeln ('Input the value of m');
6. Readln (m);
7. Writeln ('Input the value of x');
8. readln (x);
9. Writeln ('Input the value of c ');
10. Readln (c);
11. Y: = m * x + c;
12. Writeln (The value of y is:’, y);
13. End.
```

Explanation

Line 1: This is the program header. The word "Program" indicate the beginning of the program whose name is *Straight-Line.* The *(input, output)* statements shows that the program will expect some input from the keyboard and display the output on the screen.

Line 2: *Var* is short form for *variable.* A variable is a location for data in the computer memory. This statement tells the computer that variables are about to be *declared.* When a variable is declared, the computer sets aside some memory space to store a value in the variable.

Line 3: F our variables have been declared of *type integer.* This means that the memory spaces that will be set aside can only hold values that are whole numbers.

Line 4: The *Begin* statement shows that this is the start of the program body. The computer executes statements in this section. For example, the execution starts by asking the user to input the value of m.

Line 5: The *writeln* statement displays whatever is in the brackets on the screen. Notice that the statements in brackets are written between inverted commas. The string will be sent to the screen exactly the way it appears in the brackets. If you wish to display the value held in a variable on

the screen, then you have to remove the inverted comma's and write the name of the variable in the brackets e.g. *writeln (y)* will display the value held in the variable y.
Line 6:The *read* or *readln* statement reads a value and stores it in a variable. When the program is running, a read/readln statement in the code will display a blinking cursor that indicates to the user where to type the input.
Line 11: Calculates the value of y. Notice the symbol': ='. In Pascal! This is called the assignment statement. The values on the righ1 are calculated then stored in the variable y, which is on the left of the assignment symbol.
Line 12: The writeln displays the value stored in yon the screen. Notice that y is not within the inverted commas. If you gave the value of m as 10, x as 2 and c as 20 then the following should appeal on your screen: *The value of y is: 40.*
This is because the statements between the inverted comma' are meant to make the output readable on the screen. Otherwise: only the value 40 would have been displayed.
Line 13: The 'End.' statement shows the end of a program.

Fourth generation languages (4 GLs)
Fourth generation languages make programming an even easier task that the third generation languages because they present the programmer with more programming tools. Examples of such tools include command buttons, forms etc. With the advent of these languages, gone are the days when a person had to write lines upon lines of code. Instead, the programmer selects graphical objects on the screen called *controls* then uses them to create designs on a base *form.* The programmer may also use an *application generator* that works behind the scenes to generate the necessary code; hence the programmer is freed from the tedious work of writing the code. Examples of fourth generation languages are: *Visual Basic, Delphi Pascal* and *Visual COBOL.*

A sample form designed using Visual basic programming language
The programmer in this case simply picks a tool from the *toolbox* and uses it to create objects such as command buttons, textboxes etc. on the form by dragging the mouse pointer on the form during design.
Fifth generation languages (5 GLs)
Fifth generation languages are designed around the concept of solving problems by enabling the computer to depict human like intelligence. These programs are designed to make the computer solve the problem for the programmer rather than the programmer spending a lot of time to come up with the solution. With such languages, the programmer only worries about what problem needs to be solved and what conditions need to be met without worrying about how to implement an *algorithm* to solve them. Examples of these languages are those used in artificial intelligence like PROLOG, Mercury, LISP and OCCAM.

Object-oriented programming languages (OOP)
The idea behind object-oriented programming (OOP) was developed in the 1960's but its significance was not appreciated until lately. The concept behind object oriented programming languages is to look at a program as having various objects interacting to make up a whole. Each object has specific data values that are unique to it (called *state)* and a set of the things it can accomplish called *(functions or behavior).* This process of having data and functions that operate

on the data within an object is called *encapsulation.* Several objects can then be linked together to form a complete program. Examples of object-oriented languages include *Simula,* which was developed in the 1960's. However, *C++*, *Java* and *SmallTalk* are contemporary languages in this range. Although Java is sometimes associated with development of web sites it can be used to create whole application programs that do not need a web browser to run. OOP has contributed greatly to the development of *graphical user interface operating systems* and *application programs.*

I

Web Scripting languages!

Web scripting languages are used to develop or add functionalities on web pages. Web pages are hypertext documents created in a language called Hypertext Markup Language (HTML). The language simply consists of *tags* that are interpreted by the web browser software to display text when the HTML file is opened on the screen by a web browser software. A tag is a special word enclosed between the less than and greater than (<>) symbols and the browser can interpret it as a command. For example, to start a HTML page, one must use the *<HTML>* tag at the very top of the document. Other languages like Extended HTML (XML) have been derived directly from HTML with the only difference being that XML allows the user to define their own *tags* instead of using the standard HTML tags.

Unlike other programming languages, HTML does not have the declaration part and *control structures* (to be covered later in the book). Due of this reason, it is not considered as a true programming language.

Due to its simplicity, HTML has many limitations and cannot be used alone when it comes to developing functional websites. Some special blocks of code known as *scripts* may be inserted in HTML pages using scripting languages like *JavaScript*, *VBScript* and *Hypertext Preprocessor (PHP)* in order to add functionality to the HTML page. A script is a small program fragment, written in a different language other than HTML but inserted into the HTML program.

Most HTML tags have an opening tag and a closing tag. An opening tag is enclosed between < > while a closing one between </ >. Text that is to be displayed on the screen is enclosed between an opening and closing tag. For example, the statement <B> Hello </B> will display the word "Hello" in boldface on the screen. Table 3.1 shows examples of HTML tags and their meanings:

1

Table 3.1

Tag	Meaning
1. <HTML></HTML>	Marks the beginning and end of a HTML document. All other tags and text fall between these two tags.
2. <HEAD> </HEAD>	Marks the header part of the document.
3. <TITLE> </TITLE>	Gives title of the web page. Text between this tags appears in the title bar when the page is browsed.
4. <BODY></BODY>	Marks the body part of the document.
5. <CENTER></CENTER>	Centres text and objects on the web page.

6. <B></B>	Bolds the text on the web page.
7. <1></1>	Italicise the text.
8. <Hl></Hl>	Sets size of text on the web page with H6 displaying the smallest and H1 the largest Size.

Creating a script using Javascript

Javascript is a popular scripting language. Before writing your HTML program with a script inserted, make sure that you have the latest browser software installed on your computer. Older browsers may not have support for Javascript. If you are using Internet Explorer, it should be version 5.0 and above.

Open Notepad and key in the following program. Do not write the numbers to the left of each line of code.

```
1. <HTML>
2. <HEAD>
3. < TITLE> Scripting Example </TITLE>
4. </HEAD>
5. <BODY>
6. <HI> < CENTER. <B> We are the world </B> </CENTER> </HI>
7. <SCRIPT LANGUAGE = 'JavaScript'>
8. Document. Write ('My name is strongman');
9. alert ('congratulations for succeeding to run this script');
10. </SCRIPT>
11. </BODY></HTML>
```

After typing the entire program, save your file on the desktop as Example.html and then close the notepad. Notice that the icon to your file on the desktop look like that of the default web browser in your computer.

To view the web page, double click the icon of the file Example .htm1 on the desktop. Figure 3.3. shows an open webpage.

Fig. 3.3: A web page with a script.

Explanations

Line I: The tag <HTML> marks the beginning of the HTML document.

Line 2: The <TITLE> </TITLE> tags show the title of the web page. The text between this tags will appear in the title bar of the running HTML document as can be seen in Figure 3.2. Notice that the title is written in the header section i.e. between <HEAD> and </HEAD> tags.

Line 5: It marks the beginning of the body section. Anything between <BODY> and </BODY> will be executed and displayed when the webpage starts running.

Line 6: This line will display the text "We are the world" on the screen. The text will be large i.e. size HI and it will be centred on the screen. The text will also be bolded.

Line 7: It marks the start point of the script. Notice the line LANGUAGE = 'JavaScript' which tells the browser that the script will be written in JavaScript language.
Line 8: The statement Document. Write tells the browser using JavaScript to write whatever is in the brackets. Notice that in JavaScript, the end of a statement is marked by a semicolon (;).
Line 9: The word *alert* displays a message box on the screen with an OK button. Clicking the button makes the message box to disappear. The text in the brackets appears in the dialog box.
Line 10: Closes the script.
Line 11: Marks the end of the body and the HTML code.

Practical activity 3.1

1. Open a text editor program on your computer like NotePad or WordPad.I
2. Type the following program exactly the way it is in the editor:

```
    <HTML>
    <HEAD><TITLE> This is my first webpage</TITLE></HEAD> <BODY bgcolor = "red" >
    <H l><CENTER><B>Hello World</B></CENTER></HI>
</BODY> </HTML>
```

3. Save your work as webpage.html on the desktop. Make sure that the *Save As Type* box reads "All Files" before clicking the save button in order to avoid saving a text file with two extensions i.e. webpage.html. txt
4. Close your text editor. Notice that your file on the desktop has the icon of the default web browser installed on your computer. Double click it to view the web page! Figure 3.4 shows a sample of the display expected when the file is loaded to the browser. If you have a colour monitor, it should look as below only that it will have black bold text on a red background!
5. Check your program and change your background to blue, save then click the refresh button. What happens?

Fig. 3.4: Sample web page

Advantages and disadvantages of low-level and high-level languages

Having looked at the various programming languages, it is important to identify the advantages and disadvantages associated with each level of programming languages.

Advantages and disadvantages of low level languages

Advantages

1. The CPU understands machine language directly without translation.
2. The processor executes them faster because complex instructions are already broken down into smaller simpler ones.
3. Low level languages are stable and hardly crash or break down once written.

Disadvantages

1. Low level languages are difficult and cumbersome to use and learn.
2. They require highly trained experts both to develop and maintain programs.
3. Removing errors (debugging) in low level language programs is difficult.
4. Low level programs are machine dependent i.e. they are not transferable from one hardware or software platform to another.

Hence we say they are *not portable.*

Advantages and disadvantages of high level languages

Advantages

1. High level languages are portable i.e. they are transferable from one computer to another.
2. High level languages are user friendly and easy to use and learn.
3. High level languages are more flexible; hence they enhance the creativity of the programmer and increase productivity in the workplace.
4. High level languages are far much easier to correct errors (debug).

Disadvantages

1. Their nature encourages use of many instructions in a word or statement hence the complexity of these instructions causes slower program processing.
2. They have to be interpreted or compiled to machine readable form before the computer can execute them.

(Review questions 3.1)

1. Define the term computer program.
2. What is programming?
3. State three advantages of high level languages over low level, languages.
4. List four examples of high level languages and for each state its most appropriate application area.
5. Why is an executable file unique as compared to any other file?
6. Differentiate between a compiler and an interpreter. Why did early computers work well with interpreters?
7. List the various examples of programming languages per generation.
8. State one advantage of machine language over all the other languages.
9. Write the following in full:
   (a) HTML (b) OOP
10. Distinguish between source program and object code in programming.
11. Define the term encapsulation as used in object oriented programming.

Program development

The process of program development is not an easy task. Remember that in our definition of programming, we said that the program must solve a particular problem or accomplish a task. Hence, before developing a program, the requirements of the eventual users and its expected functions should be fully understood.

Program development can be broken into the following stages:

1. Problem recognition.
2. Problem definition.
3. Program design.
4. Program coding.
5. Program testing and debugging.
6. Implementation and maintenance.

The completion of one stage leads to the beginning of the next. At the same time, some stages have to be continuously reviewed in light of the step just before them to make sure that they meet the requirements. For example, after coding, the code has to be compared to the design to see whether it meets the design specification. Therefore, this method becomes very expensive if the correct requirements are not identified at each stage before the next is initiated.

It is important to note that although some people consider documentation as an independent stage, it is done at all stages of the program development lifecycle as shown in Figure 3.5. This is important so that the facts are recorded when they are still fresh and in the required detail by the programmer.

Problem recognition

Problem recognition refers to the understanding and interpretation of a particular problem. In order to understand a problem you need to look for the key words such as *compute, evaluate, compare* etc. You can then rewrite the problem in a more simplified way using the keywords.

A programmer identifies problems in the environment and seeks to solve them by writing a computer program that would provide the solution. Many of the privileges of automation that we enjoy today are as a result of people who wrote computer programs to do the tasks. For example, the intelligent control of traffic lights, the autopilot in aircraft and the use of robots in industry are good examples where problems were recognised and the computer was used as a tool to solve them. Think of the person who after seeing the tedious nature of typing using a manual typewriter decided to develop a word processor program!

In any given circumstance, the following three situations can cause the programmer to identify a problem that is worth solving:

1. *Problems* or undesirable situations that prevent an individual or organisations from achieving their purpose.

2. *Opportunity* to improve the current program. It can be argued that any unexploited opportunity is a problem.

3. *A new directive* given by the management requiring a change in the status quo.

Sample problem

Consider a mathematical problem such as calculating the area of a circle. In this case, the problem is finding the area of a circle. As a programmer, it will be your interest to develop a program that can be used to calculate the area of any circle. The equation for calculating the area of a circle is given by $A = лr^2$.

Problem definition

In problem definition, also referred to as problem analysis, the programmer tries to determine or define the likely input, processing activities and the expected output using the keywords outlined at the problem recognition stage.

At the end of this stage, the boundaries of the expected program will have been established i.e. a clear view of what the program needs to accomplish must be in place. In case, several methods are identified that can be used to solve the same problem, then the best alternative should be chosen.

In our problem of calculating the area of a circle, an investigation reveals that the parameters needed to determine the area of any circle are:

1. Input: (a) Pie (л) which is a constant.
   (b) The radius of the circle.
2. Process: The formula of calculating area of a circle which is л x radius x radius.
3. Output: The area of the circle (A).

The problem definition stage ends with the writing of a requirements report or document for the new program. It is this document that enables a programmer to come up with a program design that meets the needs at hand.

Program design

Program design is the actual development of the program's processing or problem solving logic called the *algorithm.* An algorithm refers to a limited number of logical steps that a program follows in order to solve a problem. It is the programmer who will usually come up with the algorithm after carefully analysing the requirements specification. Many programs are not usually made up of one large block of code i.e. they are not monolithic. Instead, several units called modules work together to form the whole

In modular programming, each module performs a specific task. This approach makes a program flexible, easier to read and carry out error correction.

The design phase enables the programmer to come up with models of the expected program. The models show the flow of events and data throughout the entire program from the time data is input to the time the program gives out expected information. The development of algorithms is covered later in the chapter.

Program coding

Program coding is the actual process of converting a design model into its equivalent program. This is done by creating the program using a particular programming language. The end result of this stage is a source program that can be translated into machine readable form for the computer to execute and solve the target problem. Programs can be written in many different languages but the general trend in the world today is to use programs that are easy to learn and understand such as, Pascal, C++, Visual Basic and Java. Below is a comparison of the same program written in Pascal and C++ used to calculate the area of a circle.

Program in Pascal	Program in c++

Program AreaCircle (input, output);
Const Pi = 3.142;
Var
Radius, Area: real;
Begin
Writeln ('Enter the radius');
Readln (radius);
Area: = Pi *Radius *Radius;
Writeln ('The area is', Area)
End.
	#include<iostream.h>
	main ( )
	{
	double radius, area;
	const double pi = 3. 142;
	cout«"Enter the radius"« "\n";
	cin»radius;
	area = pi *radius * radius;
	cout«"The area is"« area «"\n";
	return 0;
	}

NB: Use lowercase when coding using C++.

Going through the two programs, you will realise that they are fairly similar, irregardless of the language that was used to code them. The table below explains the program codes.

Pascal code	C++ code	Explanation

Program AreaCircle (input, output);	#include<iostream.h»	The header of the programs. The statements in ( ) and < > shows that the user inputs data via the keyboard and the program display information on the screen.
Const Pi = 3.142;	double pi = 3.142;	A constant has been declared with a name pi and value 3.142.
Var radius, area:real	double area, radius;	Real variables with fractional parts have been declared.
Begin	{	Marks the beginning of the program body or c++ function
Writeln ('Enter the radius');	cout< <"Enter radius";	Displays on the screen the string between inverted commas
Readln (Radius)	cin> >radius;	Displays a blinking cursor that tells the user that an input is needed before the program can continue.
Area: = Pi*Radius *Radius;	area = pi*radius * radius;	Calculates the area. Notice the assigment statement in Pascal is : = while in C++ it IS =
Writeln ('The area is' Area)' , ,	cout< <"The area is" area « "\n";	Display the value stored in the variable Area.

End.	return 0;
	}
		Marks the end of the program

NB: “\n" is a C++ syntax of directing the character to a new line. It is equivalent to PASCAL’s "In" used in writeln and readln.

Program testing and debugging

After coding, the program has to be tested and the errors detected corrected (debugged).

There are two types of errors (bugs) that can be encountered when testing a program. These are:

1 *Syntax errors:* These errors emanate from improper use of language rules e.g. grammar mistakes, punctuation; improper naming of variables and misspelling of user defined and *reserved* words. Reserved words are those words that have a special meaning to the programming language and should not be used by the programmer for anything else. These errors are detectable by the translator and must be corrected before the program runs.

2. *Logical errors:* They are not detectable by the translator. The program runs but gives wrong output or halts during execution. Such errors that occur during program execution are sometimes called *runtime* or *execution* errors.

Methods of error detection

There are several methods of testing the program for errors. These include:

1. *Desk checking (Dry-run)*
   It involves going through the program while still on paper before entering it in the program editor. This helps the programmer to detect the most obvious syntax and logical errors.
2. *Using debugging utilities*
   After entering the program in the program editor, you can run the debugging utilities during translation to detect syntax errors in order to correct them before execution.
3. *Using test data*
   The programmer carries out trial runs of the new program. At each run, the programmer enters various data variations and extremes including data with errors to test whether the system will grind to a halt. For example, if the input required is of numeric type, the programmer may enter alphabetic characters to see whether the program will grind to a halt unexpectedly. A good program should not crash due to incorrect data entry but should inform the user about the anomaly and request for the entry of the correct data.

Implementation and maintenance

Implementation

Implementation refers to the actual delivery and installation of the new program ready for use. New programs will obviously change the way things are done when implemented hence the need for review and maintenance.

Review and maintenance

Review and maintenance is important because of the errors that may be encountered after the program has been implemented or exposed to extensive use. A program may also fail not because of poor development but also due to poor use. Therefore proper training and post implementation support of users will always reduce the chances of having them entering invalid data that can crash the program.

Program documentation

Program documentation is the writing of support materials explaining how the program can be used by users, installed by operators or modified by other programmers. All stages of program development should be documented in order to help during future modification of the program. Documentation can either be *internal* or *external.* Internal documentation, is the written non-excutable lines (comments) in the source program that help other programmers to understand the code statements. External documentation refers to reference materials such as user manuals printed as booklets. User manuals are common examples of external documentation There are three target groups for any type of documentation:

1. *User oriented documentation.* These type enables the user to learn how to use the program as quickly as possible arid with little help from the program developer.
2. *Operator oriented documentation.* It is meant for computer operators such as the technical staff. It helps them to install and maintain the program.
3. *Programmer oriented documentation.* It is a detailed documentation written for skilled programmers. This documentation provides necessary technical information to help in

future modification of the program. In this type of documentation, all stages of the program development should be documented because.

(a) There may come a need to revise or modify the program.

(b) Other programmers may think of ways of improving your program.

Development of algorithms

As defined earlier an algorithm can be defined as a limited number of logical steps that a program follows in order to solve a problem. In most cases, algorithms can be depicted using a number of tools such as decision tables, decision tree's, flowcharts and pseudocodes. In this book, we shall mainly discuss program flowcharts and pseudocodes because they are the most widely used by programmers.

Pseudocode

As mentioned earlier, a pseudocode is a set of statements written in a readable language (usually English-like phrases) but expressing the processing logic of a program. Some of the words used in a pseudocode may be drawn directly from a programming language and then mixed with English to form structured statements that are easily understood by non-programmers and also make a lot of sense to programmers. However pseudocodes are not executable by a computer.

Guidelines for designing a good pseudocode

1. The statements must be short, clear and readable
2. The statements must not have more than one meaning i.e. should be unambiguous
3. The pseudocode lines should be clearly outlined and indented clearly.
4. A pseudocode should show clearly the start and stop of executable statements and the control structures (to be discussed later in the section).
5. The input, output and processing statements should be clearly stated, using keywords such as PRINT, READ, INPUT etc. Below are some examples that demonstrate how to write a pseudocode.

Example 3.1

Write a pseudocode that can be used to prompt the user to enter two numbers, calculate the sum and average of the two numbers and then display the output on the screen.

Solution

