

COMPUTER SCIENCE

For Class

9th - 10th

Part-2



Punjab Textbook Board, Lahore.

All rights are reserved with the Punjab Textbook Board, Lahore.

Prepared by: Punjab Textbook Board, Lahore.

Approved by: Federal Ministry of Education, Curriculum Wing, Islamabad.
vide its letter No. F.1-10/2005-Maths (Comp. Sc.) Dated July 8, 2006.

CONTENTS

Chapter #		Page #
1	PROBLEM SOLVING	1
2	DATA TYPES, ASSIGNMENT AND INPUT / OUTPUT STATEMENTS	13
3	CONTROL STRUCTURE	41
4	ARRAYS	53
5	SUB-PROGRAM AND FILE HANDLING	61
6	GRAPHICS IN BASIC	79
7	MICROSOFT WORD	89
*	GLOSSARY	120
*	INDEX	123

Authors

★ **Syed Zulqurnain Jaffery**

Assistant Professor
COMSATS Institute of Information Technology.
Sector H-8, Islamabad.

★ **Ms. Shahina Naz**

Head, Department of Computer Science,
Islamabad Model College for Girls,
F-10.2, Islamabad.

★ **Mr. Asif Ali Magsi**

Lecturer (Computer Science)
College for Boys, Islamabad.

Editor

★ **Mirza Mubasher Baig**

Research Associate
Department of Computer Science,
Lahore University of Management Sciences (LUMS), Lahore.

Supervision

★ **Mr. Mazhar Hayat**

Subject Specialist
Punjab Textbook Board, Lahore.

PROBLEM SOLVING

1.1 INTRODUCTION

We solve problems and make decisions everyday at home, at work, at play, and even at the general store. Some problems and decisions are very challenging, and require a lot of thought, emotion, and research. However, whatever the nature of the problem is, we always try to find multiple solutions so that we can have options to choose the best one.

1.2 PROBLEM-SOLVING METHOD

Problem-solving is a skill which can be developed by following a well organized approach. Programming is also a problem solving activity. If you are a good problem solver, you have the potential to become a good programmer. Problem solving methods are covered in many subject areas. Business students learn to solve problems with the related systems approach, while engineering and science students use the *engineering* and *scientific methods*. Programmers use the *software development method*.

The following steps can be followed to solve any kind of problem:

- | | |
|--------------------------------|--|
| (i) Problem identification | (ii) Specify requirements |
| (iii) Analyze the problem | (iv) Design algorithm and draw flowchart |
| (v) Write the program (Coding) | (vi) Test and Debug the program |
| (vii) Implement the program | (viii) Maintain and update the program |
| (ix) Document the program | |

1.2.1 Problem Identification

At this stage the problem being solved is observed carefully. Major areas of concern are identified and irrelevant information is filtered out. Suppose we want to develop a simple calculator. Our major concern is how basic arithmetic operations (addition, subtraction, multiplication and division) are performed? How result should be displayed? How input should be accepted etc.? We are not interested in how the Sine or Tan are calculated? How is the quadratic equation solved? And how are other mathematical operations performed? These are irrelevant to us, so we shall not bother about these. In this way, by filtering out irrelevant information we can concentrate on the actual problem.

1.2.2 Specify Requirements

Most of the users can not explain their exact software requirements. They are uncertain about what they want to do with the software. So they appear with a vague set of requirements in mind which may lead to a wrong solution. This stage demands to make clear the user's requirements so that a proper solution could be suggested. This stage involves the formation of a

requirements document which describes the features the system is expected to provide, the restrictions under which it must operate, and an abstract description of the software which provide a basis for design and implementation.

1.2.3 Analyze the Problem

At this stage the problem is decomposed into sub-problems. Rather on concentrating the bigger problem as a whole, we try to solve each sub-problem separately. This leads to a simple solution. This technique is known as **top-down design** (also called divide and conquer rule). Here we may ask certain questions to approach the right solution i.e.,

- (i) How many solutions are there to the given problem?
- (ii) Which one is the best solution?
- (iii) Can the problem be solved on computer?
- (iv) What are input and output?
- (v) How can the bigger problem be divided into sub-problems?

1.2.4 Design Algorithm and Draw Flowchart

Designing the algorithm requires to develop a finite list of steps to solve a problem. It is then verified that whether the algorithm solves the problem as intended or not. Writing algorithm is often the most difficult part of the problem-solving process. Most computer algorithms perform at least following three steps:

- (i) Get data (Input)
- (ii) Perform computation (Processing)
- (iii) Display results (Output)

Once the algorithm has been designed, it should be verified through desk checking. **Desk Checking** is an important part of algorithm design that is often over-looked. To desk check an algorithm, we must carefully perform each algorithm step just as a computer would do and verify that the algorithm works as desired. The time and effort can be saved by locating and rectifying algorithm errors at this stage.

A **program** is a set of instructions given to the computer to solve a particular problem. It is written in a programming language.

Desk Checking is the process of carefully observing the working of an algorithm, on the paper, for some test data. Algorithm is provided a variable set of input for which the output is examined.

Draw the Flowchart

After designing the algorithm, the next step is to draw a flowchart. Flowchart, in fact, maps the algorithm to a pictorial representation which helps in understanding the flow of control and

data in the algorithm. We shall discuss in detail the way the flowchart is drawn, later in this chapter.

1.2.5 Write the Program (Coding)

This step involves the conversion of an algorithm to a program, written in any programming language. For this purpose, the programmer must know the syntax of the programming language chosen.

*The grammatical rules of a programming language to write programs are referred to as **syntax** of that programming language.*

1.2.6 Test and Debug the Program

This stage requires evaluating the program to verify that it works as desired. Don't rely on just one test case. Run the program several times using different sets of data, making sure that it works correctly for every situation provided in the algorithm. If it is not producing desired results, then errors (bugs) must be pointed out and debugged. **Debugging** is the process of finding and removing errors in the program. There can be three types of programming errors; syntax errors, run time errors (execution errors), and logical errors.

***Debugging** is the process of finding and removing errors in the program.*

Syntax Errors

A syntax error occurs when the program violates one or more grammatical rules of the programming language. These errors are detected at compile time i.e., when the translator (compiler or interpreter) attempts to translate the program. There can be many reasons, for example trying to execute a wrong program statement or command such as typing PINT instead of PRINT statement; or trying to assign a value to a constant such as $5 = \text{count}$ etc.

Do You Know?

Division by zero is undefined.

Runtime Errors

A runtime error occurs when the program directs the computer to perform an illegal operation such as dividing a number by zero. Runtime errors are detected and displayed by the computer during the execution of a program. When a runtime error occurs, the computer will stop executing the program and may display a diagnostic message that helps in locating the error.

Logical Errors

Logic errors occur when a program follows a wrong logic. The translator (compiler or interpreter) does not report any error message for a logical error. These are the most difficult errors to locate. Logical errors can be identified by just looking at the wrong output of the program. Logic errors can only be detected by thoroughly testing the

program, observing all variables closely and testing each path of logic flow in the program.

1.2.7 Implement the Program

Once the program has been as tested thoroughly, it must be installed or put into operation at the site where it will be used. This is known as implementation of the program.

1.2.8 Maintain and Update the Program

Program maintenance is an ongoing process of upgrading the program to accommodate new hardware or software requirements, and introducing minor improvements. Essentially, it is the expansion, updating and improvement of a program after its installation. Regular maintenance is essential to the continued usefulness of a program. A proper maintenance depends on the existence of complete documentation.

1.2.9 Document the Program

Documentation is a detailed description of a program's algorithm, design, coding method, testing, and proper usage. Documentation is valuable for the users who rely upon the program on a day-to-day basis, and for the programmer who may be called on to modify or update it.

There are no universally accepted standards concerning what should be included in a program's documentation. Although its contents will vary somewhat depending on the complexity of the program, in general, comprehensive documentation consists of the following:

- A description of what the program is supposed to do (software requirement document).
- A description of the problem solution (the algorithm).
- A description of the program design, including any aids used (flowcharts, algorithms etc.).
- A description of the program's testing process, including the test data used and results obtained.
- A description of all corrections, modifications, and updates made to the program since it was put into operation.
- A *user manual* (user guide).

1.3 ALGORITHM

An *algorithm* is a finite set of steps which, if followed, accomplish a particular task. An algorithm must be clear, finite and effective. The simplest form of algorithm is *step-form algorithm* (like a to-do list). It consists of a sequence of numbered steps.

1.3.1 Strategy for Developing Algorithm

Algorithm development involves the following steps to carry out. We can proceed to correct solution of a particular problem by adopting the following strategy.

Step 1: Investigation

- (i) Identify the processes.
- (ii) Identify the major decisions.
- (iii) Identify the repetitions.
- (iv) Identify the variables.

Step 2: Preliminary algorithm

- (i) Devise a high-level (general) algorithm.
- (ii) Step through the algorithm. Does this "walk-through" reveal any major problems? If it does then correct the problems.

Step 3: Refining the algorithm

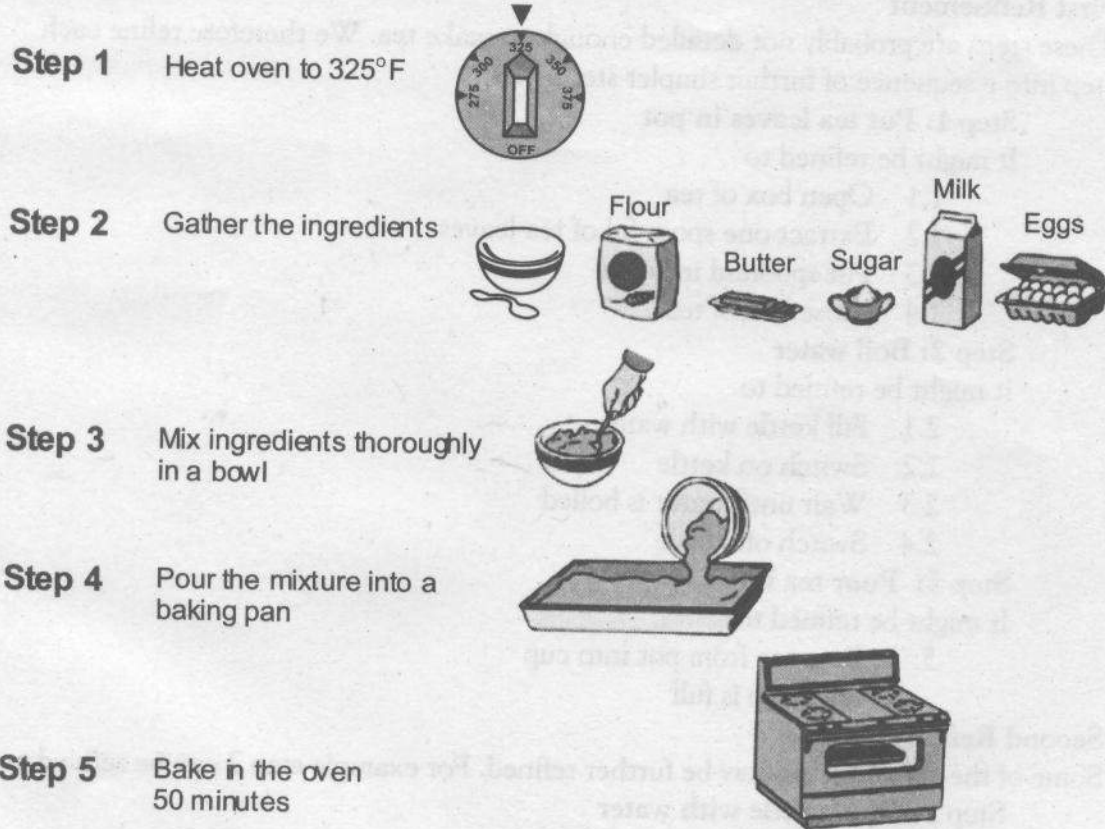
- (i) Incorporate any refinements indicated in step 2.
- (ii) Group together processes where appropriate.
- (iii) Group together variables where appropriate.
- (iv) Test the algorithm again by stepping through it.

Let us consider the following example to understand the way of approaching a solution to a problem.

Problem 1: You have to bake a cake in your house.

Step-form Algorithm

The following diagram shows the steps to follow to bake the cake.



- Step 6** Repeat
Bake 5 minutes more
Until cake top springs back when touched in the center



- Step 7** Cool on a rack before cutting

Step-form Algorithm: Baking a cake

Problem 2

Making tea: You have to make tea in your house.

Step-form Algorithm

An initial attempt at an algorithm might be:

1. Put tea leaves in pot
2. Boil water
3. Add water to pot
4. Wait 5 minutes
5. Pour tea into cup

First Refinement

⇒ These steps are probably not detailed enough to make tea. We therefore refine each step into a sequence of further simpler steps:

Step 1: Put tea leaves in pot

It might be refined to

- 1.1 Open box of tea
- 1.2 Extract one spoonful of tea leaves
- 1.3 Put spoonful into pot
- 1.4 Close box of tea

Step 2: Boil water

It might be refined to

- 2.1 Fill kettle with water
- 2.2 Switch on kettle
- 2.3 Wait until water is boiled
- 2.4 Switch off kettle

Step 5: Pour tea into cup

It might be refined to

- 5.1 Pour tea from pot into cup
until cup is full

⇒ Second Refinement

Some of the refined steps may be further refined. For example step 2 can be refined as:

Step 2.1 Fill kettle with water

It might be refined to

- 2.1.1 Put kettle under tap
- 2.1.2 Turn on tap
- 2.1.3 Wait until kettle is full
- 2.1.4 Turn off tap

Other steps may also require further refinement. After a number of refinements the robot is able to execute every step.

<u>Original Algorithm</u>	<u>First Refinement</u>	<u>Second Refinement</u>
1. Put tea leaves in pot	1.1 Open box of tea 1.2 Extract one spoonful 1.3 Put spoonful into pot 1.4 Close box of tea	1.1.1 Take tea box from shelf 1.1.2 Remove lid from box 1.4.1 Put lid on box 1.4.2 Replace tea box on shelf
2. Boil Water	2.1 Fill kettle with water 2.2 Switch on kettle 2.3 Wait until water boiled 2.4 Switch off kettle	2.1.1 Put kettle under tap 2.1.2 Turn on tap 2.1.3 Wait until kettle is full 2.1.4 Turn off tap 2.3.1 Wait until kettle whistles
3. Add water to pot	3.1 Pour water from kettle until pot is full	
4. Wait 5 Minutes		
5. Pour tea into cup	5.1 Pour tea from pot into cup until cup is full	

Example 1: Write an algorithm to find the sum of first 50 natural numbers.

Algorithm

```
BEGIN
  SUM = 0
  N = 0
  DO WHILE (N <= 50)
    SUM = SUM + N
    N = N + 1
  END DO
END
```

Example 2: Write an algorithm to find the factorial of a given number

Algorithm



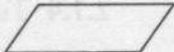




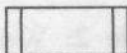
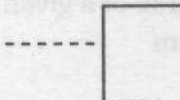
```
BEGIN
  fact = 1
  n = 1
  PRINT "Enter a number"
  INPUT num
  FOR n = 1 to num
    fact = fact * n
  NEXT n
  PRINT fact
END
```

1.4 FLOWCHART

Flowchart is the pictorial representation of an algorithm. It is a way of visually presenting the flow of data, the operations performed within the system and the sequence in which they are performed. The flowchart is similar to the layout plan of a building. As we know a designer draws a layout plan before starting construction on a building. Similarly, a programmer prefers to draw a flowchart prior to writing a computer program. As in the case of drawing of a layout plan, the flowchart is drawn according to defined rules.

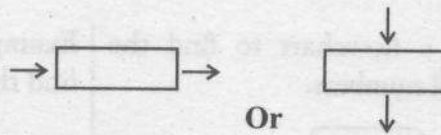
1.4.1 Symbols of Flowchart

Flowcharts are usually drawn using some standard symbols; however, other special symbols can also be developed when required. The standard symbols which are frequently used in flowcharting are shown below:

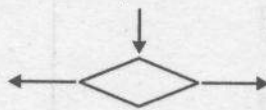
Symbol	Purpose
	Start/End of a flowchart
	Processing
	Input/output
	Decision making and branching
	Connector
	Off-page/On-page connector
	Flow lines
	Pre-defined Process (functions/sub-routines)
	Remarks

1.4.2 Guidelines for Drawing a Flowchart

- In drawing a flowchart, all necessary requirements should be listed in a logical order.
- The flowchart should be clear, neat and easy to follow. There should not be any ambiguity in understanding the flowchart.
- The usual direction of the flowchart is from top to bottom or left to right.
- Only one flow line should come out from a process symbol.



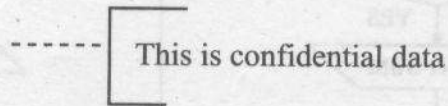
- e. Only one flow line must enter a decision symbol, but two flow lines, one for each possible answer, must leave it.



- f. Only one flow line is used in conjunction with terminal symbol. Ensure that the flowchart has a logical start and end.



- g. Write comments within Remarks symbol. We can use the remarks (annotation) symbol to describe steps more clearly.



- h. If the flowchart becomes complex, it is better to use connector symbols to reduce the number of flow lines. The intersection of flow lines should be avoided to make it more effective and clear.
- i. It is useful to test the validity of the flowchart by passing through it with a simple test data.

1.4.3 Advantages Of Flowcharts

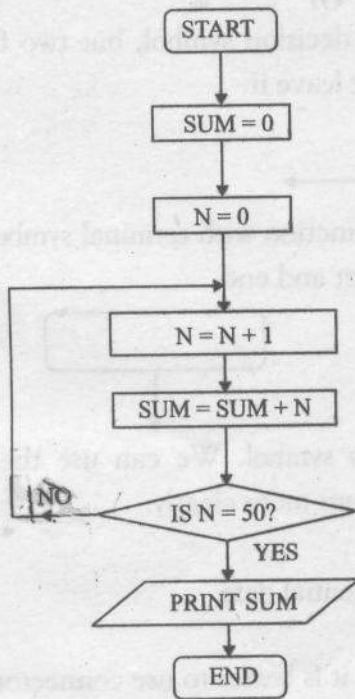
The benefits of flowcharts are:

- (i) With the help of a flowchart, the logic of an algorithm can be described more effectively.
- (ii) As flowcharts are part of the design document, hence maintenance of operational programs becomes easy.
- (iii) The flowcharts act as a guide for the program development. Therefore, they help the programmer to put efforts more efficiently on the underlying problem.
- (iv) The flowchart helps in debugging process.

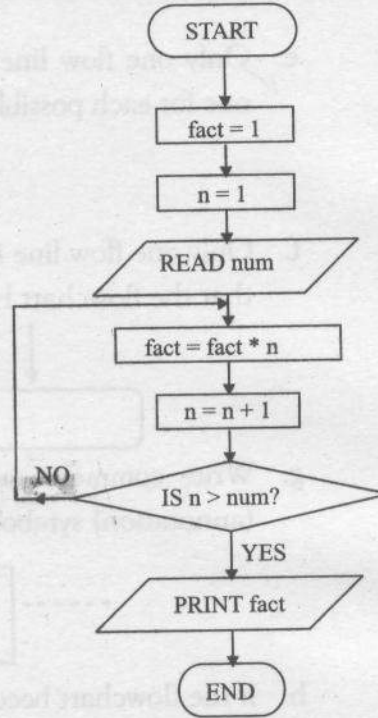
1.4.4 Limitations Of Flowcharts

- (i) It is difficult to draw flowcharts for complex problems.
- (ii) If alterations are required, the flowchart is to be redrawn.