```
START
    PRINT "Enter two numbers"
    INPUT X, Y
    SUM = X + Y
    AVERAGE = SUM/2
    PRINT SUM
    PRINT AVERAGE
STOP
```

Example 3.2

Write a structured algorithm that would prompt the user to enter the length and width of a rectangle, calculate the area and perimeter then display the result.

Solution

(i) First draw the rectangle of length (L) and width (W)

(ii) Write down the pseudocode

```
    START
        PRINT "Enter length and width":
```

```
        READ L, W
        AREA = L * W
        PERIMETER = 2(L + W)
        PRINT AREA
        PRINT PERIMETER
    STOP
```

Example 3.3

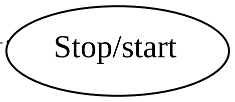
Write a pseudocode for a program that can be used to classify people according to age. If a person is more than 20 years; output "Adult" else output "Young person"

Solution

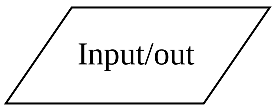
```
START
    PRINT "Enter the age" INPUT AGE
        IF AGE> 20 THEN
        PRINT "Adult"
    ELSE
        PRINT "Young person"
```

Program flowcharts

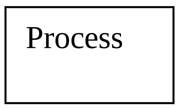
Unlike a pseudocode which expresses ideas in form of short statements, a flowchart does the same using both statements and special symbols that have specific meaning. Therefore, a flowchart is a diagrammatic representation of a program's algorithm. The symbols are combined with short text clues which are a form of shorthand understood by programmers. The special symbols used to draw program flowcharts vary but the most common ones are as outlined below:

1. 

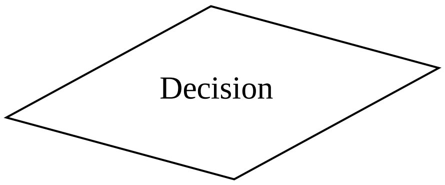
Ellipse: denotes the beginning and end of the program algorithm.



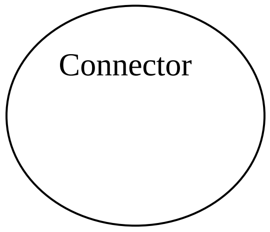
2. *Parallelogram:* used to denote an input or output operation. For example, READ A, B, PRINT SUM.



3. Rectangle: Indicates that a processing or data transformation is taking place. For example SUM=A+B.



4. Rhombus: Used to specify a condition. A condition must evaluate to a *boolean value* (True or false) for the program to execute the next instructions.



5. Connector: Used as a connecting point or interface for arrows coming from different directions.

Arrow: Used to indicate the direction of flow of the program logic.

Guidelines for drawing a flowchart
1. There should be only one entry/starting point and one exit point of the program algorithm.
2. Use the correct symbol at each stage in the flowchart. For example, it is wrong to use a decision symbol where input is required.
3. The logical flow should be clearly shown using arrows.

Comparison between a pseudocode and a flowchart
Taking our example of calculating the area of a circle mentioned in the earlier subtopic the algorithm by both methods are shown below.

Program control structures
Program control structures are blocks of statements that determine how statements are to be executed. In structured programming languages, there are three control structures namely; *sequence, selection* and *iteration* (looping).
Sequence
In sequence control structure, the computer reads instructions from a program file starting from the first top line and proceeding downwards one-by-one to the end. This is called *sequential program execution.*
Therefore, sequential program execution enables the computer to perform tasks that are arranged consecutively one after another in the code. However, most programs that solve real world problems need to enable the computer either to repeat tasks or to make decisions when certain conditions are true or false hence the need for selection and iteration.
Selection
In selection control, execution of statements depends on a condition that returns true or false. The condition must be a boolean expression. One example of a boolean expression is x > = 20. In such a case the condition is true if x is equal to or greater than 20. Any other value that is less than 20 is therefore false.
Generally, the four types of selection controls used in most high-level programming languages are:

1. IF... THEN.
2. IF ... THEN ... ELSE.
3. Nested IF.
4 CASE selection.

In this book, we shall demonstrate how to implement these controls by using both pseudocodes and flowcharts.

IF ... THEN

IF ... THEN selection is used if only one option is available. In this case, all other options are ignored. For example, in a school environment, the administration may decide to reward only those students who attain a mean mark of 80% and above. Therefore, if a students attains 80% and above, he or she is rewarded while the rest are ignored. The following pseudocode and flowchart illustrate this condition:

IF ... THEN... ELSE

IF ... THEN... ELSE selection is suitable when there are two available options. For example, in a football match, if a player does a mistake which is considered serious by the rules of the game, he/she is given a red card. Otherwise, he/she is given a yellow card. The algorithms below illustrates this situation using the IF ... THEN... ELSE selection.

Nested **IF** selection

Nested IF selection is used where two or more options have to be considered to make a selection. For example, in an Olympics track event, medals are awarded only to the first three athletes as follows:

(a) Position 1.Gold medal
(b) Position 2: Silver medal
(c) Position 3: Bronze medal

The pseudocode segment and flowchart extract below shows the structure *of* the Nested IF selection.

Pseudocode segment

```
IF position = 1 THEN
    medal = "Gold"
ELSE
    IF position = 2 THEN
    medal = "silver"
ELSE
    IF position = 3 THEN
    medal = "bronze'
ELSE
medal = "nil"
        END IF
    ENDIF
ENDIF
```

Flowchart extract

The general format of the Nested IF is

```
IF < condition> THEN
    statements
ELSE
    IF < condition> THEN
statements
ELSE
IF < condition> THEN
statements
ELSE
statements
            END IF
    END IF
ENDIF
```

CASE Selection

CASE selection is an alternative to the Nested **IF** especially where there are several options to choose from. This selection is preferred to the Nested **IF** in order to reduce the many lines of code. However, it is important to note that the boolean expression for the case selection can only be expressed using integers and alphabetic characters only. Generally the boolean expression should be CASE *integer* OF or CASE *Char* OF as illustrated in the example below. In this case, average must be an integer.

Pseudocode

```
CASE average OF
      80..]00: Grade = 'A'
      70.. 79: Grade = 'B'
      60.. 69: Grade = 'C'
      50.. 59: Grade = 'D'
      40 .. 49: Grade = 'E'
ELSE
       Grade = 'F'
ENDCASE
```

Flowchart extract

General format of case

```
        CASE x OF
                Label: statement
                Label 2: statement 2
                Label 3: statement 3
                Label n: statement n -1
ELSE
statementn
ENDCASE
```

Note that the flowchart is not different from that of the Nested IF construct-

Iteration (looping)

Iteration, also referred to a looping or repetition is designed to execute the same block of code again and again until a certain condition is fulfilled. Iteration is important in situations where the same operation has to be carried out on a set of data many times. For example, assume that you

are writing a program that will use the same formula to calculate the average marks of three subjects for each student in a class and that the student records are stored in a computer file.
To calculate the average score for each student in a class, the program should repeatedly read record by record from the file, and then use the formula after each read operation to calculate the average.
Generally, the three main looping controls are:
(a) The WHILE loop
(b) The REPEAT. .. UNTIL loop
(c) The FOR loop

The WHILE loop

The 'WHILE' loop is used if a condition has to be met before the statements within the loop are executed. Therefore, this type of loop allows the statements to be executed *zero* or *many* times. For example in banking, to withdraw money using an automated teller machine (ATM) a customer must have a balance in his/her account.
This scenario can be represented as follows using an algorithm
Pseudocode segment

```
 WHILE balance> 0 Do
      Withdraw cash
Flowchart extract
      Update account
      END WHILE
   Withdraw cash Update_account
   In our case above,
```

1. The condition *balance> 0* is first tested.
2. If it is true, the account holder is allowed to withdraw cash.
3. The program exits the loop once the balance falls to zero.

In general, the WHILE loop can be represented using a pseudocode and a flowchart extract as shown below.
Pseudocode segment

```
 WHILE < condition> DO
statements
ENDWHILE
```

The REPEAT ... UNTIL loop

Unlike the 'WHILE' loop, REPEAT... UNTIL allows the statements within it to be executed at least once since the condition is tested at the end of the loop. For example, consider the ATM cash withdrawal case discussed under the 'WHILE' loop. If the REPEAT ... UNTIL is used, then the client will be able to withdraw the cash at least once since availability of balance is tested at the end of the loop as shown below.
Pseudocode

```
 REPEAT
      Withdraw cash
      Update account
UNTIL balance >_0;
```

The general format of the REPEAT. .. UNTIL loop is;
Pseudocode

```
 REPEAT
```

```
statements
UNTIL < condition>
```

The FOR loop

The FOR loop is used in circumstances where execution of the choosen statements has to be repeated a predetermined number of times. For I
example, consider a program that can be used to calculate the sum of ten numbers provided by the user. The 'FOR' loop can be used to prompt the user to enter the 10 numbers at most ten times. Once the numbers have been entered, the program calculates and displays the accumulated sum. The loop is predetermined because it has to be repeated 10 times as shown by the algorithms below.

Pseudocode

```
FOR count = 1 to 10 DO
     PRINT "Enter a number (N)"
      INPUT N
Sum=Sum+N
END FOR
Display SUM
```

Explanation

The FOR loop in the problem above functions as follows:

1. The loop variable (count) is first initialised to the lower limit, in this case a value of 1.
2. The lower limit is then tested against the upper limit whose value is set at 10.
3. If the lower limit is less than or equal to 10, the program prompts the user to enter a number N, otherwise the computer exits the loop.
4. After the last statement in the loop has been executed, the loop variable count is incremented by a 1 and stored in the lower limit i.e. *lower limit* = *count* + 1.
5. The lower limit is then stored in *count* and step (2) is repeated

The FOR loop can also be used to count downwards from the upper limit to the lower limit. For example, in the above problem, the upper limit 10 can be tested against the lower limit 1 as follows.

```
FOR count = 10 DOWN TO 1 DO
```

Therefore the general format of the FOR loop can be represented using two sets of algorithms.

1. Pseudocode for 'FOR' loop that counts from the lower limit

```
     FOR loop variable = lower limit To upper limit DO.
     statements
ENDFOR
```

2. Pseudocode for a 'FOR' loop that counts from the upper limit down to the lower limit

```
     FOR loop variable = Upper limit DOWN TO Lower limit DO
statements;
 ENDFOR.
```

Designing more complex algorithms

The following examples demonstrates how to design more complex algorithms based on what has been covered previously. This is aimed at helping a beginner programmer understand how control structures are used to design logically correct algorithms.

Example 3.7

With aid of a pseudocode and a flowchart, design an algorithm that:

(a) Prompt the user to enter two numbers X and Y.

(b) Divide X by Y. However, if the value of Y is 0, the program should display an error message "Error: Division by zero".

Solution

Using a pseudocode

```
START
PRINT "Enter 2 numbers X and Y"
INPUT X, Y
IF Y = 0 THEN
    PRINT "Error: Division by zero"
    ELSE
        Quotient = X/Y
        PRINT X, Y, Quotient
    ENDIF
STOP
```

Example 3.8

Tusaidiane Savings Society (TSS) pays 5% interest on shares exceeding 100 000 shillings and 3% on shares that do not meet this target. However no interest is paid on deposits in the member's TSS bank account. Design an algorithm for a program that would:

(a) Prompt the user for shares and deposit of a particular member.

(b) Calculate the interest and total savings.

(c) Display the interest and total savings on the screen for a particular member of the society.

Solution

Using a pseudocode

```
START
    PRINT "Enter member name, share and deposit"
    INPUT Name, Shares, Deposit
    If shares> 100 000 THEN
        Interest = 0.05 x shares
    ELSE
        Interest = 0.03 x shares
    ENDIF


    Total savings = Deposit + shares + Interest PRINT Name, Total Saving, Interest
STOP
```

Example 3.9

In an athletics competition, an athlete is rewarded as follows:

1 st position: Gold

2nd position: Silver

3rd position: Bronze

Draw a flowchart for a program that would be used to determine the type of medal to be rewarded to each athlete.

Example 3.10

The class teacher of Form 3W in a secondary school requested a programmer to design for her a simple program that would help her do the following:

(a) Enter the names of students and marks obtained in 8 subjects - Mathematics, English, Kiswahili, Biology, Chemistry, Business studies, Computer studies and History.

(b) After entering each subject mark, the program should calculate the total and average marks for each student.

(c) Depending on the average mark obtained, the program should assign grade as follows:
- (i) Between 80 and 1 00 - A
- (ii) Between 70 and 79 - B
- (iii) Between 60 and 69 - C
- (iv) Between 50 and 59 - D
- (v) Below 50 - E

(d) The program should then display each student's name, total marks and the average grade. Using both a flowchart and a pseudocode, write an algorithm that shows the design of the program.

Solution

Using pseudocode