Example 3: Draw a flowchart to find the sum of first 50 natural numbers.



Example 4: Draw a flowchart to find the factorial of a number



Exercise

1. Fill in the blanks:

- (i) The set of instructions given to the computer to solve a problem is called _____.
- (ii) The set of rules for writing programs in a programming language is known as _____ of the language.
- (iii) Flowchart is a _____ representation of algorithm.
- (iv) An algorithm solves a problem in _____ number of steps.
- (v) During _____, a problem is decomposed into multiple sub-problems.
- (vi) Debugging is the process of finding and removing _____ in a program.
- (vii) Program _____ refers to the installation of the program in the user environment.
- (viii) Occurrence of a _____ error crashes the program.

2. Choose the correct answer:

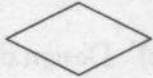
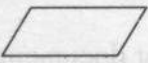

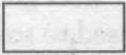

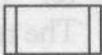
- (i) BASIC is a:

(a) High Level Language	(b) Low Level Language
(c) Assembly Language	(d) Machine Language

- (ii) How many possible solutions are there for a problem?
 - (a) One
 - (b) Two
 - (c) Three
 - (d) Multiple
- (iii) Program upgradation refers to:
 - (a) Program enhancement
 - (b) Program identification
 - (c) Program development
 - (d) Program implementation
- (iv) Which of the following tasks are performed by most of the algorithms?
 - (a) Input
 - (b) Output
 - (c) Processing
 - (d) All of these
- (v) Typographical errors in BASIC statements are:
 - (a) Runtime errors
 - (b) Logical errors
 - (c) Syntax errors
 - (d) Execution errors
- (vi) The diamond symbol represents the:
 - (a) Input/Output
 - (b) Decision making
 - (c) Processing
 - (d) Remarks
- (vii) Division by zero is:
 - (a) Syntax error
 - (b) Logical error
 - (c) Runtime error
 - (d) Not an error
- (viii) Which of the following documents describe various features of the software and the way it is used?
 - (a) Software requirement specification
 - (b) Problem description
 - (c) User manual
 - (d) Algorithm
- (ix) Algorithm is a:
 - (a) Requirement document
 - (b) Design document
 - (c) Test document
 - (d) user guide
- (x) The technique 'Divide and Conquer' is used to solve:
 - (a) Simple problems
 - (b) Complex problems
 - (c) Large problems
 - (d) Complex and large problems

3. Write T for True and F for False statements.

- (i) Syntax errors occur due to wrong program logic.
- (ii) Top-down design is followed to solve complex problems.
- (iii) Desk checking is the process of verifying the working of an algorithm.
- (iv) Debugging is an important part of analysis.
- (v) Every stage of program development is documented.
- (vi) A rectangle symbol is used for decision making in a flowchart.
- (vii) Requirement document is not helpful at development stage.
- (viii) The usual direction of flowchart is from right to left.
- (ix) Annotation symbol is used for writing comments.

4. What do you mean by problem solving? Briefly describe the problem solving process.
5. What is debugging? How many types of errors can occur in a program? Describe briefly.
6. Define algorithm. Write a step-form algorithm for making a telephone call to your friend.
7. What are the advantages of flowchart? Discuss limitation of flowchart.
8. Draw a flowchart to find the largest of three numbers.
9. Write an algorithm to calculate the area of a circle when the radius is given.
(area = $3.14 * \text{radius} * \text{radius}$)
10. Answer the following short questions:
 - (i) List steps that should be followed to solve a problem.
 - (ii) What is analysis? Describe its importance in solving a problem.
 - (iii) What method should be adopted to solve complex problems? Discuss briefly.
 - (iv) What do you mean by syntax of a programming language? Is it necessary to know the syntax for solving a problem on computer?
 - (v) Differentiate runtime errors and logical errors.
 - (vi) Why documentation is considered vital in problem solving process? Give reasons.
 - (vii) Is it necessary for an algorithm to solve a problem in finite number of steps? If yes, why?
 - (viii) Write purpose of the following flowchart symbols:
 - (i)  (ii)  (iii) 
 - (iv)  (v)  (vi) 
 - (ix) Compare flowchart and algorithm.
 - (x) Write an algorithm to calculate the distance covered by a car moving at an average speed of $v \text{ ms}^{-1}$ in time t . The program should input average speed v and time t .
[Hint: $s = vt$, where s is the distance traveled]

Answers

1. (i) Program (ii) Syntax (iii) Pictorial (iv) Finite
(v) Analysis (vi) Errors (vii) Implementation (viii) Runtime
2. (i) a (ii) d (iii) a (iv) d (v) c
(vi) b (vii) c (viii) c (ix) b (x) d
3. (i) F (ii) F (iii) T (iv) T (v) F
(vi) T (vii) F (viii) F (ix) F (x) T

DATA TYPES, ASSIGNMENT AND INPUT/OUTPUT STATEMENTS

2.1 INTRODUCTION

The BASIC language was developed by John Kemeny and Thomas Kurtz in 1963 at Dartmouth College, USA. It was invented as an instructional tool to teach fundamental programming concepts to students. It was developed to address the complexity issues of older languages.

GW-BASIC is an interpreter for BASIC (Beginner's All-Purpose Symbolic Instruction Code) language. There are many other translators (compilers and interpreters) for BASIC from different vendors such as QBASIC (Quick BASIC) which provides a menu-driven environment to write and execute BASIC programs. Here we shall refer to only GW-BASIC because it provides a simple and easy to use environment.

2.2 MODES OF OPERATION

GW-BASIC can operate in two modes i.e., direct mode and indirect mode. When GW-BASIC is loaded, it shows *ok* message. At this stage, it is in direct mode.

In *direct mode*, GW-BASIC commands are executed as they are typed. Results of arithmetic and logical operations can be displayed immediately, but the commands themselves are lost after execution. This mode is useful for debugging and for quick computations that do not require a complete program.

```
Ok
PRINT 786/3
262
Ok
PRINT "Welcome to GW-BASIC"
Welcome to GW-BASIC
Ok
```

Fig. 2.1: Example of Direct Mode

The *indirect mode* is used to type programs. Program statements are always preceded by line numbers, and are stored in memory. The program loaded in memory is executed by entering the **RUN** command (fig. 2.2).

```
Ok
auto
10 PRINT 786/3
20 PRINT "Welcome to GW-BASIC"
30 END
Ok
RUN
262
Welcome to GW-BASIC
Ok
```

Fig. 2.2: Example of Indirect Mode

- GW-BASIC **commands** are executable instructions which are operated in the direct mode. They do not require a preceding line number.
- GW-BASIC **statements** are written as a program and each statement is preceded by a line number.

2.3 WRITING PROGRAMS IN GW-BASIC

GW-BASIC provides an IDE (Integrated Development Environment) where we can write, edit, save, load and execute BASIC programs (fig. 2.3).

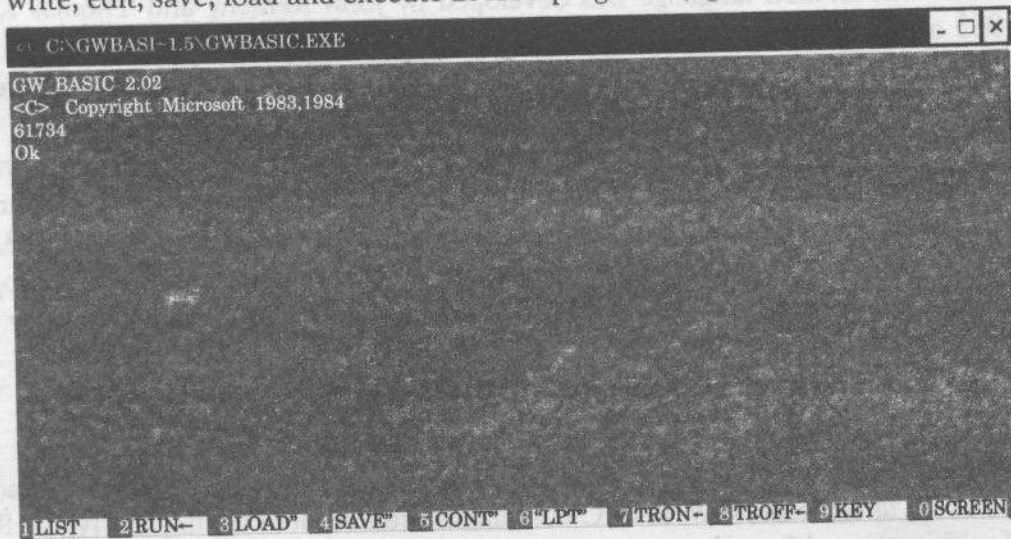


Fig. 2.3: GW-BASIC IDE

2.3.1 Create and Save the Program

The program is a file that contains specific instructions, or statements, for the computer. In a GW-BASIC program, lines have the following format:

Line# statement(s)

where, **Line#** is an unsigned integer in the range from 0 to 65529 and **statement(s)** is any valid GW-BASIC statement. A GW-BASIC program line can not have more than 255 characters. However, there may be more than one statement on a line. If so, each must be separated by a colon (:). The program's statements are executed depending on line numbers in ascending order. For example, if line 45 is typed after line 60, the computer would still run line 45 before line 60.

Reuse of an existing line number causes all of the information contained in the original line to be lost. You may erase some program lines by accident.

In order to use the program in future, we must save it. To save a file in GW-BASIC, the following procedure is used:

- (i) Press the F4 key or type SAVE command.
- (ii) Type a valid name (in quotes) for the program, and press the ENTER key. The file is saved under the specified name.