```
START
    REPEAT
PRINT "Enter name and subject marks"
INPUT Name, Mathematics, English, Kiswahili, Biology, Chemistry, Business, Computer,
History
SUM = Mathematics + English + Kiswahili + Biology + Chemistry +Business + Computer +
History
AVG = SUM/8
IF (AVG ≤80) AND (AVG≤100) THEN
  Grade = 'A'
ELSE
  IF (AVG ≥70) AND (AVG≥ 79) THEN
  Grade = 'B'
ELSE
      IF (AVG ≥60) AND (AVG≥69) THEN
      Grade = 'c'
ELSE
     IF (AVG ≥50) AND (AVG ≥59) THEN
     Grade = 'D'
     ELSE
              Grade = 'E'
             ENDIF
        ENDIF
    ENDIF
ENDIF
    PRINT Name, Sum, AVG, Grade UNTIL Count = Number of students
STOP.
```

Example 3.11
The gross salary of employees in KARU BOOKS ENTERPRISE is based on basic salary and additional benefits as follows:

(a) Employees who have worked for the company for more than 10 years receive an additional pay of 10% to their basic salary.
(b) Monthly salary bonus based on monthly sales of books as follows:

Monthly sales	Bonus Rate (%)
Above 500 000	15
Between 250 000 and 500 000	10
Below 250 000	5

Draw a flowchart for a program that would be used to calculate the gross salary then output each employee’s basic salary, gross salary and all benefits.

SYSTEM DEVELOPMENT

Chapter outline

Introduction
Description of a system
Information system
Theories of system development
Stages of system development
System documentation

Introduction

The concept of a system emerged from early *psychologists* who believed that the mind was a whole unit, rather than a collection of psychological parts as the belief was by that time. However, it was Ludwig von Bertalanff, a German biologist, who gave the name *"general systems theory"* to the discipline that devoted itself to coming up with principles that apply to all systems.

A system is a set of organised components which interact in a given environment and within a specified boundary to achieve collective goals and objectives that are *emergent*. Emergent characteristics are those that result from interaction of various components and may not exist in the individual component. Therefore, once the components come together, they become interrelated and generate new goals and objectives. For example, a bicycle system has all the components working together to provide motion when ridden. The individual components cannot provide these services to a rider when on their own!

Description of a system

A system can be described as being either soft or hard.

Soft systems

Human activity systems are said to be soft systems. They are described as soft because of three main reasons:

1. Their boundaries may be fluid or keep on changing.
2. Their goals and objectives usually conflict and may not be captured clearly at anyone time because they are based on human factors like attitudes and preferences.
3. It is difficult to precisely define exact measures of performance for them.

One example of a soft system is the political system. It is very difficult for instance to model a system that will predict the political mood in a country over a period of time. Another example is a sales tracking and prediction system in an organisation. Sales in an organisation depend on human factors like attitude in the market place.

Hard systems

Hard systems are systems whose goals and objectives are clearly defined and the outcomes from the systems processes are predictable and can be modeled accurately. Such systems are based on proven scientific laws like mathematical formulas or engineering solutions.

An example of a hard system would be a stock management system in a supermarket. It is possible to know exactly the stock levels, cost and sale price and to predict accurately the profit if all the stock is sold.

A good system incorporates both hard and soft aspects of a system. For example, a stock management system should be able to show when the demand for a certain item rises so that a decision can be made to stock more. New demand is driven by soft aspects in people's lives like attitude and seasons!

Characteristics of systems
All systems have some common characteristics. Some of these characteristics are explained below.
Holistic thinking
In holistic thinking a system is considered as a whole. Aristotle, a Greek philosopher, once said that *the whole is more than the sum of the parts.* The various components that make up a system may be simple in nature and process but their combination creates a *complex whole,* whose overall goals are more sophisticated than those of the individual components. Hence, a system should be considered as a *whole unit* rather than considering its parts individually.

Subsystems
A system is made up of different components (subsystems). Therefore a system does not exist in solitude but it may be a component of larger a system. For example, the classroom system is part of a school system, which is part of the Ministry of Education. The Ministry of education is part of the Government which is part of the global system!
Boundary and environment
Each system has a space (boundary) within which the components operate. Any entity that falls outside the boundary but interacts with the system is part of the system environment. Such entities are called *external entities.* They provide the inputs and receive the outputs from the system. For example, the external entities to a school system may include the parents, various suppliers and the society at large.
Purpose
The purpose of each system is to perform a particular task or achieve a goal. The objectives that a system is supposed to achieve enable system developers to measure the performance of a system during its operation. One main objective for a school system for instance is to enable the students to excel in national examinations.
Process
A system usually will *transform* or process data from one state to another.
System entropy
The word entropy means decay. Systems "decay" naturally over time. This means that a system slowly becomes useless to the user either due to improvement in technology, new management policies or change in user requirements. Therefore a system must be reviewed in order to improve it or to develop a new one. .
Inputs and outputs
A system communicates with its environment by receiving inputs and giving outputs. For example, a manufacturing firm can be considered as a system that gets raw materials (inputs) from the environment and transforms them into finished products (outputs) released into the environment.
Open and closed systems
A system can be described as being *open* or *closed.* An open system receives input from and gives output to the environment while a closed system does not. Open systems normally adapt to changes in the environment.
Control
Control can be defined as the method by which a system adapts to changes in the environment in order to give the expected output or to perform to the expected level. Control is achieved through *feedback* which involves having outputs from the process of the system being fed back to the

control mechanism. The control mechanism in turn adjusts control signals that are fed to the process which in turn makes sure that the output meets the set expectations. Fig. 4.1 depicts a typical system that has feedback to the control function. Imagine a motor vehicle manufacturing company that is producing several vehicles per day. Assuming that the demand rises, then feedback would show that the company is underperforming. Hence, control signals that would speed up movement of units on the assembly line can be issued to increase production.

Information system

An information system is an arrangement of people, data processes and information that work together to support and improve the day-to-day operations in a business and the decision making process. The main purposes of an information system in an organisation are:

1. Supporting information processing by enhancing tasks such as data collection, processing and communication.
2. Helping in decision making by collecting operational data, analyzing it and generating reports that can be used to support the decision making process. This process is referred to as *on-line analytical processing.*

3. Enable sharing of information. Perhaps, this is one of the greatest powers of information systems. For example, any departments in a given organisation can now share the same electronic information stored in a central database at the click of a mouse button.

Why develop new information systems?

The need for developing information systems is brought about by three circumstances:

1. *New opportunities:* A chance to improve quality of internal processes and service delivery in the organisation.
2. *Problems:* These are undesirable circumstances that prevent the organisation from meeting its goals.
3. *Directives:* These are new requirements imposed by the government, management or external influences.

Role of information system analyst

A system analyst is a person who is responsible for identifying an organisation's needs and problems then designs and develops an information system to solve them. The system analyst does this by:

1. Reviewing the existing system and making recommendations on how to improve or implement an alternative system.
2. Working hand in hand with programmers to construct a computerized system.
3. Coordinating training of the new system users and owners.

Project management

The system analyst is the overall project manager of the information system being implemented. His project management skills like assuring quality, keeping within schedule and budget determine whether the system will be successfully implemented or not. For example, a project that does not stick to its schedule will most likely overshoot its budgeted cost leading to unsuccessful completion.

Theories of system development

Several theories or methods are used in system development. The aim of all these theories and methods is to identify business requirements and to develop information systems that effectively meet them. This helps to support the day to day operations and decision making processes in an organisation.

Some of the most common system development theories include:

1. Traditional approach.
2. Rapid application development (RAD).
3. The structured approach.

At this level, we will concern ourselves mostly with the structured approach. However, we shall briefly discuss the other two methods of system development.

Traditional approach

Traditional approach relies mostly on the skills and experience of individual staff members carrying out the project. This means that there is no formal documented methodology to be followed by all system developers in the organisation. This obviously presents a chaotic scene in system development especially where more than one persons are involved in the development effort. In most cases, success depends on the heroic efforts of an individual. This means that all other projects heavily rely on a particular person for their success.

In this approach, the manual system is replaced with a computerised one without change in overall structure of the former system. Hence the weaknesses of the former system are not addressed and are carried forward to the new system. For example, in a banking hall, a manual system is characterised by long queues and poor controls. If the traditional approach is followed, each cashier will simply be given a computer. The long queues might remain and lack of controls increase because no value was added to the former information system. This method is not recommended for today's business environment.

Rapid application development (RAD)

Rapid application development (RAD) model evolved from the theory that businesses today heavily rely on information technology. Many information' systems that were manual in nature are now fully computerised. Therefore, development and implementation of information systems needs to be quick enough for the organisation to maintain a competitive advantage in the market place.

Recent developments in programming software have seen the release of fourth generation languages (4GL's) which are user-friendly because of their graphical interfaces. Rapid application development makes it possible for system developers to quickly capture user requirements by designing system interfaces in the presence of the user. This rapid application development technique is known as *prototyping,* and assumes that the user knows what they want when they see it. A prototype is a smaller working model of a real world system. Other approaches used in rapid application development include *small team with advanced tools* (SWAT) and *joint application development* (JAD).

The main disadvantage of rapid application development is that the working system may have oversights and weaknesses due to the quick Development. For example, a system may be working well but lack the necessary inbuilt security mechanisms. This would be undesirable in today’s insecure operating environment.

The structured approach

Structured approach to system development defines a set of stages that should be followed when developing a system. Each stage is well documented and specifies the activities to be carried out by the system analyst and his team while developing a system.

Stages of system development

The main stages in system 4evelopment as depicted by the structured approach include:
1. Problem recognition and definition.
2. Information gathering.
3. Requirements specification.
4. System design.
S. System construction (coding).
6. System implementation.
7. System review and maintenance.

Figure 4.2 is a diagrammatic representation of the seven stages of the system development lifecycle (SDLC).

The stages of developing a system are also called the *system development lifecycle.* Each stage serves a role in the problem solving process. The lifecycle divides the life of an information system into two major parts namely:
1. The development stage.
2. The operation and support stage.

To demonstrate how to undertake each stage, we shall consider a case study.

Case study

Computer-based library management system

Mutito high school library has 3000 text books. Each book is identified by its author, ISBN number, book ID and title. The books are arranged on the shelves using their book ID. Card catalogues are maintained for all the books. There are two types of catalogues, one arranged according to the author's names while the other is arranged according to the titles of the books. Each member is issued with three borrower cards that have a registration number and name of the member. To locate a book for borrowing, a member checks in the card catalogue for its classification then moves to the shelve to retrieve it. The member surrenders a borrower's card .at the issue counter where the staff gives out the book and stamps the date of return. A member is not allowed to borrow more than three books at anyone time. Members are charged for overdue books at a fixed rate multiplied by the number of days delayed.

We now look at each of the stages of system development in more detail with this case study in mind.

Problem recognition and definition

Problem recognition is done during the *preliminary investigation.* During the recognition phase, the system analyst seeks to answer two questions. The first is whether the proposed project is worth looking at while the second is if the project is worth pursuing. After this, the system analyst has to define the scope of the project and establish the constraints, budget and schedule. The most common constraints are usually lack of finance, lack of enough expertise and/or lack of appropriate technology to develop the system.

Problem definition, also called problem analysis is the process of identifying the problem, understanding the problem and finding out any constraints that may limit the solution. This stage requires the analyst to find out as much as possible about the current system in order to draw up a good and relevant proposal for the new system. Remember that there is always an existing system whether manual or computerised. After this, several alternative solutions are modeled. The main question asked at this point is whether the proposed solution is the right one.

Looking at our case of the school library management system, the problem at hand is to replace the inefficient manual operations such as cataloguing with an efficient computerised system. The system analyst tries to answer the following questions.
1. What are the shortcomings of the current systems?
2. What types of records are used for books and students in the library?
3. What procedure is followed to borrow/lend books?
4. How are overdue books handled when returned?
In this first stage, a special study will be carried out to establish the costs and benefits of a new system. This study is called a *feasibility study*. A new system will only be developed if its benefits are more than its costs. The end of this stage is marked by presentation of a feasibility report to the management.
The feasibility of a system is assessed in four ways:
Operational feasibility: This establishes the extent to which the users are comfortable or happy with the proposed or new system.
Schedule feasibility: This establishes whether the development of the proposed system will be accomplished within the available time.
Technical feasibility: This establishes whether the technology available is sufficient or can be upgraded for the new system. It also seeks to find out whether the staff has relevant technical skills to develop and use the new system.
Economic feasibility: This establishes whether developing the new system is cost effective by analysing all the costs and benefits of the proposed system.

Information gathering
After the feasibility study report has been approved by the management, the system analyst can then proceed to the next stage referred to as *information gathering* or *fact finding*. Some of the methods used to collect or gather data include:
1. Study of available documents.
2. Interviews.
3. Questionnaires.
4. Observation.
5. Automated methods.

Studying available documentation
The available documentation describes the current system and all its procedures. It forms a rich source of information for the analyst. Examples of such documents are card catalogues, receipts, reports, technical manuals, organisational charts and archival or backup files.

Interviews
Interviews should be carried with the relevant stakeholders in order to get views about the current system and gather information about the requirements for the proposed system. The interview method is powerful because it enables the analyst to have face to face contact with the interviewee.
Therefore in executing an interview, the following guidelines should be followed:

1. The interviewee must be informed in good time and the topic of discussion communicated accordingly to allow for adequate preparation.
2. Avoid personal biases in your questions and perspectives.

3. Be careful about body language and *proxemics.* Proxemics refers to things like sitting arrangement, body closeness and how people react when their private distance is violated.
Figure 4.3 shows a verbatim introduction of sample interview with the library manager.

INTERVIEW TITLE

BRIEF INTRODUCTION Interviewer: ....
Interviewee: ... .

Interviewer: Hello............
Interviewee: Hello. Welcome to my office.
Interviewer: Thank you. Please call me Pat. I would like to ask you a few questions about the system that we are developing
Interviewee: ....
....
....

Fig. 4.3: Example of an interview

Advantages of interviews

1. Non-verbal communication like facial expressions can be used and observed.
2. Questions can be rephrased instantly for clarification and to probe the interviewee further.

Disadvantages of interviews

1. It is difficult to organise interviews and they are time consuming.
2. The interviewee may not fully open up on some issues that may be personal or sensitive.

Questionnaires

A questionnaire is a special purpose document that allows a person to collect information and opinions from the people who receive and respond to it. The main advantage of using this method is that the questionnaires give the respondents privacy when filling them and they can do so at their own pleasure. This may enhance the sincerity of the information given.
Figure 4.4 below shows an extract of a questionnaire used to gather data from library attendants.

QUESTIONNAIRE

BRIEF INTRODUCTION Date: ...,.............
.
....

QUESTIONS

1. How long have you worked as a library attendant:
   1 yr. 2yrs. over 2 yrs.
2. How long does it take to rearrange books on the shelves?
   days weeks months

Fig. 4.4: An example of a questionnaire

Advantages of questionnaires

1. Since they are filled and returned in privacy, more sincere responses are possible.
2. The respondent can fill the questionnaire at their own pace.

Disadvantages of questionnaires

1. Good questionnaires are difficult to prepare.

2. The respondent may not fully understand the questions because of ambiguity of language hence give erroneous responses.

Observation

Observation requires the observer to participate or watch closely as a person performs activities in order to learn about the system. This method gives the analyst first hand experience about the problems and exposes him/her to the system requirements. The main advantage of observation is that concepts that are too difficult for non-technical staff to explain can be observed. However, this method has some drawbacks too. These include:

1. The person being observed might alter behaviour leading to wrong equirements being observed.
2. The need to be on-site consumes a lot of time.

Automated methods

Automated data collection is mostly used when actual data is required but difficult to get through interviews, observation or questionnaires. Such data may be collected using devices that automatically capture data from the source such as video cameras, tape recorders etc.

Preparing and presenting the fact finding report

At the end of the information gathering stage, the analyst must come up with a requirements definition report that has the following details:

1. Cover letter addressed to the management and IT task force written, by the person who gathered the facts.
2. Title page which includes the name of the project, name of analyst and the date the proposal is submitted.
3. Table of contents.
4. Executive summary which provides a snapshot of how the new system is to be implemented. It also includes recommendations of the system analyst because some people will only have to read the summary to make decisions.
5. Outlines of the system study which provides information about all the methods used in the study and who and what was studied.
6. Detailed results of the study which provide details of what the system analyst has found out about the system such as problems, constraints and opportunities that call for an alternative.
7. Summary which is a brief statement that mirrors the contents of the report? It also stresses the project's importance.

This report is then presented to the management for evaluation and further guidance.
fig 4.5 shows a sample general outline of the fact finding report presented to the management of the school library and the head of the I

Library management information system

Fact finding report

1.0 Table of contents.
2.0 Executive summary.
2.1 Objectives: The new computerised system is intended to improve efficiency in the library by:
(a) Keeping an inventory of all the books in the library and automatically updating the stock hence eliminating the tedious physical counting process.

(b) Reducing the time needed to seek for a book by 60%.
(c) Tracking overdue and lost books.

2.2 Recommendation:
This system could result in efficient processing of library transactions. It will replace the tedious manual system.

3.0 Methods used to study the system
3.1 Interviews: Used when seeking facts from management.
3.2 Questionnaires: Circulated to user staff.
3.3 Observation: Observed book search and issue.
4.0 Detailed results
4.1 Problems: Duplication of records, delays and book loss.
4.2 Opportunities: Efficiency, stock management etc.
4.3 Alternatives: Enforce controls in current system, more staff etc.
5.0 Summary.
The new system is highly recommended because the other alternative of enforcing controls and employing more staff will add operating costs with little additional value.

Fig. 4.5: A sample outline of a fact finding report

NB: The sample report is simplified for purposes of instruction at this level and should not be taken as a complete report. A complete report lay comprise of several bound pages.

Requirements specification
Requirement specification, the system analyst must come up with the detailed requirements for the new system. Remember that in the long run the hardware and software used to develop the system mainly depends on input, output and file requirements. For example, if one of the input requirements is that the system would require data in picture format then one input device that cannot be avoided is an image capturing device such as a digital camera or a scanner. At this stage the following requirements specifications are considered:

1. Output specification.
2. input specification
3. File/data stores.
4. Hardware and software requirements.

Output specifications
As opposed to data processing cycle where we follow the input-process-output model in system development, consideration is given to the output requirements of the new system first. This is because; the main interest from a sys*tem* is information (output). For example the management of the library in our case study is interested in whether the system can generate reports on overdue books, charges on late return, inventory etc.
The quality of system output depends on how well management and user requirements were identified. The Output is usually in the form of reports either in hardcopy or softcopy form.

The following factors should be put into consideration when designing the output

1. *The target audience.* For example, top management would require a summary of overall performance in the organisation while a user report may show only the transactions carried out or transactions at hand
2. The fr*equency of report generation.* Some reports are required daily, others weekly, monthly or annually. However, some are required in an *ad hoc* manner i.e. at random.
3. *Quality and format:* The quality and format of information to be generated should be put into consideration.

For our case study outlined earlier, the following outputs are needed from the library management system:
1. A report about all the overdue books showing charges against each borrower.
2. A search report for a particular book showing its classification and whether its on the shelve or not.
3. A search report about a particular member showing which books he/she is currently holding.
Table 4.1 below shows a sample report expected to be generated from the computerised library system showing all the overdue books.

Table 4.1

	OVERDUE BOOKS				
ISBN Number	Title	Member Name	Date Issued	Date Due	Overdue Days
1.9966495746	Computer studies	James L. Wak	10/01/2005	24/01/2005	6
2. 9966 49 600 x	Mathematics 1	Linda Were	12/01/2005	26/01/2005	4
3.	....	....	....	....	......
4.	....	....	....	....	......
5.	....	....	....	....	......

Input specifications

Once the system analyst has identified the information (output) requirement of the new computerised system, he/she goes ahead to identify the input needed to obtain the relevant information from the system. In our case of the library, the following inputs can be deduced from the output specification:

1. The type of data needed to add a book to the books file or database in the library. For example in the library database the following data items may be entered:
   (a) Title of the book.
   (b) Names of the author(s) of the books.
   (c) The ISBN number of the book.
   (d) Book ID
2. Determine data that is needed for someone who wishes to borrow a book.

After identifying all the inputs, the analyst designs the user interface by designing data entry forms or screens. An example of an input form is the new member registration form as shown in Figure 4.6.

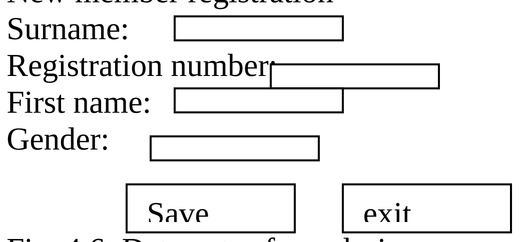

New member registration
Surname:
Registration number:
First name:
Gender:

Fig. 4.6: Data entry form design

The user interface is an important determinant of whether the system will be happily accepted by the users or not. Hence, it must be designed with a lot of care. The following guidelines should be observed:

1. Objects placed on forms like text boxes, labels and command buttons must be neatly aligned and balanced on the form.
2. The size of the form must not be too small for user legibility or too big to fit on the screen
3. The colour for the interface must be chosen carefully to avoid hurting the eye. Avoid colours that are too bright.

File/data stores

File requirements specification involves making an informed decision on files required to store data and information in the system. The system analyst should identify the number of files that will be needed by the system and determine the structure of each of the files. For example, will the files allow direct access? Will the files be sequential files stored on a magnetic tape?

The attributes of the records in a file should also be identified. An attribute is a unique characteristic of a record for which a data value can be stored in the system database. If it is a student, one attribute can be the name and the other is the student's registration number. For a book record, the attributes that can be identified include: Book ID, international serial book number (ISBN), the title, publisher, year of publication, date of issue and date of return.

However, only those attributes that are of importance to the system will be picked and used to store data for each record. In our case study for instance, we only need the Book ID, title, author, ISBN number, date of .issue and return.

These attributes will form the basis for table design in the database. Each attribute will become a field in the table. For example, there will be a *Books table* that will have fields for each record.

Factors to consider when designing a file

In order to design a good file, you need to consider the following aspects:

1. *The key attribute or field:* This is usually an attribute that is unique for each record.
2. *The type of data:* Each field has a data type. Book titles can be stored as data of type "text" while the date of borrowing a book as of the type "date" in the database.
3. *The length of each field:* This is important because the longer the field, the slower the system takes to process transactions. A name field can be specified to be 30 characters long while the integer field can be 10 characters long. However, these vary depending on the system developer's perception of how the system should store the data.
4. *Back up and recovery strategies:* The updated copies of data and information files need to be stored in a different place other than the location of the current system. This makes sure that even if the current file gets corrupted or crashes, the backed up data can be used to recover or reconstruct the original file.

Hardware and software requirements

The system analyst should specify all hardware and software requirements for the new system.
Some of the factors to consider in hardware and software specification are:

1. Economic factors such as price and acquisition method.
2. Operational factors e.g. reliability, upgradeability and compatibility with the existing resources.
3. User-friendliness.

System design

There are several tools for designing an information system. Examples of such tools are *flowcharts, data flow diagrams, entity relationship models and structured charts.* In this book, we shall concentrate on the use of the system flowcharts as the primary tool for system design.

A system flowchart is a tool for analysing processes. It allows one to break process down into individual events or activities and to display these in shorthand form showing the sequential or logical relationships between them.

After drawing the system flowchart, other algorithm design tools like pseudo codes and program flowcharts can be used to extract the processing logic for each module in the system before system construction.

The system flowchart has many similarities to the program flowchart covered earlier in the book. However, it has its own set of symbols and it seeks to depict the whole system rather than the individual program modules. Figure 4.8 shows some common system flowchart symbols.

Other symbols that are of great importance at this level are as follows:

Rectangle with rounded corners: represents an event which occurs automatically and usually triggers a chain of other events. For example, the book lending process is triggered by a student request!