2.3.2 Load the Program

Loading the program refers to bring it into memory from secondary storage device such as hard disk, so that it can be used. A saved program can be loaded by using the following procedure:

- (i) Press the F3 key or type LOAD command.
- (ii) Type the name of an existing file (in quotes).
- (iii) Press ENTER.

The file is loaded into memory, and is ready to list, edit, or run.

- If the file does not exist or an invalid path is typed, an error message is displayed.
- The default extension of GW-BASIC program is **.bas**
- To save the file in a directory, you should specify the complete path with the file name. Otherwise the file will be saved in the current working directory of GW-BASIC.

2.3.3 Execute the Program

Execution of the program refers to carrying out instructions of the program. The program must be loaded into memory (RAM) before execution. Therefore to execute a program, first load it as described above. Then press F2 key or type RUN command, the output of the program will follow it.

```
Ok
10 LET x = 10: LET y = 20
20 PRINT "SUM = "; x + y
30 END
SAVE"C:\sum. bas"
Ok
```

Fig. 2.4: Creating and saving the program

```
Ok
LOAD "c:\sum.bas"

RUN
SUM = 30
Ok
```

Fig. 2.5: Loading and running the program

2.4 Structure of the BASIC Program

Every BASIC program should follow the following rules.

- (i) Every program statement must begin with a line number
- (ii) It is a good practice to end every BASIC program with an END statement. However, it is not mandatory.
- (iii) Repetition of line numbers within a program is not allowed
- (iv) Two or more statements can be written on a line but they must be separated by a colon (:)
- (v) In BASIC, variables can be used without declaration.
- (vi) In a BASIC program, the physical appearance of the program statements does not matter. For example, in a program line number 90 can appear before line number 60; however the program statements will always execute according to the ascending order of specified line numbers.

2.5 Character Set of BASIC

Character set of a language defines all characters which are valid to use in programs written in that language. The character set of GW-BASIC comprises of alphabets, numeric and some special characters.

- The alphabets include uppercase and lowercase letters
- Numeric characters include digits from 0 to 9
- GW-BASIC characters set include the following special characters

Character	Meaning
BLANK	
=	Equal sign or assignment symbol.
+	Plus sign or string concatenation.
-	Minus sign.
*	Asterisk or multiplication symbol.
/	Slash or division symbol.
^	Caret, exponentiation symbol, or CTRL key.
(Left parenthesis.
)	Right parenthesis.
%	Percent or integer declaration.
#	Number sign or double-precision declaration.
\$	Dollar sign or string declaration.
!	Exclamation point or single-precision declaration.
[Left bracket.
]	Right bracket.
,	Comma.
""	Double quotation marks or string delimiter.
.	Period, dot, or decimal point.
'	Single quotation mark, apostrophe, or remark indicator.
;	Semicolon or carriage return suppressor.
:	Colon or line statement delimiter.
&	Ampersand or descriptor for hexadecimal and octal number conversion.
?	Question mark.

<	Less than symbol.
>	Greater than symbol.
\	Backslash or integer division symbol.
@	"At" sign.
· -	Underscore.
BACKSPACE	Deletes last character typed.
ESC	Erases the current line from the screen.
TAB	Moves print position to next tab stop. Tab stops are every eight columns.
ENTER	Terminates input to a line and moves cursor to beginning of the next line, or executes statement in direct mode.

2.6 RESERVED WORDS

Reserved words or **keywords** are the words, which have predefined meaning in BASIC. These have predefined uses and cannot be used or redefined for any other purpose in a BASIC program. Keywords cannot be used as variable names. Some of the keywords of BASIC are IF, ELSE, THEN, WHILE etc.

2.7 VARIABLES

Variables are named memory locations (memory cells) which are used to store program's input data and its computational results during program execution.

As the name suggests, the value of a variable may change during the program execution. If a variable is assigned no value, the GW-BASIC assumes the value of variable to be zero in case of numeric variables and null to string variables.

2.7.1 Rules for Naming Variables in BASIC

Each variable that is used in a BASIC program must have a name. The name of a variable is used for further references made to it. The value of the variable is accessed by its name. In GW-BASIC, there are some rules for naming variables. These are:

- (i) In GW-BASIC, a variable name can not be more than 40 characters long.
- (ii) The variable name may contain alphabets (both uppercase and lowercase), numbers, and the decimal point.
- (iii) The first character in the variable name must be an alphabet.
- (iv) Reserved words can't be used as variable names.
- (v) Blank spaces are not allowed in variable names.
- (vi) However, the last character of a variable name may be a special type declaration character indicating the type of a variable.

If type of a variable is not specified, it is assumed as a Real type variable.

2.7.2 Type Declaration Characters

In GW-BASIC, type declaration characters represent the type of variable. Following type declaration characters are recognized in GW-BASIC.

Character	Type of Variable	Example	Memory Required
\$	String variable	Name\$	String length
%	Integer variable	Marks%	2 Bytes
!	Single-precision variable	Avg!	4 Bytes
#	Double-precision variable	Area#	8 Bytes

2.7.3 Types of Variables

There are two basic types of variables:

- *Numeric Variables*
- *String Variables* that can store strings of characters

Numeric Variables

Numeric variables can store numeric values (Numeric values include both floating point numbers and whole numbers). If we don't specify the type of a numeric variable, GW-BASIC considers it as single-precision. Single-precision variables can accurately handle numbers up to six significant digits, however it can not handle seventh significant digit accurately. If more accuracy is desired, we should rather use double-precision.

String Variables

A *string* can be defined as a sequence of characters enclosed in double quotations. A string variable can therefore store sequence of characters. The nature of character string is entirely different from the nature of numeric values. BASIC and other languages use different formats for storing numeric and string data. In BASIC, a dollar sign (\$) is followed by the name of the string variable. We can not perform the same set of operations on strings that we can perform on numeric values. For example, strings can not be added, subtracted, multiplied or divided. There are some other operations that can be performed on strings such as concatenation and comparison.

Be very careful when making conversions between integer, single-precision, and double-precision variables. Otherwise you may lose accuracy accidentally.

2.8 CONSTANTS

A *constant* is a quantity whose value can not be changed. Unlike a variable, the value stored in a constant can't be changed during program execution. In BASIC, there are two types of constants. These are *numeric constants* and *string constants*.

2.8.1 Numeric Constants

Numeric constants consist of integers, single-precision, or double-precision numbers. Integer constants represent values that are counted and do

not have a fractional part e.g., +56, -678, 8, etc. Single-precision or double-precision numeric constants represent values that are measured and may contain fractional part e.g., - 4.786, 5.0, 0.45 etc. Single-precision numeric constants are stored with 7 digits (although only 6 may be accurate). Double-precision numeric constants are stored with 17 digits of precision, and printed with as many as 16 digits.

A single-precision constant is any numeric constant with either

- Seven or fewer digits
- Exponential form using E
- A trailing exclamation point (!)

A double-precision constant is any numeric constant with either

- Eight or more digits
- Exponential form using D
- A trailing number sign (#)

The following are examples of single- and double-precision numeric constants:

Single-Precision Constants Double-Precision Constants

23.08	342237861
-3.15E-04	-5.35857D-06
2145.0	3220.0#
37.4!	7645721.1334

2.8.2 String Constants

A *string constant* is a sequence of alphanumeric characters enclosed in double quotation marks. The maximum length of a string constant is 255 characters. For example, "Lahore", "4900", "I love Pakistan" etc.

2.9 BASIC COMMANDS

Here are some commonly used commands of BASIC language.

2.9.1 AUTO Command

This command automatically generates line numbers in an increasing order, each time the ENTER key is pressed.

Syntax:

- **AUTO** [*line number*][,*increment*]
- **AUTO** .[,*increment*]

Interpretation:

AUTO is useful for typing a program because it relieves us typing line numbers again and again. In the above syntax [*line number*] specifies the starting line number of the program, whereas [*increment*]

specify the subsequent increment to the previous line number. The default for both values is 10.

The period (.) can be used as a replacement for *line number* to indicate the current line. If [*increment*] after the *line number* is omitted, the last increment specified in an AUTO command is assumed.

If AUTO generates a *line number* that is already being used, an asterisk appears after the number to warn that any input will replace the existing line. However, pressing ENTER immediately after the asterisk saves the line and generates the next line number.

AUTO is terminated by entering CTRL-BREAK or CTRL-C. GW-BASIC will then return to command level.

Examples:

- AUTO 100, 20
Generates line numbers 100, 120, 140, and so on
- AUTO
Generates line numbers 10, 20, 30, 40, and so on
- AUTO , 50
Generates line numbers starting from the line number specified in the last AUTO command with the increment of 50
- AUTO
Generates line numbers starting from the current line number (i.e., the last line number generated from the previous AUTO command) with the increment specified in the previously entered AUTO command.

2.9.2 CLEAR Command

This command sets the values of all numeric variables to zero, the values of all string variables to null opened and closes all files.

Syntax:

CLEAR

Example: The adjoining figure shows the effect of CLEAR command.

```
Ok
X = 100
Ok
PRINT X
100
Ok
CLEAR
Ok
PRINT X
0
Ok
```

Fig. 2.6: Example of CLEAR Command

2.9.3 CLS Command

This command is used to clear the screen.

Syntax:

CLS [n]

Interpretation:

Here *n* is optional, but it can take any of the following values:

Value of <i>n</i>	Effect
0	Clears the screen of all text and graphics
1	Clears only the graphics
2	Clears only the text

Example:

- CLS (Type CLS at command level)
- CLS 1

2.9.4 DELETE Command

This command is used to delete program lines or line ranges of loaded program.

Syntax:

- DELETE [*line number1*][–*line number2*]
- DELETE *line number1* –

Examples:

- DELETE 70
Deletes line 70.
- DELETE 50-150
Deletes lines 50 through 150, inclusively
- DELETE -80
Deletes all lines from start up to including line 80
- DELETE 120-
Deletes all lines from line 120 to the end of the program

2.9.5 EDIT Command

This command is used to modify a program line.