Kite: represents the sort operation.

Designing a system flowchart

Designing system flowcharts gives a concise picture of the way particular processes are done within the business organisation. After this has been achieved, the next logical step of making changes to the processes for the better can be handled easily.

Although there is no formal approach for designing a system flowchart, the following guidelines are important:

1. Start by writing the title of the flowchart. For our case study, the title "Library Books Management Information System" could be sufficient.
2. If possible, start drawing the flowchart with the trigger event. In this case, our trigger would be a student request to borrow a book or to return an overdue book.
3. Note down the successive actions taken in their logical order until the event or process is concluded. Use few words to describe the actions.
4. When there are many alternatives at the decision stage, follow the most important and continue with it. Other significant but less important alternatives can be drawn elsewhere and reference made to them by using the on/or off page connectors. Figure 4.9 shows the system flowchart for the proposed computerised library management system for the school.

Explanation

From the system flowchart, we observe that:

1. A member e.g. a student requests for a particular book.
2. The system checks for the students record in the system. If the student has more than three books, a message to this effect is displayed and cannot borrow an extra book.
3. If the student has less than three books, then the book can be given out to him/her.

From the system flowchart, a program flowchart for a particular task can be extracted. Figure 4.10 illustrates the book lending process extracted from the library management system flowchart.

System construction
System construction refers to the *coding, installation* and *testing* of the modules and their components such as outputs, inputs and files.
The purpose of the construction phase is to develop and test a functional system that fulfils the business and design requirements. Indeed, programmers come in at this stage and are briefed on the system requirements as illustrated using various design tools in order for them to construct a computerised working model of the same.
System construction methods
There are a number of programming techniques that can be used to construct a designed system. These include:

1. Using the high-level structured language such as Pascal, COBOL etc
2. Using fourth generation languages (4GL) - These are easy to use programming languages. Some of the fourth generation languages are Visual Basic, Visual COBOL, Delphi Pascal etc.
3. Customising the standard packages - This involves the use of a ready made software package mostly a database software, financial package or enterprise management system.

Due to the varied approach to system construction available, Chapter 5 in this book introduces you to Visual Basic programming while Appendix I explains how a database package can be customised to construct a system. Figure 4.11 shows a data entry form constructed to enable entering a new book record into the library information database.
Testing the system
After construction, the system is tested by entering some test data to find out whether its outputs are as expected. The system is tested using the requirements specifications and the design specifications to find out whether it meets all requirements specified.
For example, if one of the requirements of the computerised library management system is to ensure that no member is allowed to borrow more than three books at the same time, it must do that without fail. Figure 4.12 shows a message box to this effect.

System implementation
System implementation is the process of delivering the system for use in day to day operating environment for the users to start using it. The areas to be addressed during system implementation include file conversion, staff training, and changeover strategies.
File conversion
Every time a new system is implemented, the format of data files might require modification or change. This process is referred to as *file conversion.* A new system may require a change in file format e.g. from manual to computerised. The factors to consider at this point are:

1. Whether the new system requires a new operating system and hardware. The best practice today is to develop systems that do not need hardware change unless it is very necessary.
2. Whether you need to install new application software. For example if you have developed a new system by Customising a database application software, you need to install that software if it is not installed.
3. Whether you need to create new database files for the new system. For example, where files are manual, electronic ones will have to be made. However, remember that we strive to develop systems that are *data independent* too. That means that the systems can be changed without affecting the organisational data structures in the databases.

Staff training

Availability of appropriate documentation like user manuals goes a long way to make staff training easy, quick and effective. System implementation can fail if the staff are not trained properly leading to great loss of company resources.

Changeover strategies

Changeover simply means how to move from the old system and start using the new. Most businesses especially those that are driven by information technology need as smooth a changeover as possible. Some of the system changeover strategies are:

Straight changeover

In straight changeover, the old system is stopped and discarded and the new system started immediately. This sudden change of old to new means that the project faces higher risks in case the new system faces problems. This is because the old system would not be there to fall back to. The advantage of this method is that it is cheaper because you do not have to run the two systems in parallel. Figure 4.13 shows the straight changeover strategy diagramatically. At a time t, a switch is made from the old to new system.

Parallel changeover

In parallel changeover, both the old and new system are run parallel to each other for some time until users have confidence in the new system then the old system is phased out. This method is a bit costly because extra resources have to be engaged to run the two systems in parallel.

However, its lower risk to business operations and thorough testing of the new system are some of its advantages. This method is not suitable for large systems because of the high operational costs during changeover. Figure 4.14 depicts a parallel changeover process.

Phased changeover

In phased changeover, a new system is implemented in phases or stages. A good example is the way the education system is changed from the old to the new curriculum. Each year at least one class level changes over to the new syllabus.

Sometimes, one phase may run a new system for testing before it is implemented into all the other phases. This is called *piloting.* The main", disadvantage of phased changeover is the danger of incompatibility between various elements i.e. hardware or software of the same system. However, its advantage is that it ensures slow but sure changeover.

Security control measures

Information and data security have become some of the most important aspects of information systems. A lot of careful planning has to be done in order to have what is called inbuilt security in the system. This is because information is under constant threat of being illegally accessed or disclosed to unauthorised parties. Therefore, the system implementers must make sure that the security features built in the system are properly configured during the implementation stage.

System maintenance and review

System maintenance is the adjustment and enhancement of requirements or correction of errors after the system has been implemented. Regardless of how well the system is constructed and tested, errors may be detected when the system is in use.

System review is a formal process of going through the specifications and testing the system after implementation to find out whether it still meets the original objectives. This act is sometimes called *review* and *audit.* If the system does not meet the stated objectives, system development might start all over again.

System documentation

System documentation is a life long process in the system development lifecycle. After a system has been implemented, any maintenance work must be documented in order to update the existing documentation. In this chapter, we have constantly provided sample documentation in every stage of system development using the school library management system case study. Generally comprehensive system documentation consists of the following:

1. Report on fact finding
2. Requirement specification
3. System and module flowcharts
4. Table/file structures description
5. Sample test data and expected output
6. Output reports

Reports on fact finding

At the end of fact finding stage, the system analyst should prepare a well detailed report that mainly outlines:

1. The methods used to collect data.
2. Weaknesses of the current system as evidenced by the collected data.
3. Recommendations: Why there is a need to replace or upgrade the current system.

Figure 4.5 on page 104 shows a sample fact finding report for the school library system.

Requirement specification

The report on requirement specification outlines mainly the:

1. Output requirements for the new system such as reports.
2. Input requirements.
3. Hardware and software required to develop the new system.

Table 4.1 on page 106 gives a sample report expected from a computerised library system, while Figure 4.6 on page 107 gives a simple illustration of an input form for new library members.

System flowchart

The system flowchart shows the overall functionality of the proposed information system. Therefore at the end of the designed phase, the system analyst should write a report that contains:

1. The system flowchart or data flow diagrams that shows the processing logic of the information system.
2. Any module flowchart that may help programmers in construction of the required subsystem or modules. a sample module flowchart.

Table file structures description

Depending on approach used in system construction, the report should contain file or table structure definitions. For example, if you opt to construct a system using customisation approach, details on table structures should be well documented (see Appendix I). Figure 4.15 shows a sample table structure of the Books table in a library system.

Sample test data

To test whether the new computerised information system is working as expected, you need to use test data for every module (subsystem). For example, in our library case study, we need to test sample data for books entry, book borrowing etc. Table 4.2 shows a sample test data that can be entered in the database whenever a book is borrowed.

Table 4.2

Last name	Middle name	Class	Stream	Book borrowed	Date borrowed	Return date

Mburu	James	3	C	Maths Form I	8/6/2005	26/6/2005
Janet	Achieng	3	A	Kamusi sanifu	23/6/2005	30/6/2005
Helen	Mutua	4	B	Elementary chemistry	6/6/2005	24/6/2005
Ali	Mohamed	2	A	Computer studies	6/5/2005	23/5/2005
Kerich	Dennis	I	A	Biology form I	6/6/2005	24/6/2005
James	Kamau	3	B	Computer studies	7/6/2005	25/6/2005

Output reports
To prove that the system is working and giving the desired result, you should provide a number of sample output from various system modules. Figure 4.16 shows a sample report showing a list of members who have borrowed books.

User manual
User manuals are supposed to help a person to use the system with as little guidance as possible. Therefore, the manual must contain information such as:

1. How to install, start and run the system.
2. How the system appears when running (interface).
3. How to carry out various tasks e.g. in our case study, this would include new books entry, lending/borrowing, data entry etc.

4. Error correction and how to get help when faced by exceptions. This would be in a troubleshooting guide.

Figure 4.17 shows a sample main menu screen switchboard from which the user can access other modules.

PROGRAMMING WITH VISUAL BASIC

Chapter outline

Introduction

In the previous chapter, we studied how to develop a system using some system analysis and design methods without necessarily referring to any particular programming language. Therefore, we have been looking at *what* a system analyst ought to do rather than *how* a particular task should be done. In this chapter, we shall learn how to develop programs using Visual Basic programming language. Note that any other programming language can be used to develop programs. Therefore, as a leaner, you are at liberty to select any other programming language and learn how to use it to construct systems.

Definition of Visual Basic

Visual Basic (VB) is a graphical user interface programming language for creating software systems that run under Microsoft Windows environment. It is modeled in line with BASIC language which was originally created to help students learn how to program.

Visual Basic has the following features:

1. It is an *event driven* programming language. An event is a response generated by the program when the user performs an action e.g. a mouse click. The response depends on the code held in an event procedure. An event procedure is a group of self contained statements that are executed by the computer when a trigger action is performed. For example, a mouse click may cause a menu to be displayed on the screen.

2. It has a collection of tools that are used by programmers. These tools are called *controls.* The controls can be accessed easily by clicking their icons on the toolbox to select them.
3. It has special objects called *forms* that have a title bar at the top. The programmer can then add controls such as a menu bar, status bar, toolbars, buttons, and slide bars to the form when creating an application.

Starting Microsoft Visual Basic

You can start Visual Basic from the Programs menu by pointing to Microsoft Visual Studio and then selecting Microsoft Visual Basic. Depending on the way the application has been setup, you may see the New Project dialog box

This dialog box allows you to set up a new project. You can set up several types of projects as shown by the different icons in this dialog box. However, for the purposes of learning, let us select the *Standard.EXE* project found on the *New* tab. Click the Open button. The application window will appear. This is the Visual Basic Integrated Development Environment (IDE).

Features of Visual Basic Integrated Development Environment (IDE) window

The Visual Basic application window (Figure 5.2) has many similarities to common windows based applications.

The following is a summary of the various features identified in the window:

The standard toolbar

Like in many other applications, the standard toolbar has the standard icons that are shortcut commands to the menu bar commands. Simply point to an icon and a text tip will appear telling you the name of the icon. Of great interest to us this time are the three commands of Start, Break and End.

To execute a Visual Basic project, simply click the Start button or select Start1from the Run menu. Execution of the program can be temporarily suspended by clicking the Break button. Resume the run by selecting Continue from the Run menu. Finally, the execution can be stopped by clicking the End button.

Forms and controls

Forms and controls are generally called objects. An object is usually associated with an event hence most objects have their own *properties* and *methods*. The properties define the appearance of the object while the method is a predefined action that can be set to occur or affect the object. For example *show* method makes a form visible on the screen.

The form design window is the place where the programmer does the actual design of the user interface for the program. This is accomplished by selecting the relevant *control icons* from the toolbox then placing them on the form design window. The control can then be moved and resized as desired by the programmer.

The entire form can be moved around on the screen by holding the title bar of the project container window then dragging it to the required position.

Project window

The project window displays a hierarchical list of files associated with a particular project. The files represent forms and program modules in the project. For example, Figure 5.2 shows a form whose title is Form 1.

Properties window

The properties window displays a set of various characteristics unique to each *active* object in the project. Some examples of properties include name, caption, colour, height etc.

When a new project starts, it is most likely that the form design window is active. In this case, the properties for the form will be displayed and can be changed by the programmer as appropriate. For example, we can change the caption of the form by clicking on the *Alphabetic tab* in the properties window then changing the value associated to *Caption* property from Form.l to say "Book Data Form". Notice that the caption in the title bar of the active form changes simultaneously as you type the new name in the properties window.

Code editor window

Figure 5.2 shows that the project window has three icons at the top left hand comer. One of them is called the *code view icon.* Point to the icon with the mouse pointer and a tool tip will appear showing its name. If you click it, the code editor window opens displaying the code of the object that is currently selected. Alternatively, double click the object on the form for which you want to edit its code. The code editor window will open (Figure 5.4). In this case, we double clicked the form.

Form layout window

The form layout window allows you to specify the screen location of the form when the project is Executed. This can be done by dragging the form in this window to the desired location.

The Visual Basic toolbox

Visual Basic tool box has controls that enable a person to design and customise forms. Figure 5.5 depicts the toolbox.

Pointer: This is not actually a control but a tool used to select an object or objects to be worked on.

Picture box: A picture box is used to display graphical objects or text and to initiate event actions. It is similar to an image box but has more properties, redraws slower and cannot be stretched.

Label: A label displays text on a form. The text cannot be reassigned during program execution though its appearance can be altered.

Text box: A text box provides a means of entering or displaying text. The text may be already available or can be entered during the program execution.

Frame: A frame is a tool rather than a control used as a container for a group of controls.
Command button: A control used to activate an event driven procedure.

Check box: As opposed to the option buttons a check box is used if more than one selection are to be made.
Option button: Option buttons are used when one selection must be made before an action is carried out. For example, if you want to turn off a computer running on Microsoft Window 9x, check the *shut down option button* then click Yes/OK.

Combo box: A combo box is a special type of list box that combines the capabilities of a text box and a list box. It provides a list of text items for selection by the user during program execution. Items can also be added during program execution.

List box: The list box control provides a list of text items for selection by the user.
Horizontal scroll bar: A horizontal scroll bar is used for drawing horizontal scroll bars on a form.

Vertical scroll bar: A vertical scroll bar is used for drawing vertical scroll bar on a form.
Timer: Timer control allows timed events to occur repeatedly at a specific time intervals.

Drive list box: A drive list box provides a means of selecting a drive from a list of existing drives.

Directory list box: A directory list box provides a way of selecting a directory from a list.
File list box: A file list box provides a way of selecting files within a group of files.

Shape tool: A shape tool is used to draw circles, ellipses, squares and rectangles within forms.
Line tool: A line tool is used when drawing straight lines within forms.
Image box: An image box control is used to display graphical objects.

Data control: A data control when placed on a form provides a means of displaying information from an existing database.

NB The number of tools available on the tool box depends on the edition and release of Visual Basic you are using

Saving a Visual Basic project

Saving a project in Visual Basic is a bit different compared to other applications. This is because Visual Basic operation involves saving multiple files.

(a) To save a new Visual Basic project for the first time, select the Save project As command from the File menu. The *Save File As* dialog box of Figure 5.6 (a) appears. You will first have to enter a form name e.g. bookdata and then click the *Save* button. The form is saved as a file with extension .frm.

(b) After this, the Save Project As dialog appears. Type the name of the project and it will be saved as a file with extension. *vbp (Visual Basic project*

Opening an existing project

You can open a project when Visual Basic starts or from the file menu. To open a project during startup, from the New Project dialog box that appears when Visual Basic is starting, click the Existing tab. A list of existing projects will be displayed as shown in Figure 5.7

(a). Alternatively, if Visual Basic is already running, close all other projects then click the Open Project command on the File menu. The Open Project dialog box appears from which you can select the Existing tab. (Figure 5.7 (b)). Type the name of/or select the project to open.

Visual Basic fundamental concepts

Before going further into detailed discussion on how to develop programs in. Visual Basic, it is important to highlight some of the fundamental concepts used in Visual Basic.

Data types

Table 5.1 shows the data types supported by Visual Basic:

Table 5.1

Data type	*Description*
Integer	A whole number with no fractional part. Integers range from -32768 to 32767.
Long integer	Numbers which are integers but have a bigger value and range. They range from -2,147,483,648 to 2,147,483,647.
Single	A single precision real constant includes a fractional part. The largest value is 3.4 x 10^{38}
Double	It includes a fractional part but has far much larger magnitude than the single real number constant. However, it cannot be larger than l.2x 1 0^{308}
String	Characters enclosed in quotation marks
Boolean	Data types that have only two logical states i.e. true or false.

Constants

In programming, a constant is a value that remains the same. It does not change during the execution of the program. There are three types of constants: *string, numeric* and *named* constants.

String constants

A string constant can be defined as a sequence of characters enclosed in quotation marks. They are used to write non-numeric values like telephone numbers, addresses and names. The following are examples of string constants: "Holiday season", "345678", "Why don't you call", "$72.10"

Numeric constants

Numbers are usually referred to as constants in Visual Basic. In most cases, the numeric constants are either whole numbers (integer) or double or single. Some examples of numeric constants are:

0, 3, 6000 +70,   1.4E+2, 1.674, -.00456,

Named constants

A named constant is identified by name rather than its actual value. In Visual Basic, the reserved word *Canst* is used to declare this type of constants e.g. *Canst* Price *As* Integer = 20, means that an integer value 20 is identified using the name *price.*

Variables

A variable is a memory location referred to by name used to hold a value that is subject to change during program execution. When a variable is declared in a program, the computer sets aside memory space to hold a value in the variable. The variable may be a numeric constant, a character, a string or any other data item.

In Visual Basic, the following rules should be followed when declaring variables:

I. The variable name must not have more than 255 characters.
2. The variable name must start with a letter.
3. The letter case is not important when declaring variables.
4. A variable name must not be a *reserved* word. A reserved word is a special word that Visual Basic understands as a command, internal function name or simply set aside for use by the program. Examples of reserved words are *Case, Else, Dim,* etc.
5. Variable names should not have spaces.

A variable is declared using the reserved word *Dim.* Dim is the short form of the word dimension (size). It is used to associate a variable with a specific data type. For example to declare a variable that will store the first name of a student in a program, the following declaration can be used:

Dim FirstName *As* String

This means that the variable FirstName is of the type string.

Variants

A variant is a variable whose data type has not been explicitly declared by the programmer. In this case, the variable type will be determined by the value held by the variant. These types of variables are called variant type because their data type keeps on changing with the values they hold. Although this may sound okay, it is not a good programming practice. All variables should be explicitly declared.

Scope

The word scope refers to the level of the program that a variable, a constant or a procedure is recognised. The scope of a variable constant is said to be global or local. The scope of a procedure variables or constants is set by declaring it as either *private* or *public.* The following terms are used in this regard:

I. Private subprocedure.
2. Public subprocedure.
3. Global variables/constants.

4. Local variables/constants.

A public procedure can be accessed from any module or form in the program. However, a private procedure is accessed in the module or form within which it is declared.
A variable or constant that is declared within a procedure is said to be local. This means that it can only be accessed within that procedure. However, a variable or constant that is declared outside the standard module but within the same project is said to be global. Such variables can be accessed by all procedures within the project.
If a public procedure is to be accessed from a module or a form other than the one it is contained in, the form or module name must be preceeded by the module containing the definitions i.e. *GlobalModuleName.Active FormName (Arguments).*
In Visual Basic, to declare variables as either private or public use the syntax:
1. Private X As Integer
2. Public Y As String
In the first declaration, the private variable X can only be accessed in the module within which it is declared. In the second example, it can be accessed by other subroutines in the same project.
User defined data types
Sometimes, a programmer may want to have a single variable that can store several data types as a unit. Such a data type whose individual components are standard data items are referred to as *user defined data types.* Some examples of user defined data types are *records, arrays, enumerated types, lists* etc. They are user defined because it is the programmer who creates them in the program. Each element or member can be accessed individually by using a period between the user defined variable and the member name e.g. if a record known as schoolrecord has a name field, then to access the name field, a programmer will type *schoolrecord.name* in the code.
Using suffixes to declare variables and constants
Suffixes are special symbols appended to the end of a variable or constant name in order to associate them with a particular data type. It makes programming easier and faster. Table 5.2 summarises some common
suffixes.

Table 5.2

		Examples	
Suffix	Data type	Long declaration	Short declaration
%	Integer	Dim A As Integer	A%
!	Single	Dim X As Single	X!
$	String	Dim Q As String	Q$
&	Long integer	Dim C As Long Integer	C&
#	Double	Dim P As Double	P#

Mathematical operators
In order to write correct mathematical expressions, you need to understand some of the operators used in Visual Basic. These include *arithmetic, relational* and *logical* operators.
Arithmetic operators
Arithmetic operators are special symbols that are used to write arithmetic expressions like addition, subtraction, division and multiplication. However, it is also possible to carry out other mathematical operations using other operators. Table 5.3 shows a summary of symbols used in arithmetic operations:
Table 5.3

Symbol	Name	Operation

+	Plus sign	Addition
-	Minus sign	Subtraction
*	Asterisk	Multiplication
/	Slash	Division
/\	Caret	Exponentiation
"	Back slash	Integer division
MOD	Modulus	Integer remainder

NB: The last two rows in the table can be explained as follows:

1. In an integer division, each of the numbers being divided is first rounded to become an integer then the quotient is truncated to an integer. For example, 2.4/3.5 will result in 2/4 being evaluated.
2. The MOD operator returns the remainder of an integer division. For example, 4 MOD 3 returns 1 and 5MOD3 returns 2.

Operator precedence/hierarchy

Arithmetic operators in Visual Basic have the following order of precedence when used in a program. Notice that the order may not be the same as that of algebra in mathematics. Table 5.4 illustrates how to carry out operations.

Table 5.4

precedence	symbol	Explanation
1	Exponentiation	All exponential expressions are performed first
2	Multiplication/division	Carried out after exponentiation; left to right In an expression.
3	Integer division	Done after all multiplication and normal division operations
4	Integer remainder	Done after integer division
5	Addition and subtraction	Are carried out last, left to right.

For example write down the order of evaluation of the following expression:
(b /\ 2 - 4 * a * c)/2 * a.

Relational operators

A relational operator is used in an expression that returns a true or false value when evaluated. The operators can compare numeric variables, constants or expressions. Table 5.5 shows a summary of relational operators used in Visual Basic.

Table 5.5

Operator	Name
=	Equal to
<>	Not equal to
>	Greater than
<	Less than
<=	Less than or equal to
>=	Greater than or equal to

The following are examples of relational expressions:
1. Y < > 100 2. Y> 60 3. S< = sqr(G + V) 3. Loss = (Expenses - Profit)

The expression in 1 will evaluate to true if y is not equal to 100 otherwise it will evaluate to false.

A decision construct in a program based on a relational expression would make a decision depending on the outcome of the evaluation.

Logical operators

Logical operators are used to combine expressions to form compound expressions. Table 5.6 summarises some common logical operators used in Visual Basic.

Table 5.6

Operator	*Operation*
And	Results in a condition that is true if both expressions are true.
Or	Results in a condition that is true if one of the conditions is true or both are true.
Xor	Results in a condition that is true if one of the conditions is true and the other is false.
Not	Negates the value of a logical expression.

For example:

1. (x = 30) AND (student = "John")
2. IF (x = 20) OR (y = 30) THEN
3. IF x NOT 0 THEN

The assignment statement

An assignment statement is an executable statement that assigns whatever is on the right of the assignment operator to the variable on the left. In Visual Basic, the assignment operator is the equal sign (=). A variable can be assigned a value as follows:

Variable = Expression

Examples of assignment statements are:

1. P = 0.25 2. E = M*C"2 3. Area = 1/2 * base* height 4. X = X + I

NB: The last statement on the right (X = X + I) does not make mathematical sense. Yet, this is the beauty of the assignment statement in programming. The statement can be interpreted as follows:

Add 1 *to the original value of X and assign the sum to X.*

The Print statement

The *Print* statement displays data on the active form of your Visual Basic project. It can be used to display processing results of a program. If you run the project, you will notice that the result of the Print statement starts being displayed in the top left comer of the form. The structure of the Print statement is:

Print VariableName (this prints out the value held in a variable).

or

Print "String"

(this prints out the string between the two quotation marks. If there is nothing between the two quotation marks, a blank line will be the output)

The skeletal code listing below shows a sample code that displays the values stored in the variables StudentName, TotalMarks and AverageMark on a form.

Private sub Form_Load ( )

Dim StudentName As String, TotalMarks As Single, AverageMark As Single