Syntax:

- EDIT *line number*
- EDIT.

Interpretation:

line number is the number of the program line which we want to edit. The period (.) refers to the current line.

Examples:

- EDIT 140
Displays program line number 140 for editing
- EDIT.
Displays the current program line for editing

2.9.6 FILES Command

This command is used to list the names of all files residing on the specified drive.

Syntax:

FILES [*pathname*]

Interpretation:

[*pathname*] is the optional parameter which if omitted; the command lists all files in the current directory of the selected drive. Wildcards such as * (asterisk) and ? can also be used in pathname. Question mark (?) is used to match any characters in the filename or extension, and asterisk (*) is used for any filename or extension.

Examples:

- **FILES**
Lists all files in the current directory of the selected drive
- **FILES "*.doc"**
Lists all files whose extension is doc
- **FILES "D:*.*"**
Lists all files on the D: drive with any extension
- **FILES "Mar?.xls"**
Lists all files
 - Whose extension is xls
 - Whose name consists of four characters
 - First three characters of their names are Mar and the fourth character could be any

```
Ok
FILES
C:\GWBASI-1.5
GWBASIC .EXE          GWBASIC3.EXE          SMINFO.  SYS
660643840 Bytes free
Ok

FILES "G:*.*"
G:\
1STYEA~1              1STYEA~2              2NDYEA~1              2NDYEA~2
ADMISS~1 <DIR>        ADNAN                  <DIR>        AMJAD          <DIR>        CHAT          <DIR>
CLASS1~1.XLS          CLASS12 .XLS          FINAL2~1          <DIR>        GORIYA~1.RTF
PAPER <DIR>          PRIMEM~1.DOC          PROGRA~1          <DIR>        RESULT~1.XLS
RESULT~2.XLS          RESULT~3.XLS          TESTRE~1          <DIR>        TOPOLOGY .DOC
XII-CL~1.XLS          XII-CL~2.XLS
1023932928 Bytes free
Ok
```

Fig. 2.7: Example of FILES Command

2.9.7 KILL Command

This command is used to remove/delete a file from the disk.

Syntax:

KILL *filename*

Interpretation:

KILL command is used to delete all types of files.

Examples:

- KILL "Inventory.bas"
Deletes the file Inventory.bas in the current directory
- KILL "G:\Goods\Inventory.*"
Deletes all files named **Inventory** with any extension

Be careful while using KILL command. Always specify the filename's extension when using the KILL command. You may lose your data accidentally while using this command.

2.9.8 LIST Command

This command is used to display a loaded program partially or completely on the screen.

Syntax:

- LIST [*line number*][- *line number*][,*filename*]
- LIST [*line number* -][,*filename*]

Interpretation:

In the above syntax, all parameters are optional which if omitted; the command lists the last entered program. [*line number*] is a valid line number within the range of 0 to 65529. If *filename* is not specified with the LIST command, the specified lines of the last typed/loaded program are listed.

Examples:

- LIST
Lists all lines in the program
- LIST -20
Lists lines of the programs up to the line number 20
- LIST 10-20
Lists lines from 10 through 20
- LIST 20-
Lists lines 20 through the end of the program

2.9.9 LOAD Command

This command loads a file from disk to memory.

Syntax:

LOAD *filename*[,*r*]

Interpretation:

filename is the name of the file to be loaded. If the [*r*] option is used with LOAD command, the program runs after it is loaded.

Examples:

- LOAD "D:\fact.bas"
Loads the file named fact.bas from D: drive
- LOAD "D:\fact.bas", r
Loads and executes the file fact.bas from D: drive

2.9.10 MKDIR Command

This command is used to create a subdirectory.

Syntax:

MKDIR *pathname*

Interpretation

pathname identify the location where the subdirectory is created. It is a string expression that should not exceed 63 characters.

Example

MKDIR "D:\Goods\Inventory"

Creates the subdirectory **Inventory** within the directory of **Goods**.

2.9.11 NAME Command

This command is used to rename a file.

Syntax:

NAME *old-filename* **AS** *new-filename*

Interpretation:

The file will be renamed; the *old-filename* is replaced by the *new-filename*

Example

NAME "Remarks.doc" **AS** "RMKS.doc"

Gives the name **RMKS.doc** to the file **Remarks.doc**

2.9.12 RENUM Command

This command is used to renumber the program lines.

Syntax:

RENUM [*new number*], [*old number*] [,*increment*]

Interpretation

New number is the starting line number in the new sequence. The default is 10. *old number* is the line in the current program where renumbering is to begin. The default is the first line of the program, *increment* is the increment to be used in the new sequence. The default is 10.

Examples

- **RENUM**
Assign new numbers to the whole program (if different) starting from line# 10 with the default increment of 10.
- **RENUM** 80, , 30
Assign new numbers to the whole program starting from the line# 80 with the increment of 30 in line numbers.
- **RENUM** 150, 70, 50

Assign new numbers to lines from 70 to the end of the program such that the new sequence will start from 150 (i.e., line# 70 will be renumbered to 150) and an increment of 50 is made for each next line.

2.9.13 RMDIR Command

This command is used to remove/delete a directory from the disk.

Syntax:

RMDIR *pathname*

Interpretation

The *pathname* is the path of an existing directory which should not exceed 63 characters. The directory to be deleted must be empty otherwise an error message will appear.

Example

RMDIR "D:\GOODS\INVENTORY"

Deletes the subdirectory INVENTORY of the directory GOODS.

2.9.14 RUN Command

This command is used to execute the program currently in memory. If the program is not in memory, it first load and then run it.

Syntax:

- **RUN** [*line number*] [, **r**]
- **RUN** *filename* [, **r**]

Interpretation

By default the RUN command starts executing the program from the beginning. However, if the *line number* is specified then the execution of the program starts from that particular line number. When the *filename* is specified with RUN command, it closes all opened files and deletes all memory contents before loading and executing the specified file from the disk. The **r** option is used to keep all data files opened during the execution of RUN command.

Example:

RUN "table.bas", **r**

Executes table.bas without closing data files

If you are using the speaker on the computer, please note that executing the RUN command will turn off any sound that is currently running and will reset to Music Foreground.

2.9.15 SAVE Command

This command is used to save the program on the disk for later use.

Syntax:

- **SAVE** *filename*, [, **a**]
- **SAVE** *filename*, [, **p**]

Interpretation

By default GW-BASIC saves the file in a compressed binary format. If the option [a] is specified, the file is saved in ASCII format. The option [p] saves the file in an encoded binary format (protected format). We can not list or edit a file saved in protected format, however it can be executed.

Examples

- SAVE "matrix.bas", a
Saves the file matrix.bas in ASCII format
- SAVE "matrix.bas", p
Saves the file in encoded binary format

2.9.16 SYSTEM Command

This command is used to exit from GW-BASIC and return to operating system environment.

Syntax:

SYSTEM

Example

SYSTEM (Type in direct mode)

2.9.17 LIST Command

This command is used to list all or part of the program currently in memory to the printer.

Syntax:

LLIST [*line number*] [-*line number*]

LLIST [*line number*-]

Interpretation:

GW-BASIC always returns to command level after a LLIST is executed. The line range options for LLIST are the same as for LIST.

2.9.18 PRINT Command

This command is used to print data at the printer.

Syntax:

LPRINT [*list of expressions*] [;]

Interpretation:

list of expressions consists of the string or numeric expressions separated by semicolons. *string expressions* is a string literal or variable consisting of special formatting characters. The formatting characters determine the field and the format of printed strings or numbers.

This statement is the same as PRINT, except that output goes to the printer. For more information about string and numeric fields and the variables used in them, see the print statement. The LPRINT statement assumes that the printer is an 80-character wide printer. To

reset the number of characters that can be printed across the printed page (assuming that the printer is wider than 80 characters), see the WIDTH statement.

2.9.19 CONT Command

This command is used to continue program execution after a break.

Syntax:

CONT

Interpretation:

Resumes program execution after CTRL-BREAK, STOP halts a program. Execution continues at the point where the break happened. If the break took place during an INPUT statement, execution continues after reprinting the prompt.

CONT is useful in debugging; in that it lets us set break points with the STOP statement, modify variables using direct statements, continue program execution, or use GOTO to resume execution at a particular line number. If a program line is modified, CONT will be invalid.

2.10 BASIC Statements

Here are some commonly used statements of BASIC language.

2.10.1 END Statement

This statement is used to terminate program execution, close all files and return to the command level.

Syntax:

END

2.10.2 REM Statements

This is a non-executable statement and is used to add explanatory remarks in the program.

Syntax:

- **REM [remarks]**
- **' [remarks]**

Example

```
10 REM This program calculates the average of two numbers
20 a = 15
30 b = 25
40 avg = (a + b) / 2
50 ' Display the average
60 PRINT "Average = "; avg
70 END
```

In this program line# 10 and line# 50 are non-executable. The BASIC interpreter does nothing to these lines; they are not translated

to machine language during the translation process. The REM statement however increases the readability of the code and helps in modifying, understanding and debugging the program

2.10.3 STOP Statement

This statement is used to terminate program execution temporarily, and return to command level.

Syntax:

STOP

Interpretation:

STOP statements may be used anywhere in a program to terminate execution of the program. When a STOP statement is encountered in the program, the following message is printed:

Break in line nnnnn

Unlike the END statement, the STOP statement does not close files. GW-BASIC always returns to command level after a STOP is executed. Execution is resumed by issuing a CONT command.

```
Ok
auto
10 INPUT A, B, C
20 K = A 2* 5.3: L = B 3/0.26
30 STOP
40 M = C*K+100: PRINT M
50 END
Ok

RUN
? 1, 2, 3
Break in 30
Ok

PRINT L
30.76928
Ok

Continue
115.9
Ok
```

Fig. 2.8: Example of CONT Command

2.11 OPERATORS IN BASIC

Operators are symbols which are used to perform certain operations on data. These include arithmetic, relational, logical, and assignment operators.