```
...........
StudentName = "Edward"
 TotalMarks = 1000
AverageMark = 200
……………
Print, StudentName, TotaIMc;zrks,AverageMark
End sub
```

The following output will be displayed on the form: Edward 1000 200

NB: The use of a comma to separate the various elements to be output by the Print statement makes them to be widely spaced on the screen. To have a compact display with the various outputs close to each other, then use a semicolon to separate the various elements e.g. Print StudentName; TotalMarks AverageMark would display

```
Edward      1000        200
```

Library functions

Library functions are modules that have been pre-written and included in the Visual Basic language. A function performs a single task like calculating a mathematical expression then returns a value. A function will usually have a name and can be called when needed to perform a particular task. Visual Basic has many library functions. A library function is accessed by stating its name followed by the information that must be supplied to the function enclosed in parenthesis. This process is called *calling a/unction.* The values held in the parenthesis are called *arguments.* When the function is called, it uses the supplied arguments to perform an operation and *return a value.* Table 5.7 shows a summary of some the library functions used in Visual Basic.

Table 5.7

Function	*Call*	*Description*
Date	z = Date ( )	Returns the current date
Exp	z = Exp(w)	Returns the value of eW
Sqr	z = Sqr(w)	Returns the square root of w
Log	z = Log(w)	Returns the natural logarithm of w
Str	z = Strew)	Returns a string whose characters are w e.g. Str(6.80) = "6.80"
Cos	z = 'Cos(w)	Returns the cosine of w
Abs	z = Abs(w)	Returns absolute value ofw e.g. Abs(-8) = 8
U case	z = Ucase(w)	Returns the upper case equivalent of w
Int	z = Int(w)	Returns the largest integer that is algebraically less than w e.g. Int(2.2) = 2
Time	z = Time	Returns the current system time.
Format	z = Format	Formats the output to appear as stated.

Using the format function

The format function helps the programmer to display data in many different formats. Table 5.8 below shows how data can be formatted.

Table 5.8

EXPRESSION	SAMPLE FORMATTED OUTPUT
Print format(16.778994,”##,##”)	16.78(notice the rounding)
Print format(now, “mm-dd-yyyy”)	1-20-2006

Print format (15678, “##,###.00”)	15,678.00
Label1.caption=format$(sprice, “##,###,##”)	10,630.75

NB: The dollar sign may be used with the format function depending on the type of output required. In the last row of the table, the formatted output is displayed on a label control.

Program comments

Comments are English statements included in the code to *document* a program. The comments are not executed when the program is running. In Visual Basic, a comment starts with a single apostrophe (') followed by the comment statement. For example:

```
X = Sin(p) 'find the sine of p and store it in X
Print X    'display the value ofX on the screen
```

Converting a numeric string to a value

If you enter a number in a text box, the best way to convert it into a value is to use the *Val* function This is because the computer treats anything typed in the textbox as a string. The *Val* function operates on the string and returns a numerical value.

Hence, if the name property of a text box is set to txtLength, then the number typed in it can be read as a value into a variable x as follows:

x = Val(txtLength.Text)

Converting a value to a string

Assuming a particular variable has a numeric value and you wish to display the value in a text box, then you have to convert the value to a string first. Suppose that the name of the text box is txtArea and the integer variable is x then:

txtArea.Text = Str(x) ‘ display the value in x as a string in txtArea.

Project 5.1

Create a program that can be used to calculate the area of a rectangle. The program should prompt the user to enter the length and width of the rectangle. Figure 5.8 shows an overview of such a program.

To create the project, proceed as follows:

1. Open Visual Basic and select the Standard.EXE from the New Project dialog box. A blank form will be displayed as shown in Figure 5.9.

2. Place the necessary controls on the form. We need three text boxes and two command buttons. Double click the text box tool on the toolbox then the command button. The controls are automatically placed on the form and are given default names like Text! and Commandl. Add another command button. Use the shape tool to place a rectangle on the form. Drag the controls to the desired location on the form and repeat the process until you have the interface as shown in Figure 5.10.
3. The next step is to define the set of properties for each control. To do this, right click the control then click properties. Set each control properties in the properties window as shown in the Table 5.9 below:

Table 5.9

Object	*Property*	*Value*

Label1	Caption Name	"Rectangle" IblRectangle
Text1	Font Name Text	Arial txtLength (blank)
Text2	Font Name Text	Arial txtWidth (blank)
Text3	Font Name Text	Arial txtArea (blank)
Shape	Shape Border Width FillStyle FillColor	0- Rectangle 2 4-Upward Diagonal Blue
Commandl	Caption Name Font	Exit cmdExit Arial
Command2	Caption Name Font	Calculate cmdCalculate Arial

NB: The conventional way of setting the name property of any object in Visual Basic is to use object prefixes such as txt for text or boxes, lbl for labels, cmd for command buttons etc.

4. After setting the properties, we can now write the event procedures that calculate the area of the rectangle once values are entered in the txtLength and txtBreadth text boxes. To write the code, double click calculate button and add the code listing between Private sub cmdCalculate click and the End sub.

```
Dim L As Single, W As Single, A As Single
 L= Val(txtLength. Text)
 W= Val(txt Width. Text)
A=L*W
txtArea= Str( A)
```

For the cmdExit button, just write End between Private sub cmdExit click and the End sub.

Figure 5.11 shows how your code should look like in the code window.

5. Run the project. Enter a value in the txtLength and another in the txtWidth text boxes then click the Calculate button. Notice that the area of the rectangle is displayed as shown in Figure 5.12.

Explanation

1. The first line declares the variables needed for the program to run and store values.
2. The second and third lines assign the values in the text boxes called txtLength and txtWidth variables Land W respectively.
3. The second last statement calculates the area of the rectangle while the last uses a function *Str* to display the result as a string value in the textbox named txtArea.

Adding more forms to a project

Sometimes a Visual Basic project may require the use of more than one form. To add a form to a project, the following procedure is followed:
1. Click the Project menu then select Add Form command.
2. In the dialog that appears, select the type of form you want to add from the New tab then click Open button.

Control structures

In Chapter 3, we introduced the three control structures that determine the execution of statements in a program. In this topic we shall discuss how these controls are implemented in Visual Basic.

Selection

The most common selection controls used in Visual Basic are:
1. IF...... THEN.
2. IF... THEN...... ELSE.
3.Nested IF.
4.CASE selection.

The IF...THEN selection

The IF. . . THEN selection is used to execute a single statement or a group of statements that represent only one alternative or option. This means that some circumstances have only one option for selection depending on whether a condition evaluates to true or false. For example IF y = 80 THEN

```
Print ("……..")
END IF
```

The statements in the If...Then...End If will be executed if the value of y is equal to 80 otherwise the whole block will be bypassed and the statements after END IF will be executed.

IF...THEN...ELSE selection

The IF... THEN.. . ELSE allows the execution of two alternatives or options depending on whether the test returns a true or false. Consider the following example:

```
IF average> 80 THEN
    Print("Excellent work! Congratulations")
ELSE
    Print("Work harder next time")
END IF
```

In this case, the statements starting with 'Excellent Work! ....' will be displayed on the screen if the average is greater than 80. Otherwise, if average is less than 80, the statement "Work harder next time" will be displayed on the screen.

Nested IF selection

Nested IF selection is used if there are more than two available options to choose from. The general format of Nested IF is

```
IF <condition> THEN
```

```
    Statements
ELSE IF <condition> THEN
    Statement
ELSE IF <condition> THEN
    Statement
ELSE <condition> THEN
    Statement
```

Project 5.2

Write a program that can be used to find the roots of a quadratic expression.

$ax^2 + bx + C = 0$

The formulas to solve the values of x in this expression are:

$$X_1 = \frac{-b - sqr(b^2 - 4ac)}{2a}$$

$$X_2 = \frac{-b - sqr(b^2 - 4ac)}{2a}$$

$$X = \frac{-b}{2a}$$

i.e.$b^2 - 4ac = 0$ or $b^2 = 4ac$

To create the project, proceed as follows:

1. Test whether b^2 is larger than or equal to 4*a*c. If it is true, then:

2. Create a new project that would be used to calculate the values of x, X_1 and X_2 also called roots of the quadratic equation.

3. In the appropriate event procedure e.g. the calculate button, add the code listed below:

```
 'Calculating roots of a Quadratic Equation
Dim a,b,c As Integer
Dim z,X,X1,X2 As Double
z=(b^2-4*a*c)
IF z> 0 THEN
    X1=(-b+Sqr(d)) / (2*a)
    x2 –x1=(-b-Sqr(d)) / (2*a)
ELSE IF z= 0 THEN
    x = -b / (2*a)
ELSE IF z= 0 THEN
    Print ("This quadratic equation is not soluble")
    END IF
ENDIF
ENDIF
```

Explanation

1. The expression b^2-4*a*c is first assigned to a variable z.

2. z is then tested if it is greater than zero. If greater, the program calculates the values of X_1, X_2 and x. Otherwise a message "This quadratic equation is not soluble" is displayed.

The Case construct

As discussed earlier in control structures of chapter 3, the Case construct is used to reduce the tedious work of coding associated with the nested IF... construct. Let us look at a case statement that can display the name of a weekday when its number is provided.

```
 'Display weekday
Dim D As Integer
SELECT CASE D
CASE 0
     Print("Sunday")
CASE 1
     Print("Monday")
CASE 2
     Print("Tuesday")
CASE 3
     Print(" Wednesday")
CASE 4
     Print("Thursday")
CASE 5
     Print("Friday")
CASE 6
     Print("Saturday")
CASE ELSE
     MsgBox("Error -That is not a day in the week")
End SELECT
```

Looping construct

In Visual Basic looping instructs the computer to execute the same block of code many times before a particular condition is met. Common looping constructs used in Visual Basic are:

1. DO loop.
2. WHILE...Wendloop.
3. FOR loop.

DO loop

There are *four* ways of writing the Do loop. Two of the constructs require the condition to be tested first while the other two require that the condition be tested after the statement within the construct is executed at least once. The general formats of the *four* Do constructs are:

```
1. DO WHILE <condition>
     Statements
     Loop

2.DO Until <condition>
    Statements
Loop
     3. DO
     Statement,
Loop UNTIL <condition>

4. DO
    Statements
```

Loop WHILE <*condition*>

Explanation
The Do While. ..Loop in 1 continues to execute as long as the condition is true. The Do Until. ..Loop in 2 continues to execute as long as the condition remains false. The loop in 3 executes the statements at least once as long as the condition is not true. Lastly, the loop in 4 executes the statements within the construct at least once as long as the condition remains true. *for* the first two DO statements, the test of the condition is done at the beginning of the loop hence avoiding execution of any statements in the loop as long as the condition required is not met. On the other hand, the last two loops allow a *pass* through the loop at least once before the condition is tested.
example 5.1
Write a program using the DO loop that will write numbers between 1 and 20 on the screen then stop. Save the project as Example 5.1.
Solution
1. Double click the form to display its code window
2. Add the following code listing between the Private sub form-load and
End sub.

```
x=O
Counter=O
DO WHILE counter<=20
        x=x+1
        Counter=counter+ 1
        Print x
        Loop
```

NB: All the other DO loops can be easily implemented with little effort following the general format provided.
Looping with While-Wend
The While..Wend construct can be used in place of a Do While loop. The general format of this construct is:

```
While <condition>
      Statements
Wend
```

As with the Do While loop, this loop executes the statements within the construct as long as the condition is true.
Practical activity 5.1
Open the project for Example 5.1 and replace its code with the code below.

```
Dim x As Integer, Counter As Integer
x=O
Counter = 0
While counter<=20
x=x+1
counter=counter+1
      Print x
Wend
```

Save the project as Example 53.
The FOR - NEXT loop

The FOR - NEXT Loop executes the statements within the construct a predetermined number of times. The general format of the FOR - NEXT
Loop is:

```
For index = lower limit To upper limit          'sets upper and lower limits
statements
Next index                           'increments index by 1.
```

You can set the STEP value of the index as follows:

```
For index = lower limit To upper limit STEP 2
    statements
Next index                       'STEP increments index by 2.
```

The FOR construct is very suitable for accessing values in a list or *array* because of their sequential nature.

Practical activity **5.2**
Open the project for Example5.3 and replace the code with the listing below:

```
Dim Number As Integer, Counter As Integer
Number= 0
For counter= 1 TO 20 STEP 2
    Number= Number +  counter
    Print Number
Next counter
```

Working with graphical objects

To add a picture or object on to a form, simply click the picture box or the image tool on the toolbox and then drag to define the area where you want the picture to be placed. Right click the object and then click properties. Set the Name and Path properties. The Path property tells the control where the picture is located.

Notice that after setting the properties, the picture is displayed on the form. If it is lying on top of other controls and you want it to be the background, right click it then select *Send to Back.*

Modules and procedures

Large projects must be made modular in nature to enhance maintenance and independence of the various blocks of code. The three main modules used in Visual Basic are explained in Table 5.10.

Table 5.10

Module	*Definition*
Standard modules	Their declarations and procedures can be accessed by other modules. They are usually stored in files that have the extension. *bas* You can create a standard module by clicking Project then Add Module.
Form module	Are stored as files with extension *frm* Each form may have many procedures and controls. To add a form to the project, click the Project menu then Add Form.

Procedures	These are events or self contained blocks of code or commands that can be accessed and used by programmers. The *called* procedure will normally carry out a specific function it was designed to do. Subroutines and functions are examples of procedures.

Declaring general subroutines

So far, we have mainly discussed event driven procedures. Visual Basic also lets the programmer create general procedures that can be *called* within another procedure (subroutine). The general format of a general purpose subroutine is:

```
 Sub SubroutineName( )
      statements
End Sub
```

Example 5.2

Write a general subroutine that would solve y = x^n , given that the values of n are integers.

Solution

```
Sub Power of Number ( )
Dim x As Single, y As Single, n As Integer y = x^n
Print y
End Sub
```

Explanation

The above general subroutine is given the name *PowerofNumber*. To call the subroutine in another procedure such as an event procedure, simply use the *call* statement. For example, to call the subroutine in the cmdCalculate _Click event procedure, type:

Call PowerofNumber () *or* Power Of Number ()

The code is executed once the command button for the cmdCalculate click event procedure is clicked.

Project 5.3

Consider a subroutine that will determine the largest of two numbers. Use the code listing below to create a proj ect that can be used to determine the lagest value once the user clicks a command button.