2.11.1 Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on values (numbers). The GW-BASIC defines the following standard arithmetic operators:

Operation	Symbol	Algebraic Expression	BASIC Expression
Addition	+	$a + b$	$a + b$
Subtraction	-	$a - b$	$a - b$

Multiplication	*	$a \times b$	$a * b$
Division	/	a / b	a / b
Exponent	^	a^n	$a \wedge n$
Negation	-	$-a$	$-a$
Modulus	MOD	$a \text{ MOD } b$	$a \text{ MOD } b$
Integral Division	\	$a \setminus b$	$a \setminus b$

The use of first six operators is straightforward. The last two operators are **modulus** (also called **remainder** operator) and **integral-division**. Contrary to the **division** operator which returns the quotient, the *modulus* operator returns the remainder of an integral division. For example, if a, and b are two integers having values 8 and 3 respectively, then the express **a MOD b** will be evaluated to **2**, which is the remainder of integral division. The integral-division does not allow a fractional value in the quotient. It always returns the quotient as a whole number. For example, the result of the expression **a \ b** (for above two variables a and b) will be 2.

2.11.2 Relational Operators

Relational operators are used to compare two values. These operators always evaluates to **true** or **false**. They always produce a **non-zero** value (in most cases 1) if the relational expression evaluates to **true**, or a **0** value if the relational expression evaluates to **false**. There are six basic relational operators in BASIC. Suppose a, b, and c are three integer variables having values 123, 215 and 123 respectively then:

Operation	Symbol	Expression	Evaluation
Equal to (comparison)	=	$a = c$	true (non-zero)
Less than	<	$b < a$	false (zero)
Greater than	>	$a > c$	false (zero)
Less than or Equal to	<=	$a <= b$	true (non-zero)
Greater than or Equal to	>=	$b <= a$	false (zero)
Not Equal to	<>	$a <> b$	true (non-zero)

2.11.3 Logical Operators

Logical operators let us combine simple conditions to construct more complex ones (By condition, we mean an expression evaluating to **true** or **false**). There are three basic logical operators in BASIC. These are AND, OR, and NOT.

The first logical operator i.e., AND when combines two conditions, evaluates to *true* if both the conditions are true, otherwise, it evaluates to *false*. The second logical operator OR when combines two conditions, evaluates to *true* if any one of the conditions is true, otherwise evaluates to *false*. Similarly, the third logical operator NOT when applied to a condition, reverse the result

of the evaluation. It means that if the condition evaluates to true, the logical NOT operator evaluates to *false* and vice versa. In an expression, logical operations are performed after arithmetic and relational operations. To understand the working of these operators consider the following table:

Logical Operation	Value	Value	Result
NOT	X	-	NOT X
	T	-	F
	F	-	T
AND	X	Y	X AND Y
	T	T	T
	T	F	F
	F	T	F
	F	F	F
OR	X	Y	X OR Y
	T	T	T
	T	F	T
	F	T	T
	F	F	F

Table 2.1: Results of logical operators

2.11.4 Concatenation Operators

All relational operators can be used with strings to perform comparison. In addition to relational operators another operation called string concatenation can also be applied to string constants and variables. The symbol for string concatenation operation is '+' and it joins two strings. Consider the adjoining figure showing the example of string concatenation:

```
Ok
auto
10 A$ = "Punjab Text"
20 B$ = "Book Board"
30 PRINT A$ + B$
40 END
50
Ok
RUN
Punjab TextBook Board
Ok
```

Fig. 2.9: String concatenation

2.11.5 Assignment Operator

The assignment operator is used to store a value, string or a computational result in a variable. In BASIC, the symbol = represents the operator i.e.

variable_name = expression

where expression may be a numeric or string expression. e.g.,

a = 10

a\$ = "Hello"

The value to the right side of the operator is assigned to the variable on the left side of the assignment operator. This statement is also called

assignment statement. Note it that the symbol '=' is also used for comparison; however it depends on the context where it is used.

2.11.6 Operator Precedence

An operator's precedence determines its order of evaluation in an expression. Table 2.2 lists the precedence of some of the BASIC operators from highest to lowest.

Operator	Precedence	
()		
^		
- (Negation)		
*, /		
\		
MOD		
+, -		
=, <>, <, <=, >, >=		
NOT		
AND		
OR		
= (assignment operator)		
		Lowest

Table 2.3: Operators' precedence

2.12 TYPE CONVERSION

When the program tries to store one type of numeric value to the variable of another type, GW-BASIC performs the type conversion according to the following rules:

- If a numeric constant of one type is assigned to a numeric variable of a different type, the number is converted according to the type of the variable.

For example:

```
10 LET x% = 51.39 'integer variable - storing a floating point value
20 PRINT x%
RUN
51
```

- During the evaluation of an expression (arithmetic or relational), all of the operands are converted to the degree of precision of the most precise operand.

For example:

```
10 A# = 12#/13
20 PRINT A#
RUN
.9230769230769231
```

The arithmetic is performed in double-precision, and the result is returned in A# as a double-precision value.

```
10 A = 12#/13
20 PRINT A
RUN
.9230769
```

The arithmetic is performed in double-precision, and the result is returned to A (single-precision variable) rounded and printed as a single-precision value.

- When a floating-point value is converted to an integer, the fractional portion is rounded. For example:

```
10 num% = 23.67
20 PRINT num%
RUN
24
```

- A string variable cannot be assigned to a numeric value.

2.13 ASSIGNMENT STATEMENT

GW-BASIC presents two ways to assign the value of an expression to a variable. First, by using assignment operator i.e., '=' which we have discussed in the previous text and the second is by using LET statement. Amazingly it is the only statement which itself is optional but whose parameters are mandatory.

Syntax:

[LET] *variable* = *expression*

Interpretation

Here, the word LET is optional, however we must specify the variable name and the expression whose value is to assign to the variable.

Example

We have already seen some examples of LET statement in this text.

Let us consider another example:

```
10 REM Calculate Average of Two Numbers
20 LET m = 24
30 LET n = 26
40 LET avg = (m + n) / 2
50 PRINT "average = "; avg
60 END
RUN
average = 25
```

Note: This program can also be completed without using LET statement

INPUT/OUTPUT Statements

Like other high level programming languages, GW-BASIC also provide statements to input data and show results. Input/Output statements make the

program more interactive and usable. In this section we shall discuss some basic input/output statements of GW-BASIC.

2.14 READ/DATA STATEMENT

This statement is used to store the numeric and string constants that are accessed by a READ statement specified somewhere in the program.

Syntax:

DATA *comma-separated list of constants*

Interpretation

Constants may be any string or numeric constant. String constants must be enclosed in double quotation marks only if they contain commas, colons, or spaces. Otherwise, quotation marks are not needed. The list of constants with a DATA statement can not exceed one line. If the list is too long to fit in one line then we should use another DATA statement to specify the remaining list of constants. The data contained in multiple DATA statements is assumed to be a single continuous list of items regardless of where the DATA is placed in the program. The variable type (numeric or string) given in the READ statement must agree with the corresponding constant in the DATA statement.

Example

```
10 'This program demonstrates the use of DATA and READ statements
20 READ A, B, C$
30 PRINT C$; "=", (A+B)/2
40 DATA 10, 20, "Average"
50 END
RUN
```

Average = 15

READ Statement

This statement reads values from the DATA statement and assigns them to corresponding variables.

Syntax:

READ *comma-separated list of variables*

Interpretation

The READ statement is the complementary part of the DATA statement. The READ and DATA statements are always used in conjunction. The READ statement specifies a list of variables and reads the corresponding values for these variables from the list of constants specified in DATA statement. It means the first variable in READ statement is assigned the first value from the list of constants in DATA statement; the second variable is assigned the second value, the third variable is assigned the third value and so on.

However, we should be very careful about the type of the variables and constants (in DATA statement) i.e., a numeric variable must be assigned a numeric value and a string variable must be assigned a string value. If the program tries to assign a string value to a numeric variable or vice versa, a "type mismatch error" will occur.

A READ statement may read values from several DATA statements (if there are more variables in READ statement than the number of constants in DATA statement), similarly several READ statements may read values from one DATA statement. If the number of variables in list of variables exceeds the number of elements in the DATA statement(s), an "Out of data" message occur. If the number of variables specified is fewer than the number of elements in the DATA statement(s), subsequent READ statements begin reading data at the first unread element. If there are no subsequent READ statements, the extra data is ignored.

Example:

```
10 REM In this program one READ statement reads data from two
   DATA statements
20 REM This program calculates the perimeter of a pentagon, where
   a, b, c, d, and e
30 REM are the lengths of sides of pentagon
40 READ A, B, C, D, E, P$
50 perimeter = a + b + c + d + e
60 PRINT P$; " = "; perimeter
70 DATA 12, 18, 24
80 DATA 18, 35, perimeter
90 END
RUN
perimeter = 107
```

2.15 RESTORE statement

This statement causes the DATA statement to be reused (if it has already been used) by the READ statement.

Syntax:

RESTORE [*line number*]

Interpretation

The *line number* specifies the line number of a DATA statement which has to be read again. The next READ statement reads the first item in the specified DATA statement. We can omit line number, if so the next READ statement reads the first item in the first DATA statement.

Example:

```
10 READ A, B, C
20 RESTORE
```

```

30 READ X, Y, Z
40 PRINT A, B, C
50 PRINT
60 PRINT X, Y, Z
70 DATA 10, 20, 30
80 DATA 40, 50, 60
90 END
RUN
10      20      30
10      20      30

```

Note: It is clear from the above example that the second DATA statement is not read by the second READ statement; rather RESTORE set the first DATA statement to be reread by the next READ statement.

2.16 INPUT Statement

This statement is used to input data from the user during the program execution.

Syntax:

- **INPUT** [;] [*prompt string*;] *comma-separated list of variables*
- **INPUT** [;] [*prompt string*,] *comma-separated list of variables*

Interpretation

Prompt string is the message that is displayed on the screen to assist the user to input correct data. We can specify more than one variable with a single INPUT statement. During program execution the values entered by the user are assigned to the corresponding variables according to the same sequence in which they are listed. The number of data items supplied by the user must be the same as the number of variables in the list. The type of each data item input must agree with the type specified by the variable name.

When a semicolon is used to separate the *prompt string* from the list of variables (as in the first syntax) a question mark (?) appears at the end of the prompt string. This question mark can be avoided by using a comma (as in the second syntax) instead of the semicolon.

Example:

The following figure shows a program that demonstrates the use of INPUT statement.

```

10 REM This program calculates the circumference of the circle
20 INPUT "Enter the radius of the circle >", RADIUS!
30 CIRCUMFERENCE# = 2 * 3.14 * RADIUS!
40 PRINT "Circumference = "; CIRCUMFERENCE#
50 END
Ok

RUN
Enter the radius of the circle >2.7
Circumference = 16.95600128173828
Ok

```

Fig.2.10: Example of INPUT statement

When an INPUT statement is encountered during program execution, the program halts, the prompt string is displayed, and the user types in the requested data. Strings that input to an INPUT statement need not be surrounded by quotation marks unless they contain commas or blanks.

2.17 PRINT Statement

This is the most frequently used statement in BASIC. Almost every program in BASIC makes use of it. All the programs we have seen so far in this text have used it in different ways. This statement is used to display text and numbers on the screen.

Syntax:

- PRINT [list of expressions][;]
- ?[list of expressions][;]

Interpretation

Expressions in the list may be numeric and/or string expressions, separated by commas, spaces, or semicolons. If we omit the list of expressions, a blank line is printed. We can also use a question mark (?) instead of PRINT statement. This will behave in the same way as the PRINT statement behaves.

When the two expressions in a PRINT statement are separated by a semicolon (;), the second expression is displayed on the screen just after the text of the first expression. GW-BASIC divides each line into print zone of 14 spaces. We can also use a comma (,) instead of semicolon to separate expressions; this causes the second expression to be displayed at the start of next zone.

Examples

The following figure shows the use print statement.

```

Ok
LIST
10 A$= "SINDH": B$= "PUNJAB": C$= "BALOCHISTAN": D$= "NWFP": E$= "KASHMIR"
20 PRINT A$: B$: C$: D$: E$
30 PRINT
40 PRINT A$, B$, C$, D$, E$
50 END
Ok

RUN
SINDHPUNJABBALOCHISTANNWFPKASHMIR

SINDH      PUNJAB      BALOCHISTAN      NWFP      KASHMIR
Ok

Ok
LIST
10 A=10 : B=20 : C=30
20 CITY$ = "Lahore": COUNTRY$ = "Pakistan"
30 PRINT A, B, C
40 PRINT A, B, C, CITY$, COUNTRY$
Ok

RUN
10      20      30
10      20      30      Lahore      Pakistan
Ok

```

Fig. 2.11: Examples of PRINT statement

2.18 PRINT USING Statement

This command is used to display numbers and strings on the screen in a specified format.

Syntax:

PRINT USING *string expressions*; *list of expressions* [;]

Interpretation

String expressions is a string literal or variable consisting of special formatting characters. The formatting characters determine the field and the format of printed strings or numbers. *List of expressions* consists of the string or numeric expressions separated by semicolons.

String Fields

The following three characters may be used to format the string field:

- **!** – Specifies that only the first character in the string is to be printed.
- **\n spaces ** – specifies that 2+n characters from the string are to be printed. If the backslashes are typed with no spaces, two characters are printed; if the backslashes are typed with one space, three characters are printed, and so on.

```
10 A$="Work": B$="Hard"
```

```
30 PRINT USING "!"; A$; B$
```

```
40 PRINT USING "\ "; A$; B$
```

```
50 PRINT USING "\ "; A$; B$;"!"
```

```
RUN
```

```
WH
```

```
WorkHard
```

```
WorkHard!!
```

- **&** – Specifies a variable length string field. When the field is specified with &, the string is output exactly as input.

Numeric Fields

The following special characters may be used to format the numeric field:

- **#** – A hash sign is used to represent each digit position. Digit positions are always filled. If the number to be printed has fewer digits than positions specified, the number is right-justified (preceded by spaces) in the field.
- A decimal point may be inserted at any position in the field. If the format string specifies that a digit is to precede the decimal point, the digit always is printed (as 0 if necessary). Numbers are rounded as necessary. For example:

```
> PRINT USING "###.##"; .85
```

```
> 0.85
```

```
> PRINT USING "####.##"; 254.753
```

- 254.75
- PRINT USING "###.###";10.2,5.3,66.789,.234
- 5.30 66.79 0.23

In the last example, three spaces were inserted at the end of the format string to separate the printed values on the line.

- A plus sign at the beginning or end of the format string causes the sign of the number (plus or minus) to be printed before or after the number.
- A comma to the left of the decimal point in the format string causes a comma to be printed to the left of every third digit to the left of the decimal point. A comma at the end of the format string is printed as part of the string.

Exercise

1. Fill in the blanks:

- (i) BASIC stands for _____.
- (ii) BASIC was developed by John Kemeny and Thomas Kurtz in _____ at Dartmouth College, USA.
- (iii) GW-BASIC shows a(n) _____ message, when loaded.
- (iv) In _____ mode, the commands are executed as they are typed.
- (v) Every statement in a BASIC program is preceded by a _____.
- (vi) A GW-BASIC program line can not have more than _____ characters.
- (vii) There can be maximum _____ lines in a GW-BASIC program.
- (viii) The default extension of a GW-BASIC program is _____.
- (ix) The words which have pre-defined meaning in a programming language are called _____.
- (x) _____ is an example of a non-executable statement.

2. Choose the correct answer:

- (i) GW-BASIC can operate in:

(a) One mode	(b) Two modes
(c) Three modes	(d) Several modes
- (ii) The maximum length of a variable name in GW-BASIC is:

(a) 31	(b) 32	(c) 40	(d) 45
--------	--------	--------	--------
- (iii) If two or more statements are written on a line, they must be separated by a:

(a) Colon	(b) Semi colon	(c) Comma	(d) Hyphen
-----------	----------------	-----------	------------
- (iv) Which of the following is a type declaration character for integer variables?

(a) !	(b) %	(c) #	(d) \$
-------	-------	-------	--------

- (v) Which of the following operators has the highest precedence?
 (a) ^ (b) * (c) + (d) =
- (vi) A variable name must start with a(n):
 (a) Alphabet (b) Underscore
 (c) Digit (d) Alphabet or Underscore
- (vii) Which of the following is a short key to run a program in GW-BASIC?
 (a) F4 (b) F3 (c) F2 (d) F1
- (viii) When a floating-point value is converted to an integer, the fractional part:
 (a) Truncated (b) Rounded off
 (c) May be truncated or rounded off (d) Conversion is impossible
- (ix) Which of the following statement temporarily stops the execution of a program?
 (a) BREAK (b) END (c) PAUSE (d) STOP
- (x) Which of the following command continues the program whose execution was terminated temporarily?
 (a) CONTINUE (b) CONT
 (c) RESTART (d) START
3. Write T for True and F for False statements.
- QBASIC provides a menu-driven environment.
 - Every BASIC command is preceded by a line number.
 - KILL command is used to terminate a running process on the system.
 - DELETE command is used to delete a file.
 - AUTO command is used to generate line numbers automatically.
 - ? can be used as a replacement of PRINT command.
 - In GW-BASIC, F1 key is used to get help.
 - A program must be loaded before execution.
 - READ statement gets data through keyboard.
 - The value of a constant can not be changed during program execution.
4. In how many modes, GW-BASIC can operate? Discuss briefly.
5. Describe rules of naming variable in GW-BASIC.
6. What are type declaration characters? Explain their uses with examples.
7. Briefly describe the uses of arithmetic, logical, and relational operators.
8. What does it mean by *type conversion*? Describe rules of type conversion in BASIC.
9. Write a program to read ten values specified in DATA statement, and display the sum of these values on the screen.
10. Answer the following short questions:
- Write the purpose of the function keys i.e., from F1 to F9 in GW-BASIC.
 - What does IDE stand for? Discuss features of GW-BASIC IDE.

- (iii) Explain the term 'Loading a program'. Why should a program be loaded before execution?
- (iv) Differentiate BASIC commands and statements.
- (v) What is the difference between CLEAR command and CLS command?
- (vi) Write the purpose and syntax of the following commands:
 (a) DELETE (b) KILL (c) FILES
 (d) LIST (e) LOAD (f) SYSTEM
 (g) NAME (h) RENUM (i) RUN
 (j) SAVE
- (vii) Briefly describe the structure of a BASIC program.
- (viii) Differentiate variable and constant.
- (ix) Write a program that asks for the name, roll number, class, section, and marks in different subjects of a student of class 10. The program should calculate and display total marks and percentage of the student. [Hint: use INPUT statement to get data from the user. Suppose total marks are 850]
- (x) Write a program to calculate the distance covered by a car moving at an average speed of $v \text{ ms}^{-1}$ in time t . The program should input average speed and time. [use INPUT statement to get the values for v and t . You have developed the algorithm for the program in the exercise of the previous chapter]
- (xi) Give an example to explain the use of comma (,) and semi colon (;) with PRINT statement.
11. Write a program to calculate the volume of a cylinder. The program should get the values for height of the cylinder and the radius of its base from the user through INPUT statement.
 [Hint: volume = $3.14 \times \text{radius} \times \text{radius} \times \text{height}$]
12. Write a program to compute the square of a given number. The program should get the number from the user through INPUT statement.
13. Write a program to calculate and print the sum and average of three numbers using LET statement.