```
Sub Largest ( )
Dim Larger As Integer, x As Integer, y As Integer
If (x>y) Then
      Larger = x
      Print ("The larger number is: ",x)
Else If (x<y) Then
      Larger = y
      Print ('The larger number is ",y)
Else
      Print("Both numbers are equal")
   End If
 End If
End Sub
```

To create the project, proceed as follows:

1. Open a new project and save it as LargerNumber.vbp. Save the form as LargerofTwo.frm. Set the objects properties for the three labels (Tabe 5.11) and design the interface as shown in Figure 5.13.

Table 5.11

Control	Properties	Settings
Labell	Name Caption	lblLarger Larger between
Label 2	Name Caption	lblx x
Label 3	Name Caption	 y
Text! ,2,3	Name Text	txtLarger, txtx, txty (blank)
Command 1	Name Caption	cmdExit Exit
Command2	Name Caption	cmdCompare Compare

Double click the Calculate and Exit *buttons and* add *the codes as shown:*

```
    Option Explicit
    Dim x As Single, y As Single
    Private sub cmdCompareXY- Click ()
    X = Val (txtX Text)
    Y = Val (txt Y. Text)
     call Largest                'calls a procedure named Largest that
End Sub                            passes the values of x and ythe
                                       procedure.
```

3. Create a general subroutine called Largest and write the code listing below:

```
    Sub Largest ( )
    Dim Larger As Single
    If (x > y) Then
        Larger = x
    txtLarger = x
    ElseIf (x < y) Then
        Larger = y
    txtLarger. Text = y
    Else
        txtLarger. Text = "The numbers are equal" End If
    End Sub
    Private Sub cmdExit – Click()
    End                          ‘code for the  exit button
    End Sub
```

This code will appear as shown in Figure 5.14. The call statement in the cmdCalculate makes the *Largest* Sub procedure to be executed once the End sub is encountered, the control is returned back to the next statement often the procedure call.

4. Click the run button and enter a value in the text box labelled X and the other in Y. Figure 5.15 shows that the value of Y i.e. 50 is larger than that of X.

Creating menus, message and dialog boxes Menus enable the users to quickly select commands while dialog boxes enable the user to enter values and make selections when working with the computer.

Drop down menus

Drop down menus are the most common menus found in applications. When a menu is selected, a drop down list of commands is displayed.

For example to create a menu bar with the menu items File, Edit and Format with commands for each menu item as shown in the Table 5.12 below.

Table 5.12

Menu	File	Edit	Format
Commands	open Save as Exit	copy Cut Paste	font Background Cells

Proceed as follows:

1. Click the Menu editor command on the toolbar. Alternatively, from Tools click the Menu Editor command. The menu editor window opens.
2. Under caption, type the word File. This is the screen name of the menu item.
3. In the Name box enter mnuFile. This is the name that will be used to refer to the File item when coding.
4. Under Shortcut, select a shortcut key combination. This is optional.
5. Click Insert button and notice that the cursor moves to the next line.
6. Type Open in the Caption box and mnuFileOpen in the Name box.
7. Click the flush right button to insert four dots to the left of Open i.e.

```
    File
...........Open
```

This means that Open is an item of the File menu. Repeat steps 1 to 4 for all the items then click the OK button. Your complete set up should appear as shown in Figure 5.16 below.

8. Click OK to close the menu Editor window and go to form design. You will notice that your form now has a menu bar at the top as shown in Figure 5.17 below. Clicking the File menu displays a list of the options that you defined.

It is now possible to activate the menu commands by adding code in their event procedures. To do this, simply click a menu option while in design view and the code window opens in which you can add its code.

Message boxes

A message box is a special type of dialog box used to display a message to the user. To display a message box, use the syntax:

MsgBox *"Message string", Button/icon, "Title bar Caption".*

For Example: MsgBox("Drive not Ready. Retry?", vbRetryCancel + vbQuestion, "Drive Error")

Displays a message box (Figure 5.18) that tells user that the drive is not ready. Note the *Retry* and *Cancel* buttons, the *Question mark* in the dialog box and *"Drive Error"* on the Title bar.

Table 5.13 outlines some of the buttons/icons used in Visual Basic

Table 5.13

button	Constant
OK	vbOKonly
Critical message icon	vbCritical
Critical query icon	vbQuestion
Warning message icon	vbExclamation
Information message icon	vbInformation

Dialog boxes

A dialog box enables the user to have a "dialog" with the computer i.e. to exchange information with the computer. You have seen these many times when you try to open or save a file; a dialog will always pop up asking you to fill or select some options. A dialog box is a special purpose form. It is usually accessed when a menu item is selected. Most dialog boxes have common controls like option buttons, list boxes and command buttons.

Simple dialog boxes

Visual Basic enables the user to create forms that can be used as dialog boxes. However, they are secondary to the primary form that is used to access them. Such a form is referred to as a *modal* form. This means that the form remains on top of the active form until the user clicks the OK or cancel button. To add a modal form to your project, use the *Load form* command. For example, if the name of the form is Form3 then the *Load Form3* command will display the form into the current project. To stop the form from executing, use the unload command i.e. *Unload Form3.*

In most cases, a form that is loaded may not even be visible in the project window. To make it visible, simply use the Show method. Remember that a method carries out actions on objects. For example to display Form 3, use the syntax Form3.Show. To show the form as modal, use Fonn3.Show l or Form3.Show vbModal.

Project 5.4

Open a project on which you created the menu bar and do the following:

1. From the project menu click Addform then click New from the Add Form dialog box.

2. Select dialog A form with two command buttons ie Ok and Cancel is added to your project. Notice in the Project Explorer window that a new form called Form2 is placed hierarchically below the first form.
   Change the form caption and name appropriately.

3. From your main form file menu click the Open command.

4. In the menuFileOpen code window that opens add the code as shown in Figure 5.19.

5. Close the code editor window and run project. Notice that when you click File then Open the dialog box you created is displayed on top of Form 1 (Figure 5.20). Because of the Show 1 method, you cannot access the main form until you click ok or cancel button on the dialog box.

Directory list box

You can also add a directory list box on a form. To do this, proceed as follows:

1. In the Project Explorer window, double click Form2 to open it in design view.
2. Double click the DirListBox on the toolbar to place it on the form.

3. Run the project and select the Open command from the File menu, you will be able to browse all the directories on your computer using the dialog box that you just created.(Figure 5.21).

Common dialog box control

Microsoft common dialog control is a component used to add dialog' boxes such as Save As and Open onto a form. The *common dialog control* I is not available on the toolbox unless it is added. To add it, click *Project* then *Components.* The components dialog box as shown in Figure 5.22 will appear. Select the option Microsoft common dialog control as shown, then click the *Apply* button. Close the components dialog and check your toolbox for the new icon called common dialog.

Project 5.5

Using the common dialog box control, create a project that can be used to display the open and save As dialog boxes. To create the project, proceed as follows:

1. On the form, create a menu bar with menu items File, Edit and Format and two command buttons with captions *open* and *save* as shown in Figure 5.23.
2. Right click the common dialog control on your form to display its properties window. Set the name property e.g. commonDiagl.
3. Double click Open from the File menu or the Open Command button on the design form then add the statement below between the Private sub and End sub.
    CommonDiagl.ShowOpen
4. Double click the mnuSaveAs or the Save Command and type the code CommonDiag I. Show Save

Save the project and run the form. Notice that if you click the Open command or button when the form is running, the Open dialog box opens as shown in Figure 5.24.

In the same way a Print dialog box can be called by adding the following to a menu or button event: *CommonDiag 1.ShowPrint*

Other dialog boxes supported by common dialog box are Show Color
and Show Font.

List boxes and combo boxes

List boxes and combo boxes are used to display a list of items the user can select from.

List boxes

A list box enables the user to select an item from a list of entries. Selecting an item on a list during runtime causes the value of the selected item to be assigned to the *ListIndex* property. The ListIndex is an integer with a range of 0 to n-I where the second item corresponds to index 1. Items can be added to a list using the *AddItem* method or using the list function.

Project 5.6

Create a project that leads a list of items into a list box when the form is loaded.
To create the project, proceed as follows:
I. Place the list box on the form and then drag it to the expected size.
2. Double click the Form and add the following code window in the Form_Load() event procedure:

```
Listl.List(O) = "Nairobi"              Listl.Addltem("Nairobi")
Listl.List(1) = "Kisumu"               Listl.Addltem("Kisumu")
Listl.List(2) = "Mombasa"              Listl.Addltem("Mombasa ")
Listl.List(3) = "Bungoma"       or     Listl.Addltem("Bungoma")
Listl.List(4) = "Nyeri"         Listl.Addltem ("Nyeri")
Listl.List(5) = "Nakuru"        Listl.Addltem("Nakuru ")
```

Alternatively you can add the items in the list property of the list box. To do this, display the list properties window, in the list property, enter the items. To move to the next line, during item

entry, press Ctrl + Enter. Figure 5.25 shows the item list assigned to a list box using the properties window.
3. Run the project. You will notice that the list builds dynamically as you click in the box when the form is running.

NB: The ListIndex and ListCount properties are very important. ListIndex can be used in CASE selection construct to point to the most recently selected list entry. For example,

```
Select Case Listlndex
Case 0          'first entry in list corresponds to 0, not 1
    "Nairobi"
Case 1
"Kisumu "
Case 2
    "Mombasa"
Case 3
..........."        ..
End Select
```

On the contrary, the ListCount property shows the number of entries in a list and it starts from 1, not O.

Combo boxes

As opposed to a list box, a combo box allows the user not only to select but also make a new entry into the list of items.

Project 5.7

Create a project that loads a list items into a combo box when the form is loaded.
To create the project, proceed as follows:
1. Place the combo box control on the form.
2. Open its properties window and in the *List property* add the following items:
"Red"
"Yellow"
"Green"
"White"
"Black"
3. Run the project to load the form. Click the down arrow on the combo box to view the list of item as shown in Figure 5.26 below.

Visual Basic data structures

Apart from using simple data structures such as Integer, Single, Double etc., Visual Basic comes with features that allow the programmer to create complex data structures. These include *arrays, records* and *files.*

Arrays

An array is a data structure that stores several data items of the same type hence it is called a homogeneous data structure. Table 5.14 shows how data of type Integer is stored in an array.
The array has cells. The numbers 0 to 5 are called *array indices or subscripts.* In Visual Basic, an array starts from cell 0 as shown in the table.

Declaring an array

To declare an array, use the statement *Dim ArrayName (n),* where n stands for the number of elements in the array. For example:

Dim Scores (4) is an array that holds five elements. An element is an item in an array that can be accessed using the array variable name and a subscript. For example, consider an array of 5 elements called Scores. To display the 4th element use:

```
Score4 = scores(3)
Print Score4
```

The value stored in the variable score4 is printed on the form. To store a value in an array variable, use the statement ArrayVariable (index). For example, to assign a numeric constant 90 to the fifth cell use

```
scores( 4) = 90
```

This statement will assign a value 90 in the cell 4th of the array.

NB: If the array was very large and you wish to read or enter values in it, it would be very tedious entering a value cell by cell i.e. Score(O) = 10

```
Score( 1) = 20
Score(3) =30
Score(n) = x
```

To avoid this, the *For.. loop* can be used. Remember that this loop is used where the number of iterations are pre-determined hence its suitability for use with arrays.

For example, if you wish to request the user to enter six values in array named A, the following FOR loop can be used:

```
Dim A(5) As Integer
Dim index As Integer
Private Sub EnterArrayValues()
      For index= 0 To 5
      A(ifldex) = Val(Textl.Text)
      Next index
End Sub
```

Notice that the array is declared this time outside the Subroutine in order to make it a general procedure that can be accessed by other subroutines. To Display values of the array on the form the following statement can be used:

```
Private Sub Display Array Values ()
 For index = 0 To 5
      Text2. Text= Str(A(index))
      Next index
End Sub
```

Two dimensional arrays

A two dimensional array is a data structure in which elements are arranged in rows and columns. Two subscripts are used to identify an item. For example Dim Score(2,4) means that the element is in the 3rd row, 5th column. This is because the array was declared to start from row 0 column 0 same as Dim Score (0 to 2, 0 to 4). To manipulate elements stored in a two dimensional array, we use the nested For.. Next or For Each ... Next loop as shown below:

```
Dim iRow As Integer
Dim jColumn As Integer
For iRow = 1 T03
    For jColumn = 1 to 5
 Score(iRow, jColumn)
```

```
    Next jColumn
Next iRow
```

Records

A record comprises of a series of related data items (fields). They differ from arrays in that whereas all elements in an array are of the same data type, data items in a record may differ in type. For example, when you want to store a students record with the fields Name, Registration number and date of birth in a file as a unit, then you can define a record that would hold all this data items in one structure. For example the following declaration will create StudentDetails record:

```
Type StudentDetai/s
    Name As String 15
     RegistrationNumber As String 10
    DateofBirth As Date
…………        .
End Type
```

The first line in the type declaration gives the name of the data structure i.e. StudentDetails. The other three lines define fields in the data type. Notice that each field has its own data type and that the string data types are of fixed length. The last line i.e. End type marks the end of the record definition. A record is defined in a standard code module or the General Declaration section of a form module.

It is not possible to access the elements of a data structure directly. This is why after defining data structure, you must declare a variable that will be used to hold the records. This is done by declaring a variable whose type is the data structure. The general format of such a declaration is:
Dim *RecordVariablelVame* As *RecordName*
For example, the StudentDetails record can be declared as a variable using the statement:
Dim *Student* As *StudentDetails*
This means that the variable Student is of type StudentDetails. To access an element in the structure, the following syntax have to be used:
RecordVariable.FieldName e.g. Student.Name

Array of records

An set of user-defined data types can be stored and accessed in arrays too. Consider the record declared below:

```
Type Student
     RegistrationNumber As Integer Name As String * 20
End Type
```

An array of the student record can then be declared as follows: *Dim StudentArray(39) As Student*
This means that the array can store fourty student records. To access a field in the array of records use the statement: *Arrayname(index).FieldName* e.g. *StudentArray(20).Name* will return the name of the student whose record is in the 20th cell.

Data files

A file can be defined as a storage location of related information or records. Visual Basic recognises three types of files namely: *text (sequential) files, random access files* and *binary files.* Sequential files consist of data made of text strings. A *random access file* contains data records each of equal length with each record having a unique identifier. Remember that a record is a data structure that has several related data items about a particular entity. In random files, a record can be accessed directly by specifying its corresponding record number or location. A *binary file* contains information expressed as a stream of bytes. Such files store graphics, sound files or compiled code.

Sequential files

A sequential file stores data items one after another in sequence. The keywords used in manipulating a sequential file include Output, Input, Write, Read and Close. Table 5.15 gives a summary of the keywords used in manipulating sequential files.

Table 5.15

Keyword	*Description*
Output	Data is written on the storage media at the beginning of the file, overwriting any existing data.
Input	Opens the sequential file for reading if data is to be retrieved.
Append	Data is added at the end of the last item in a file.
Write	Data is output into the sequential file.
Read	Reads data from the sequential file.
Close	Terminates the processing of a disk file and writes the last partially filled buffer onto the storage media.
EOF(Filenumber)	*End of File* marker. When reading data from a disk, EOF tells the program when to stop. Attempt to read past the EOF causes a run-time error.

The following statements are very important when working with this type of files:

(a) Open *filename* For Output As #n

(b) Open *filename* For Input As #n

(c) Close #n

where *n* refers to the file number (n is usually an integer value).

Manipulating data in a sequential file

To manipulate a text file you can use the tools that corne with Visual Basic such as the common dialog box. For example, to open a file for input, use the showOpen method of the common dialog box. The skeletal code below shows that when the user enters the name of the file in the open dialog box, it is assigned to a string variable TargetFile ready for input.

```
Private Sub mnuOpen _Click
Dim TargetFile As String, x As Integer, item As string
 CommonDialogl.ShowOpen
TargetFile= CommonDialog I.FileName
Open TargetFile For Input As #1
x= 0
Do Until EOF(I)
    input # 1, item
     'Write a code that reads each data
     'Item from the TargetFile until the end-of file-condition is true
    x=x+l
Loop
```

```
Close #1
End Sub
```

Likewise, you can write data to a new file by operating it in Output mode as shown in the skeletal code listing below. Note that the file number used is #2 to avoid overwriting the first file.

```
Private Sub mnuSaveAs Click
Dim NewFile As String, x As Integer, item As string
CommonDialog 1.ShowSave
NewFile= CommonDialogl.FileName
Open NewFile For Output As #2
n= 30
For x = 1 TO n
    'Write a code that writes each data item into the NewFile Next x
Close #2
End Sub
```

When you write to a file, all the original data in the file is lost. To avoid losing the content of a file open it in Append mode as shown in the skeletal code below.

```
Private Sub mnuSaveAs Click
Dim TargetFile As String, x As Integer, n As Integer
CommonDialog 1.ShowSave
TargetFile= CommonDialog.FileName
Open TargetFile For Append As #1
n= 20
For x = 1 TO n
    'write each data item at the end of an existing file Next x
Close #1
End Sub
```

Project 5.8

Using Visual Basic programming, develop a simple sequential file processing program that would read data from a file called *ReaderFile.txt* and write the contents to another file called *Receive. txt* . The system should be menu driven. The project should use the Open and Save as dialog boxes to access the two files on the disk.

To create the project, proceed as follows:

1. Create a new project and save the default fonn as *sequentialForm.* Save the project as *SequentialFiles. vbp*
2. Create a File menu having menu commands as shown in Figure 5.27 below: i.e. Click the tools then menu editor command.
3. Place the Common dialog control, a combo box and a label on the form.
4. Double click the form then add the code listing below.

```
    Private Sub Form_Load()
    Form I. Caption = "Text File I"
    Labell. Caption = "Learning to work with text files"
    mnuSave As. Enabled = False
    Combo 1. Text = " "
    End Sub
```

5. Double click the Open command and type the code below:

```
 Private Sub mnuOpen – Click()
Dim TargetFile As String, Item As String
Dim x As Integer
CommonDialog I.Filter = "Text files (*. txt)/*. txt" 'open only text files
CommonDialog1.ShowOpen
TargetFile = CommonDialog1.FileName
Open TargetFile For Input As #1
x= 0
Do Until EOF(1)
 Input #1, Item
   ComboI.List(x) = Item  'Assign items to the list property of combo box
ComboI.Tag = x 'Assign the index x to the tag property that represent the number of items read from the file.
   x= x+ 1
Loop
Combo 1. Text = "Contents"
Combo I.Locked = True
mnuSave As.Enabled = True     'enable command after file open
Close #1
End Sub
Private Sub mnuSave _As – Click()
Dim NewTargetFile As String
Dim x As Integer, m As Integer
CommonDialogI.Filter = "Textfiles (*.txt)/*.txt"
CommonDialog I.ShowSave
NewTargetFile = CommonDialogI.FileName
Open NewTargetFile For Output As #2
n = Val (Combo 1. Tag)             'Assign the value of the tag to n
For x = 0 To n
      Print #2, ComboI.List(x)
Next x
Close #2
End Sub
```

Explanations

1.on running the program, a form is displayed on which is a combo box and a file menu.
2. On clicking the open command in the File menu, the open dialog box is displayed from which the user can select or type the file name.The file is assigned to a variable TargetFile.
3. Minimise the Visual Basic IDE Window and start a text editor program such as Notepad. In the window, write the text: we are the world.
4. After saving the text file, close the program and now return to your project. Click File then Open on your running project. In the Open dialog box, locate your text file and open it. Notice now that the words "We are the World" are displayed in the Combo box as shown in Figure 5.28. You can now use the Save_As command to save the file elsewhere with another name.

NB: If we alter the text and separate the words in the sentence with commas or quotes each word will be read into the computer separately. In such a case the words will be displayed in a drop down list in the combo box.

5. The reason why the text is displayed in the combo box is because we used the Print statement for output and directed the output to the combo box. However, to write the content into the file, use the Write #n statement instead of Print #n.

Project 5.9

Laura, a proprietor of *Laura Refreshment Cafe* has requested you to create a program that stores her products in a sequential file. The data entry form should display beverages in a combo box while juice flavours should be displayed in a list box. These two controls are populated with data stored in a sequential file. Using a command button, the user should be able to add more beverages in the combo box.
To create the project, proceed as follows:

1. Create a new project with one form and add appropriate controls on the form as shown in Figure 5.29.

2. Display each control properties window and set the properties of each control such as:
   (a) Name and caption of add beverages command button.
   (b) A combo box with items such as Tea, Coffee, Cocoa, Beans etc.
   (c) A List box with items such as Orange, Passim)., Mango etc.
   (d) A menu bar with File and Edit Menus. The commands in the
   File menu include Save and Exit while those in the Edit menu includes Delete an item, Clear the list and add an item

3, Write a code for each event procedure as shown by the listing below.
   (a) Double click the Add Beverages command button and type the code below:

```
        Option Explicit
        Dim mbUnsaved As Boolean
        Private Sub cmdAddBevarage – Click()
        'Add new Beverage If cboBeverages.Text <> "" Then
    With cboBeverages
    .Add/tem cboBeverages
    .Text= ""
  End With
  mbUnsaved = True
Else
  MsgBox "Enter the name of Beverage", vbExclamation, "Missing Data"
End If cboBeverages.SetFocus
End Sub
```

(b) Double click the form to open the code editor window and enter the code listing below:

```
    Private Sub Form_Load()            .
     'Load the list
    Dim stBeverages As String
    On Error GoTo HandleErrors 'Turn on the error handling routine
    Open "a:BevFile.Dat" For Input As #1
```

```
      Do Until EOF(1)
        Input #1, stBeverages
      cboBeverages.AddItem stBeverages
      Loop
      Close #1
      Form Load exit:
  Exit Sub
  HandleErrors:
                              'The code below raises any type of error encountered during the file open
operation
Dim iResponse As Integer
Select Case Err.Number
Case 53, 76 ' Invalid path or file name
  iResponse = MsgBox("Create a new file?", vbYesNo + vb Question, "File Not Found") Display
a message box
If iResponse = vb Yes Then
  Resume form_Load _exit 'Exit the Procedure
Else
  mnuFileExit - Click 'Exit the program
End If
      Case 71
      iResponse = MsgBox("Drive not Ready. Retry?", .
      vbRetryCancel + vbQuestion, "Drive Error") display a message box
If iResponse = vbRetry Then
  Resume
Else
       mnuFileExit - Click 'Exit the program
End If

      Case Else
                                                          .
      Err.Raise Err display a message for any other type of error
      End Select
End Sub
Private Sub mnuEditAdd – Click()
 'Add a new beverage list cmdAddBevarage _Click
End Sub
```

(c) Double click the Add command from Edit menu and type the code below:

```
      Private Sub mnuEditdelete – Click()
      'Delete a selected item If cboBeverages.ListIndex <> -1 Then
          cboBeverages.RemoveItem cboBeverages.ListIndex
          mbUnsaved = True
          Else
          MsgBox "Select an item you want to delete", vbInformation, - I
"Select an Item"
 End If
End Sub
```
(d) Double click the Exit command in the file menu and type the code listing below:

```
    Private Sub mnuFileExit – Click()
     'Exit from the program and save changes
    Dim iResponse As Integer
    If mbUnsaved = True Then
        iResponse = MsgBox("Save changes?”,vb YesNo + vbQuestion, "Save Changes")
        If iResponse = vb Yes Then
         mnuFileSave Click
        End If
    End If
    End
    End Sub
```
(e) Double click the save command in the file menu and type the code listing below:

```
    Private Sub mnuFileSave – Click()
     'Save the combo box content into a sequential file
     Dim iIndex As Integer
    Dim iMax As Integer
    Open "a:BevFile.Dat" For Output As #1
    iMax = cboBeverages.ListCount - 1
    For iIndex = 0 To iMax
       Write #1, cboBeverages.List(iIndex)
    Next iIndex
    Close #1
    mbUnsaved = False
    End Sub
```
Explanation

1. *On* running the program, the form is loaded and you can add a beverage item in the combo box as shown in Figure 5.30.

3. When the user clicks the save command on the File menu, the Beverages list is saved on the floppy disk. You can view the items using Microsoft Windows NotePad or any Text Editor. Below is a list of items saved in the file *BevFile.Dat.*

```
"Chocolate"
"Tea"
"Coffee"
"Milo"
"Plain Coffee"
"Soya"      I
"Cocoa"
```

4. On *error* statement have been used in the Form Load procedure. This code ensures that the program does not halt in case the user forgets to insert a floppy disk. On error statement and the err objects are therefore used to trap run-time errors. The correct syntax of the On Error statement is: I

On Error GoTo LineLabel e.g. On Error GoTo ErrorHandler I

The lineLabel refers to the error handling code that begins with a line label such as *HandleErrors:*

The *resume* statement tells the program to resume back to the statement that caused the error. *Resume Next* option can also be used I to tells the program to execute the statement immediately following the line that caused the error.

The *Err object* holds information about an error that has occurred. The name of the application that caused the error is stored in the *source* property. The *Number* property contains error numbers ranging from 0 to 65,535. Examples of *Err.Number* are shown in Table 5.16.

Table 5.16

Number	*Err.Description*	*Number*	*Err.Description*
7	Out of memory	58	File already Exists
11	Division by zero	61	Disk full
13	Type mismatch	71	Disk not ready
53	File not found	76	Path not found

You can use the *raise* method to set the error code, effectively turning on an error. This is necessary if the error that has occurred is not the one of the anticipated. The statement *Err.Raise Err* in the form_load procedure therefore tells the program to display an error message for any other unhandled error.

Random access files

Random access files consists of fixed length records each assigned a unique identification number. The keywords use in random file manipulation are summarised in Table 5.17 below:

Table 5.17

Keyword	Description
Random	Records can be input or output and accessed in any, order.
Get # *FileNumber*	Reads data from a random disk file and place data into the record name variable. The variable must be a user-defined data type e.g. a record variable.
Put # *FileNumber*	Writes the content of a record on the disk. The record number determines the relative location within the file. If the record number is omitted, the record will be placed in the next location from the last Get or Put operation.
Len	Len refers to the length of a single record. For example *Len(EmployeeRecord)* retUrns the size of the employeeRecord in bytes.

LOF(FileNumber)	Rather than using EOF, LOF is used to find the end of a random file. This function returns the size of the file in bytes. *FileNumber* is the file number from the currently open file. To determine the highest record number, divide the return value of LOF by the size of one record. *For Example* *Number _of_records = LOF(1)/ Len (EmployeeRecord).*
Seek(FileNumber)	Returns the current location of the file pointer. For a sequential file the current byte number is returned while in randof!! files, seek returns the position of the next record. For example *NextRecord = seek* (1)

To open a random file for input or output use the syntax: The following statement:
Open *filename* For Random As #n Len = *record length*
Reading records from a random file
Records are read one at a time using the following procedure
Get #n,record number, data item
Sometimes, you may ignore the record number but instead put two commas I.e.
Get #n"data item
The computer in this case will look for the record following the last'Get operation.
Writing to a random file
The statement below can be used to write records to a file.
Put #n,record number, data item or put #n"data item
The latter statement would write a record next to the last Put operation.
NB: Random files will accept data records only.

Project 5.10

You have been requested to develop a program that can store employees contact details. Create a project that has two forms namely the *main form* and a *data form* that can be used to manipulate employees records and store them in a random file. The forms should have the following controls:

Main form
1. A List box where the names of the employees are displayed.
2. Command buttons for Adding, Deleting, Updating and browsing employee's records.
3. Menu bar with File and View menus.

Data form
1. Labels and Text boxes for LastName, FirstName, Address and City/Town
2. Navigation buttons i.e. First, Next, Previous and Last.
3. OK and Cancel buttons.

To create the project, proceed as follows:

1. Create a project that has two forms namely the *main form* and a *data form* and add controls as shown in Figure 5.31 (a) and (b).

2. Open the properties window and set the properties of each control such as:
   (a) Name and caption of each form controls

(b) Command buttons for both forms.
(c) A combo box with items such as Tea, Coffee, Cocoa Soya etc.

3. Create a menu bar with *File* and *View* menus. The File menu should include at least one command e.g. *Exit* while the View menu should have at least *View Records command.*

4. Write codes for each event procedure and standard procedure as shown by the listing below.
   Standard(Global) Procedure
   Enter this code in a standard module code editor window.

```
Option Explicit
Type employeesRecord            'Define a record with five fields.
       stLastName As String * 15
       stFirstName As String * 15
       stAddress As String * 10
       stCity As String * 12
       stDeleted As String * 1
End Type                        'End of record defination
Public gEmployee As employeesRecord
Public gstFileAction As String
Main form Procedures
Private Sub cmdAddNew – Click()
  'Display a blank record in order to add a new employee
  gstFileAction = "A"
frmFile.Show vbModal
End Sub

Private Sub cmdBrowse – Click()
'Browse the File
    gstFileAction = "B"
frmFile.Show vbModal End Sub
Private Sub cmddelete – Click()
    'Delete an employee from the list
    iflstEmployee.ListIndex <> -1 Then
    gstFileAction = "D"
    frmFile.Show vbModal
    Else
    MsgBox "Select a record to be deleted!", vbOKOnly + vbInformation, "Delete!"
End if
End Sub
Private Sub cmdUpdate - ClickO
 'Update an employee record
    iflstEmployee.ListIndex <> -1 Then
    gstFileAction = "u"
    frmFile.Show vbModal
    Else
    MsgBox "Select a record to update", vbOKOnly + vbInformation, "Update"
    End if
    End Sub
Private Sub Form_Load()
```

```
'Store the details of the list box in a random file
 Dim iIndex As Integer
Dim iResponse As Integer
On Error GoTo HandleErrors
Open "a:Employee.Dat" For Random As #1 Len = Len(gEmployee)
 IfLOF(l) / Len (gEmployee) > 0 Then 'Iffile not empty
  For ilndex = 1 To LOF(1) / Len (gEmployee)
  Get #1, ilndex, gEmployee
  If gEmployee.stDeleted <> "Y" Then
  AddtoList (iIndex)
  End If
  Next iIndex
Else
iResponse = MsgBox("Create a new file?", vb YesNo + vb Question, "File Not Found")
If iResponse = vbNo Then
mnuFileExit _Click 'Exit the program
End If
End If
form_Load - exit:
Exit Sub
HandleErrors:
If Err.Number = 71 Then
   iResponse = MsgBox("Drive not Ready. Retry?", vbRetryCancel + vb Question, "Drive
Error")
   If iResponse = vbReiry Then
   Resume
      Else
      mnuFileExit _Click 'Exit the program
     End If
Else
   On Error GoTo 0 'Turn error trap off
   End If
   End Sub
Private Sub mnuFileExit – Click()
 'End the program
Close #1
End
End Sub
Public Sub AddtoList(iIndex As Integer)
 'Add an employee in the list box
Dim stName As String
stName = Trim(gEmployee.stLastName) & ", "& gEmployee.stFirstName
IstEmployee.AddItem stName
IstEmployee.ItemData(lstEmployee.NewIndex) = iIndex
End Sub
Private Sub mnu ViewEmployee – Click()
```

```
    'Display the employees file
    gstFileAction = "B"      'Browse
   frmFile.Show vbModal
    End Sub
   Data form Procedures
Option Explicit
Dim miIndex As Integer
Private Sub cmdCancel- Click()
    'Switch to the main form
   frmFile.Hide
    End Sub
   Private Sub DisplayRecords()
    'Display record fields in the form text controls
     txtLastName. Text = gEmployee.stLastName
     txtFirstName. Text = gEmployee.stFirstName
    txtAddress. Text = gEmployee.stAddress
    txtCity. Text = gEmployee.stCity
End Sub
Private Sub SaveRecords()
gEmployee.stLastName = txtLastName. Text
gEmployee.stFirstName = txtFirstName. Text
gEmployee.stAddress = txtAddress. Text
gEmployee.stCity = txtCity. Text
End Sub
Private Sub cmdFirst – Click()
     'Display the first record
   frmMain!lstEmployee.ListIndex = 0
   Readrecords
End Sub
Private Sub cmdNext_Click()
    'Display the next record
    If frmMain! IstEmployee.ListIndex <
  frmMain!lstEmployee.ListCount -1 Then
  jrmMain!lstEmployee.Listlndex = jrmMain!lstEmployee.Listlndex + 1
  Readrecords
  Else
  cmdFirst Click
  End If
End Sub
Private Sub cmdLast – Click()
     'Display the last record
   frmMainflstEmployee.Listlndex =
   frmMain!lstEmployee.ListCount -1
Readrecords
End sub
Private Sub cmdOK_Click()
```

```
    'Select the action to perform then click OK
    Dim stName As String
    Select Case gstFileAction
    Case "A"
         milndex = LOF(1) / Len (gEmployee) + 1
        gEmployee.stDeleted = "N"
        SaveRecords
        WriteRecords
        frmMain.AddtoList (milndex)
frmMain!lstEmployee.ListIndex = frmMain!lstEmployee.Newlndex
Case "D"
 miIndex =frmMain!lstEmplayee.ItemData(frmMain!lstEmployeeListIndex)
 gEmployee.stDeleted = "Y"
WriteRecords
frmMain!lstEmployee.RemoveltemfrmMain! IstEmployee.Listlndex
  Case "U"
       SaveRecords
milndex= frmMain!lstEmployee.ItemData(frmMain!lstEmplayeeListIndex)
       WriteRecords
 frmMain!lstEmployee.Removeltem frmMain! IstEmployee.Listlndex
frmMain.AddtoList (miIndex)
Case "B"
End Select
 frmFile.Hide 'Return to the main form
 End Sub
 Private Sub ClearTextBoxes()
    txtLastName.Text = ""
    txtFirstName. Text = ""
    txtAddress. Text = ""
   txtCity. Text = ""
   End Sub
    Private Sub NavigationEnabled()
   cmdFirst.Enabled = True
   cmdNext.Enabled = True
    cmdPrevious.Enabled = True
   cmdLast.Enabled = True End Sub
Private Sub NavigationDisabled()
    cmdFirst.Enabled = False
    cmdNext.Enabled = False
    cmdPrevious.Enabled = False
  cmdLast.Enabled = False
End Sub
Private Sub cmdPrevious – Click()
   If frmMain!lstEmployee.Listlndex < 1 Then
cmdLast Click
    Else
    frmMain!lstEmployee.ListIndex = .frmMain!lstEmployee.ListIndex - 1
    Readrecords
    End If
    End Sub
    Private Sub Readrecords()
    'Read data from a random file
    miIndex= frmMain!lstEmployee.itemData(frmMain!lstEmployee.ListIndex}
   Get #1, miIndex, gEmployee
   DisplayRecords
   End Sub
    Private Sub WriteRecords()
   'Save data to a disk
    Put #1, miIndex, gEmployee
    End Sub
   Private Sub Form_Activate()
   'Activate the data form and set focus to the LastName field
   Select Case gstFileAction
Case "A"
ClearTextBoxes
NavigationDisabled
txtLastName.SetFocus
Case "D"
Readrecords
  NavigationDisabled
Case "U"
 Readrecords
NavigationEnabled
Case "B"
  NavigationEnabled
  cmdFirst Click
  End Select
End Sub
```
Explanation

1. On running the program, the main form is loaded that shows a list of employees as shown in Figure 5.32.
2. When the user clicks the Add New button, a blank data form is displayed with focus set on the LastName text box. (See Figure 5.33). If you add a new employee record and then click the OK button, the employee's name is added into the list box and saved into random file.
3. On clicking the Delete, Update or Browse buttons, the records are displayed on the form for the user to perform the appropriate action. Figure 5.34 (a) shows a record that is just about to be deleted while Figure 5.34 (b) shows the main form display after deleting the selected record.

Binary files

A binary file is used to store graphics, sound or compiled files. Applications that make use of binary files are complicated and beyond the scope of this book. However, to manipulate a binary file the keywords Put, Get, EOF, Len, LOF, Seek, Loc (pointer location) are used.

To open a binary file we use the statement; *OpenfileName* for *Binary As #n.* Below is a skeletal listing of a Visual Basic program that read and writes the same data into a binary file.

```
Private Sub mnuCopy – Click()
Dim FileLength As Integer, Index As Integer
Dim FileName As String
Dim FileByteO As Byte
FileName = .........'find out the file name
Open FileName For Binary As #1
FileLength = LOF(1)
ReDim FileByte(Filelength)
For index = 1 To FileLength                'read the file
 Get # 1 "FileByte(Index)                 'the second field has no value
      Next index
For Index = 1 To FileLength
Put #1, Index, FileByte(Index)            'the second field has a value
       Next Index
Close #1
End Sub
```

Explanation

The Get #n statement start reading the first byte in the file and then reads each successive byte until the end of the file. The Put #n statement has a value hence it will overwrite the contents of the file it is writing to. To avoid this, leave index empty so that the write operation simply appends the records to the end of the file.

1. What is an array subscript?
2. Differentiate between a record and' an array.
3. Differentiate between a sequential file and a random access file.

4. State the function of each of the following file manipulation commands.

(a) Input #1 (b) Get#n (c) Put #n (d) Qpen
5. Differentiate between (a) EOF and LOF

6. An employee file has the following fields: First name, Second name, Employment number, Date of employment and Date of birth. Using the type declaration, create a user defined data type that will store ail this data for one employee.
7. Declare a variable that can store the user defined data type in question 6 above.
8. Assuming the user wants to access the DateofEmployment field in a record variable'employeeDetails, write a statement that can do this.

Linking Visual Basic forms to a database

In Visual Basic, it is possible to access an existing database. This is done by adding data controls on a form. For example assuming a database called library exists and it has a table called students, to access this table in Visual Basic, proceed as follows:

1. Open your Visual Basic project. Click the Project Explorer icon on the toolbar to display the explorer window. Double click Form1(Form1.frm) in the project explorer window to display the blank form as in Figure 5.35.

2. If the properties window is not displayed, right click the form then select Properties from the shortcut menu. Change the properties as indicated below:

Property	*Value*
Name	Studentform
Caption	Enter or View Students

3. Double click the Data control tool on the toolbox. A data control is placed on the form automatically. Now move and resize it as you wish. Change the following properties of the data control:

Property	*Value*
Name	StudentFormData
Caption	ViewStudents
Connect	Access 98/2000/2003 (Depends on the database type and version installed in your computer).
DatabaseN am	Provide the full directory path to the database by clicking the... icon then browse to the database.
RecordSourct	STUDENTS (select this table from a drop down menu)

The last two properties link your form to the database that has the data source records. For example, in our case, the data source is the students table found in the Library database.

4. To view the fields of each record, add text boxes in the form that would accommodate all fields from each record. To link each text to the data source, double click the text box and then change the following properties for each text box.

Textbox 1

Property	value
Name	Studentregistrationtxt
Data source	Studentformdata (remember this is the name you gave to the data control on the form)
Datafield	Registration number (select from dropdown menu)
Text	Reg no

Textbox 2

Property	*Value*
Name	Studentfirstnametxt
DataSource	Studentformdata
Datafield	First name

Repeat the above for all the available textboxes then save the project. Figure 5.36 (a) shows three textboxes linked to the STUDENT table.

5. Run the project to display the content of the underlying table. Figure 5.36 (b) shows the sample record from the underlying table. Notice that the fields on the form do not have labels. It is the high time we add labels and command buttons.

To add labels proceed as follows:

(a) Double click the label control on the toolbox. A label is placed on the form.

(b) Drag the label and place it above the Registration number field on the form.
(c) Change the label name and caption appropriately.
(d) Set the labels properties appropriately as shown in the table below.
Labels 1

Name	*Caption*
Labelregno	Registration no:
LabelFirstName	First name:
LabelSecName	Second name:

To add a command button proceed as follows:
(a) Double click the button from the toolbox.
(b) Change the name and caption of the command button.

Name	*Caption*
EnterNewRecord	New record
SaveNewRecord	Save

I
(c) Change the buttons properties as indicated below
Your form should now look as shown in Figure 5.37.

Project 5.11

Create a new project called *SchoolLibrary* that can be used to access a database called *Biblio.mdb* that conies with Visual Basic.
To create the project, proceed as follows:
1. Start Microsoft Access and locate for the Biblio database. The path to the database may be *C:IProgram FileslMicrosoft Visual Studio\VB98\Biblio. mdb.* Figure 5.39 shows the tables in the database.
2. Click the queries tab and open *All Titles* in design view. Note that the query has four fields namely: Titles, ISBN, Author and Year Published.

3. Create a project that contains one form, large enough to hold the four fields. Click on the Data Control tool and draw a data control at the bottom of the form as shown in Figure 5.40.
4. Add labels and text boxes that will be used as bound controls for the fields from the query. Set *Name, Caption* and *Text* Properties for the labels and textboxes accordingly.
5. Change the Data control properties as follows:

(a) Set the Name property to *datBooks* and Caption to *Display a book.*

(b) Make sure the *connect* property is set to *Access.*

(c) Click the DatabaseName settings button (...) to display database name dialog box. From the dialog box, locate the database called Biblio using the path *C: IProgram Files \Microsoft Visual*

Studio\VB98\Biblio.mdb

6. Scroll to RecordSource property and click the down arrow. From the list of tables and queries displayed, select All Titles. Figure 5.41 shows a section of the Data control properties settings.
7. Click each textbox and set the DataFie1d appropriately. For example, click the txtISBN, then from the DataFie1d property, select ISBN.
8. Write the code listed below for the command Close button.

```
Private sub cmdClose – click()
, Exit from the project
        End
        End Sub
```

9. Save the form as books and the project as *VBBiblio.* Figure 5.42 shows the complete form in design view.
10. Run the project. Notice that the form bound controls are populated with data from the *All Titles* query as shown in Figure 5.43. Click the navigation arrows of the data control to view other records.

Using the Recordset object to manipulate a database

When you set the record source property of a data control to the name of a table or query, you are defining a new object called a *Recordset.* The recordset has its own set of properties and methods which you can use to move from record to record. Below is a summary of some methods used with Recordset.

Method	*Description*
AddNew	Clears the bound controls for a new record entry
Update	Beginning of file. Updates the underlying table or query by adding the new field.
Delete	End of file. Used for deleting the currently selected record.
BOF	Used to check for the beginning of a file.
EOF	Used to test the end of a file.
Move	Used to navigate through the database. Other Move methods includes *MoveNext, MoveLast* and *MovePrevious*

Project 5.12

Using the Recordset object, modify the VBBiblio project by adding three more command buttons namely Add New, Save and Delete on the Books form. Using the three buttons, the user should be able to add a new record, automatically update the underlying data source or delete a selected record.

To create the project, proceed as follows:

1. Open VBBiblio project and add the three buttons as shown in Figure5.44.
2. Write the code for each of the buttons. Below is a sample code for the modified VBBiblio project.

```
Private Sub cmdAdd – Click()
 'Add a new record
 If cmdAdd. Caption = "&Add New" Then
     datBooks.Recordset.AddNew 'Clear the fields for new entry
     txtISBN.SetFocus
     DisableNavigations
     cmdSave.Enabled = True 'Enable the save button
     cmdAdd.Caption = "&Cancel" 'Allow a cancel option
Else
     datBooks.Recordset. Cancel Update
     EnableNavigations
      cmdSave.Enabled = False
      cmdAdd.Caption = "&Add New"
  End If
```

```
      End Sub
Private Sub DisableNavigations()
 cmdFirst.Enabled = False
cmdNext.Enabled = False
cmdPrevious.Enabled = False
cmdLast.Enabled = False
cmdDelete.Enabled = False
End Sub
Private Sub EnableNavigations()
 cmdFirst.Enabled = True
cmdNext.Enabled = True
cmdPrevious.Enabled = True
cmdLast.Enabled = True
cmdDelete.Enabled = True
End Sub
Private Sub
cmdClose – Click()      'Close the form
  End
End Sub
Private Sub cmdDelete_Click()         'Delete the selected record
  With datBooks.Recordset
  .Delete
  .MoveNext
If.EOF Then
      .MovePrevious
    If.BOF Then
    MsgBox " The record set is empty", vblnformation, "No
  Records"
      End If
    End If
 End With End Sub Private Sub cmdFirst – Click()
    'Move to the first record
    datBooks.Recordset.MoveFirst
End Sub
Private Sub cmdLast – Click()
   'Move to the Last record
datBooks.Recordset
.MoveLast
End Sub
Private Sub cmdNext_Click()    'Move to the next record
With datBooks.Recordset
        .MoveNext
    If.EOF Then
    .MoveFirst
End If
End With
End Sub
Private Sub cmdPrevious – Click()   'Move to the previous record
With datBooks.Recordset
        .MovePrevious
     If.BOF Then
        .MoveLast
    End If
  End With
End Sub
Private Sub cmdSave – Click()
   'Save the current record
   datBooks.Recordset.Update
   EnableNavigations
   cmdSave.Enabled = False
   cmdAdd.Caption = "&Add New"
End Sub
```

Explanation

On running the program, the form is loaded that shows each record from *All Titles query*. The user can navigate to view other records using the navigation buttons. This project also allows the user to add, save and delete a record as shown in Figure 5.45.

Practical activity 5.3

Using an existing database called Hospital, write a project that access the employees Table that has at least five fields i.e. First Name, LastName, Dateofbirth, Department and Status. The project form should have Navigation, Add, save and Delete buttons

1. Assuming you have a database containing the following fields:
   (a) Names
   (b) Contact numbers
   (c) Town
   Explain how you can access these fields in the database from Visual Basic environment.
2. Explain the difference between a data control and a databound control.
3. Explain how you would link up a form in Visual Basic to a database called Biblio that comes with VisualBasic located in a folder VB98. VB98 is accessed through the path C:/Program File/Microsoft Visual Studio VB98.

Creating a Visual Basic executable file

You can convert your project into an executable (.EXE) file that can be run on Microsoft Windows desktop. The EXE file contains the information for all your project files including the form files and modules. After you create the EXE file, the other files are not affected. However anytime you make changes to the source code, you must recreate the EXE file.
To create the EXE file Proceed as follows:
1. From the:Filemenu, select *Make xxxx.exewhere* xxxx stands for the currently open project.
2. In the make project dialog box, type the name of your executable file then click OK.
3. You can then place your EXE file in the programs menu or on the desktop.

Conclusion

Therefore with knowledge in Visual Basic programming and system development discussed in Chapter 4, you can be able to create versatile information systems. This being an introduction there is a lot in Visual Basic programming. You can learn more on Visual Basic Programming from *Microsoft Visual studio on-line help* known as *MSDN* collection.

APPENDIX I
Introduction to database design

Most systems will involve storage of data in a database and then having different users accessing the data in their own unique way. Sometimes the data may be stored in a file that is created on the disk. This appendix demonstrates how to construct system files or tables, using Microsoft Access and finally automate your database to make it a fully-pledged application.
Remember that in system analysis, we said that for a library system, we will need to store data about books and students as entities of the system. These records have to be stored in a database. These entities have a *relationship* which needs to be defined during database design. For example, because a student can borrow many books, this can be represented in the database design as a "one-to-many relationship". Figure App. 1.1 represents this relationship using a simple relationship diagram. The line that joins the two entities divides into three on the Books side to show that one student can borrow many books.

Other relationships that are possible between entities are "one-to-one" and "many-to-many". In Figure App. 1.2. for example, one student can only have one registration number.

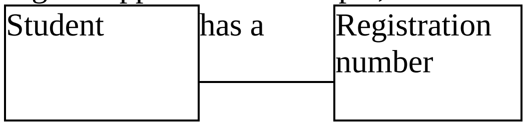


Fig. App. 1.2: One-to-one relationship

Similarly, in a school environment, one course unit can be done by many students and one student can do many course units (Figure App. 1.3).

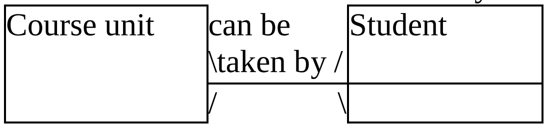


Fig. App. 1.3: Many-to-many relationship

It is therefore possible to have a database design that has all the three entity relationships. However, when designing your database, best practice demands one to develop one-to-many relationships between entities. The diagram that shows the logical relationship between data entities is called the *entity relationship diagram (ERD).*

Defining attributes

Attributes need to be identified for each of the entities identified in the entity relationship diagram. The attributes will become the fields of the database table that will store data for each entity. A complete set of
attributes in a table will form a record. Each entity will have one attribute that will hold a unique value for each of the records entered in the database. This unique attribute is called the *key attribute* or the *primary key (PK).* To create a relationship between two tables, include the primary key of one table in the second table i.e. create a field in the second table that has the same data type as the primary key of the first table. This field created in the second table is called the *foreign key* (FK). Table App. 1.1 shows the attributes for two entities in a library management system.

Table App. 1.1

Entities and their attributes	
Entity	Attribute
Books	Book number (PK) ISBN number Author Publisher Purchase price
Student	Registration number First name Middle name. Class Stream

A careful study of the two tables will reveal that some of the attributes need to be separated and used to develop other entities. This is because they will encourage duplicate storage of data in the database. For example, one "Publisher" may have published many books. Hence, if you have one hundred different books in the library done by the same publisher, you will have to enter the same name a hundred times in the database!

Table App.I.2 shows this anomaly. This data duplication is not allowed and hence we can create an entity called "Publishers" that will have the name of a publisher entered once in the database then allow you to keep on referring to it when the need arises. The process of trying to eliminate storage of duplicate values in the database is called *normalization.* It is governed by some rules that are outside the scope of this book. However, the simple principle is that you try to separate off any attributes that may eventually carry repeating values in the database to form their own tables.

Table App. 1.2

ISBN	Title	Publisher	Date published	Date issued	Return date
09625063xx 08765432xx	GIS Data Form 1 Geog.	Jets book Jets book	2000 2001	1/3/2005 3/4/2005	8/3/2005 10/4/2005

To avoid repetition, Table App. 1.2 can be broken down into two, one for the books and another for publishers as shown in Table App. 1.3.

Table App. 1.3

Entities and their attributes	
Entity	Attribute
Book	Book Number (PK) ISBN Title. Date of publication. Year published Purchase price Student number (FK) Publisher **ID** (FK)

Student	Registration number (PK). First name. Middle name. Class Stream
Publisher	Publisher **ID** (PK). Publisher name. Address City Phone number

The entity relationship diagram of the system will now look as shown in Figure App. lA.
This entity relationship diagram shows that a student can borrow many books. Likewise a publisher can publish many books.

Creating the database

This section presumes that you have covered the Form 2 syllabus on databases. To create a database, proceed as follows:

1. From the start menu, select Microsoft Access. The dialog box of Figure.App. 1.5 appears.

Fig.App. 1.5:

2. Select *BlankAccess Database* option then click the *OK* button. The *File New Database* dialog box appears. In the File name box, type the name of the new database to be created as shown in Figure App. 1.6.

3. In the *Save in* box select the folder where you want to save your database, then click the Create Database button.
4. The database as shown in Figure App. 1.7 appears on the screen allowing you to create the various tables.

Defining a file/table structure

To define the file/table structure:

1. Click the *Tables* tab in the database window and then double click *Create table in Design view* command.
2. Create the three tables i.e. students, books and publisher using the attributes defined earlier in Table App. 1.3. Figure App. 1.8 shows a students table in design view.

Setting the primary key and foreign key

A primary key can be defined as a unique field that identifies each record in a database table. This means that the primary key in a database table holds unique values for each record and cannot accept duplicate, null or empty values. For example, if one student has the registration number 2000, no other student can be assigned the same number and a value must be entered into the field before proceeding to another record.
On the other hand, the foreign key is included as an attribute of an entity in order to create a relationship with another entity in the database. For example, the *Student registration number*

field in the BOOKS entity creates a relationship between the STUDENT entity and the BOOKS entity. The entity with the foreign key is usually on the many side of a relationship. Values in a foreign key field of a table in a database are unique but are allowed to repeat as long as they appear in the primary key of the primary table.
To define table relationships:

1. From the tools menu, click *relationship.* Show table dialog box appears as shown in Figure App. 1.9.
2. Select the tables and click *add* so that they are displayed on the relationship window.
3. To create a relationship, drag the primary key of the parent table to the foreign key of the "child" table. For example to create a relationship between Students table and Books table, drag the Fig. App. 1.10: Defining relationships registration number of the students table to the registration number of the books table. Eventually, you will get a relationship such as the one shown in Figure App. 1.10.

4. Enforce referential integrity: Referential integrity governs the nature of records in a "one-to-many relationship". This means that all foreign keys in the child table must have a matching record in the parent table. Therefore you cannot add a record in the child table if it does not have a related record in the parent table.

   Check the 'Cascade Update Related Fields' if you want your database to update the child tables once the parent record is updated. This means that if the student's registration number is changed, all the records in the related tables will automatically be updated. FigureApp. 1.11 shows an edit relationship for the two tables i.e. students and books.

Once you have created your database tables and related them accordingly, you can then enter data into the tables using forms which you create. Since database forms design was covered in Form 2 the rest of this appendix illustrates only data entry into the system.

Figure App. 1.15 shows a sample report generated from the database showing a summary report indicating which student has which book. The report has been generated from a query that is created using the three related tables.

Automating your database application
Once you finish creating the basic database objects i.e. tables, forms, reports and queries, you may wish to automate operations such as data entry, printing reports and making the user interface more user friendly.
In Microsoft Access, some of the tools used to automate a database are:

1. *Switchboard:* This is a special purpose form used to automatically access to other database objects such as forms and reports.

2. *Macros:* This is a set of one or more actions used to automate commonly performed tasks such as opening and closing forms and printing reports.
3. *Visual Basic for application (VBA) module:* The module object in Microsoft Access is a program coding widow for a subset of Visual Basic programming language called *Visual Basic for Application* (VBA). VBA is similar to Visual Basic only that you cannot create a new program (project) within.

Creating a switchboard To create a switchboard;

1. From the Tools menu, point to Database utilities then click Switchboard manager.
2. If no switchboards are available, Microsoft Access will prompt you whether you want to create one. Click *Yes.*
3. From the switchboard manager dialog box click Edit.
4. From the Edit switchboard dialog box, click Edit. The Edit switchboard Item dialog box appears in which you can specify the menu items to be included on the switchboard. Figure App. 1.16 shows an illustration of switchboard editing dialog box for the library system.
5. In the "Edit switchboard item" text box, enter the name of the menu. From the command list box, select the object to be accessed e.g.
"Open form in Edit mode". In the Form list box, select the name of the object e.g. StudentsForm user click *OK.*
6. Finally click Close buttons to close the switchboard dialog boxes. Figure App. 1.17 below shows a switchboard for the library management system.

Creating macros

To create a macro in Microsoft Access:

1. Click the Macros tab then New. The macros design grid appears.
2. In the macros design window, *Action column,* click the down arrow to select an action that you want to be executed directly every time you run the macro. For example, scroll downwards and click OpenForm.
3. In the lower part of the macro design window specify the action arguments. For example, for the action *OpenForm* select the name of the form e.g. *switchboard, view mode.*

Creating Visual Basic for application (VBA) module

Since in chapter 5 we introduced you to Visual Basic programming, you can transfer the same skills to coding using Visual Basic for Application. You will realise that the only difference between the two is that Microsoft Access modules are only limited to particular objects and that you can not create a new project using this. For more on Visual Basic for Application coding, you can get help that comes with Microsoft Access. However, it is important to note that Visual Basic for application is also available as an independent programming language which you can use just like Visual Basic.

There are two approaches for creating VBA codes:

1. Adding a code to form and report using event procedure. To add a code in this way, open the form or report in design view and then double click the form icon located at the meeting point of the vertical and the horizontal ruler.
2. From the objects properties list, click the events tab then click the build button from the resulting dialog box, choose Code Builder.

The coding window for the object you chose is displayed as shown in Figure App. 1.18.
Enter the VBA code between the private sub object name event and the end sub.

Securing your database from unauthorised access

There are a number of security options that can be used to set security. However the simplest methods is the use of a password.
To setup a database password in Microsoft Access,
1. Open the database file in exclusive mode as shown in Figure App. 1.19.
2. From Tools menu, point to Security then click Set password.
3. Enter the password in the password text box and re-enter the same password in verify box then click: OK

When a user opens the database, he/she will be prompted to enter the password as show in Figure App. 1.20.

Setting up the startup options

You can finally customise your application workplace by specifying the startup options. Start up options lets you customise your database in order to make it more presentable to the users. Such a database will always be starting as an application instead of loading it from Microsoft Access application window.
To set up the start up options for your application, from Tools menu, *click<start up* then specify the following

1. *Applications title:* The title that will appear on the title bar instead of the default Microsoft Access title.
2. *Application icon:* You can choose an icon that goes together with your application title.
3. *Menu bar:* You can specify the menu options that should appear when running the application.
4. *Display form:* Choose the form you want automatically displayed once you start the application e.g. the switchboard.
5. Uncheck "Display Database Window" to limit the users from viewing the objects created in your database. .

Conclusion

Although this Appendix has tried to open you to a lot of utilities, tools and commands you can use to come up with a fully fledged application, it is important to note that Microsoft Access has a lot of potential which would take hundreds of pages to explain. However, you can use this Appendix to help you explore further on skills and concepts that can help you develop information systems through customisation approach.

APPENDIX II

Coding schemes

Binary Coded Decimal (BCD)

Decimal number	0	1	2	3	4	5
Binary equivalent	0000	0001	0010	0011	0100	0101
Decimal number	6	7	8	9		
Binary equivalent	0110	0111	1000	1001		

Standard Binary Coded Decimal

Letter	Binary	Letter	Binary	Letter	Binary	Letter	Binary
A	11 000 1	H	111 000	0	10011 0	V	010101
B	11 00 10	I	111001	P	100111	W	010110
C	11 00 11	*J*	100001	Q	10 1000	X	010111
D	110100	K	100010	R	10100 I	Y	011000
E	110101	L	100011	S	010010	Z	011 00 1
F	110110	M	100100	T	010011		
G	110111	N	100101	U	010100		

ASCII and EBCDIC

Character	7 -bit ASCII	8-bit EBCDIC
0	0110000	1111 0000
1	0110001	1111 000 1
2	0110010	11110010
3	*0110011*	11110011
4	0110100	11110100
5	0110101	11110101
6	0110110	11110110
7	*0110111*	11110111
8	0111000	11111000
9	0111001	11111001

Character	7 -bit ASCII	8-bit EBCDIC
A	1000001	11000001
B	1000010	11000010
C	1000011	11000011
D	1000100	11000100
E	1000101	11000101
F	1000110	11000110
G	1000111	11000111
H	1001000	11001000
I	1001001	11001001
J	1001110	11010001
K	1001011	11010010
L	1001100	11010011
M	1001101	11010100
N	1001110	11010101
0	1001111	11010110
P	1010000	11010111

Q	1010001	11011000
R	1010010	11011001
S	1010011	11100010
T	1010100	11100011
U	1010101	11100100
V	1010110	11100101
W	1010111	11100110
X	1011000	11100111
Y	1011001	11101000
Z	1011010	11101001
Character	7 -bit ASCII	8-bit EBCDIC
blank	0100000	01000000
	0101110	01001011
,	0101100	01101011
+	0101011	01001110
-	0101010	01100000
/	0101111	01011100
=	0111101	01100001
(	0101000	01111110
$	0100100	01011011
)	0101001	01011101

NB: Alphabetic lower case characters have their symbolic representation too but not included in this scheme