Answers

1. (i) Beginners All purpose Symbolic Instruction Code.
 (ii) 1963 (iii) Ok (iv) Direct (v) Line number
 (vi) 255 (vii) 65529 (viii) bas (ix) Keywords/Reserve words
 (x) REM
2. (i) b (ii) c (iii) a (iv) b (v) a
 (vi) a (vii) c (viii) b (ix) d (x) b
3. (i) T (ii) F (iii) F (iv) F (v) T
 (vi) T (vii) F (viii) T (ix) F (x) T

CONTROL STRUCTURES

3.1 INTRODUCTION

Control structures control the flow of execution of a program. There are three types of control structures in BASIC; these are *sequence*, *selection* and *loop*. All programs use some of these control structures to implement the program logic.

So far we have been using only the *sequence* structure. In *sequence* structure instructions are executed according to the increasing order of their line numbers. So the instructions at smaller line numbers are always executed first, then instructions at greater line numbers. For example, the following program demonstrates the *sequence* structure:

```

10 REM This program does not transfer control to any statement
20 REM conditionally or unconditionally. The statements are executed
30 REM in the same sequence in which they are written.
40 R = 10.5
50 AREA = 3.14 * R * R
60 PRINT "Area = "; AREA
70 END

```

Output

346.185

In GW-BASIC, during the execution of a program the program control can be transferred from one part of the program to another conditionally or unconditionally. GW-BASIC provides statements for both types of transfer of control.

- In *unconditional transfer of control* the program control switches to a specific line by skipping one or more lines without any condition.
- In *conditional transfer of control* the program control switches to a specific line number by skipping one or more program lines depending on a certain condition.

3.2 UNCONDITIONAL TRANSFER OF CONTROL

The unconditional transfer of control causes a move of the control from one part of the program to the other without any condition. In GW-BASIC, the GOTO statement is used to implement unconditional transfer of control.

3.2.1 GOTO Statement

GOTO statement is used to unconditionally transfer control from a program line to a specified line out of the normal program sequence.

Syntax:

Line# GOTO line number

Interpretation

The *line number* is a valid line number in the program. The program control immediately jumps to the specified line number without testing any condition. This causes interruption in the normal program flow which is not considered a good practice in modern style of programming. That's why; in most of the programming languages the use of GOTO statement is discouraged. If there is an executable statement at the line number specified in GOTO statement, then this and the statements following it are executed. Otherwise, the program execution starts from the first executable statement after this statement.

Examples

Following examples demonstrate the use of GOTO statement:

```
10 READ A,B,X,Y
20 GOTO 60
30 LET X = X*X+A
40 LET Y = Y*Y + B
50 PRINT X, Y
60 REM Because of unconditional transfer of control, X and Y will
65 REM not be calculated in the above lines
70 LET X = A*X
80 LET Y = B*Y
90 PRINT X, Y
100 DATA 6,3,4,5
110 END
```

This shows after reading the values of A,B,X,Y, at line number 10, the control jumps to line number 60 where it encounters REM (which a non-executable statement). The control then moves to line number 70 where X and Y are calculated. Then at line number 90, the values of X and Y are printed and the program ends. Statements at line number 30, 40, and 50 are skipped without execution. If we want to execute 30, 40, and 50 then our program needs few more jump statements, i.e., GOTO statements. Consider the following program.

```
10 READ A, B, X, Y
20 GOTO 60
30 LET X = X*X+A
40 LET Y = Y*Y+B
50 PRINT X, Y
55 GOTO 100
60 REM Because of unconditional transfer of control, X and Y
65 REM will not be calculated in the above lines
```

```

70 LET X = A*X
80 LET Y = B*Y
90 PRINT X, Y
95 GO TO 30
100 DATA 6,3,4,5
110 END

```

Thus by introducing line number 55 and 95 all the statements are executed but this makes the program more complex. So line number 30 and 40 find the values of $X = X*X + A$, and $Y = Y*Y + B$ and line numbers 70 and so find values of $X = A*X$ and $Y = B*Y$.

3.3 Conditional Transfer of Control

The conditional transfer of control causes the switching of the control from one part of the program to the other depending on a certain condition. In GW-BASIC, there are many statements that conditionally transfer the control from one part of the program to the other. Here, we shall discuss them briefly.

3.3.1 ON...GOTO Statements

It is a multiple branching statement. Unlike GOTO statement which allows only one transfer point, the ON...GOTO statement can have more than one transfer points, thus providing multiple branching facility.

Syntax:

ON numeric variable or expression GOTO n1, n2, n3 ...

Interpretation

The *expression* is a valid BASIC expression and n1, n2, n3 ... are the valid line numbers in the program where the control will be transferred. The range of value of numeric variable or expression is 0 to 255.

If the value of *numeric variable* or *expression* is 1, the control will be transferred to line number n1, if the value is 2 then the control will be transferred to line number n2, if value is 3 then the control will be transferred to line number n3 and so on. If the value is less or more than the count of line numbers following the GOTO statement then an "OUT OF RANGE" error message will be displayed.

Example:

Let us consider a program to add, subtract, divide and multiply two numbers A and B.

```

5 INPUT A,B
10 INPUT " 1-ADD; 2-SUB; 3-MUL; 4-DIV"; N
20 ON N GOTO 30, 40, 50, 60
30 PRINT A+B : END
40 PRINT A-B : END

```

```
50 PRINT A*B : END
60 PRINT A/B : END
```

When line number 5 is executed, a question mark (?) appears on the screen. This is in fact the prompt where values for A and B are typed. Line number 10 displays the following message on the screen:

1-ADD, 2-SUB, 3-MUL, 4-DIV?

If we enter 1 then N takes the value of 1. If we enter 2 or 3 or 4, N will take the same value accordingly. If the value of N is 1 then control is transferred to line number 30, 2 then control is transferred to line number 40, 3 then control is transferred to line number 50, 4 then control is transferred to line number 60

On execution of line number 30 addition of A and B will be displayed on the screen and program will come to an end. Similarly, in line number 40, 50, 60 printing of the result of subtraction, multiplication and division will be displayed respectively.

There can be many ways to write a program. Let us write the same program (in the above example) in another way:

```
5 INPUT A, B
10 INPUT "1-ADD, 2-SUB, 3-MUL, 4-DIV"; N
20 ON N GOTO 30, 40, 50, 60
30 PRINT A+B : GOTO 70
40 PRINT A-B : GOTO 70
50 PRINT A*B : GOTO 70
60 PRINT A/B
70 END
```

3.3.2 ON ERROR GOTO Statement

This command enables error trapping feature of GWBASIC and specify the first line of error handling routine.

Syntax:

ON ERROR GOTO line number

Interpretation

line number is a valid line number in the program. Error handlers in GWBASIC are 'turned on' with an ON ERROR GOTO statement. It is often used at the beginning of the program so that errors occurred anywhere in the program can be trapped. When an error occurs during the program execution, the control immediately transfers to the specified line number. This line number specifies the beginning of a user-defined error handling routine which processes the error accordingly. The errors can be handled in direct as well as indirect mode.

In GW-BASIC, each possible error has been assigned a unique code. When an error occurs its code is assigned to a special variable, named ERR and the line number where the error was encountered is assigned to another special variable, named ERL. The ERR and ERL are reserved words.

GW-BASIC needs to know when it has finished handling an error. We can exit from an error handling routine using the RESUME, RESUME NEXT, RESUME line number or END statement. The RESUME statement returns the execution to the statement that caused the error and tries to execute it again. Hence we should only use this statement if the program or the user were somehow able to fix the problem immediately. RESUME NEXT resumes the execution from the statement immediately following the statement that caused error. And RESUME line number resumes the execution at the specified line number.

Example

```
10 ON ERROR GOTO 70
20 INPUT "Enter first No. ", n1%
30 INPUT "Enter second No. ", n2%
40 r% = n1% * n2%
50 PRINT "Result = "; r%
60 END
70 PRINT "Error Code = "; ERR
80 PRINT "Error is on line no. "; ERL
90 END
```

RUN

```
Enter first No. 400
Enter second No. 450
Error Code = 6
Error is on line no.40
```

Here, r is an integer variable and the result of $400 * 450$ can not be stored in it. Hence an overflow error occurs with code number 6, which is assigned to the variable ERR.

3.4 SELECTION STRUCTURE

A selection structure chooses which alternative program statement(s) to execute. In GW-BASIC, We have IF...THEN, and IF...THEN...ELSE statements to implement selection structure.

3.4.1 The IF... THEN Statement

The IF...THEN is a decision making statement, depending upon the decision, it can change the order of program execution. It is used to select a

path flow in a program based on a condition. A *condition* is an expression that either evaluates to true (usually represented by 1) or false (represented by 0).

Syntax:

- IF *expression* THEN *Statement*
 - IF *expression* THEN *line number*
- If the expression is true then either the statement at the specified *line number* or the statement following the THEN keyword is executed.

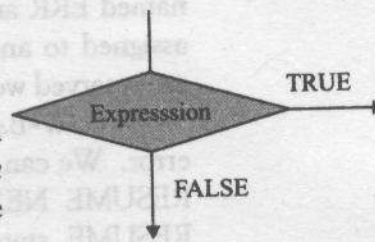


Fig. 3.1: Flowchart of IF...THEN statement

Example:

Write a program to find out the train fare depending on the Kilometers traveled. The minimum fare charged is Rs. 6.00. This minimum fare remains valid for 5 km traveling distance or less. After 5 km, 0.75 rupees per kilometer are added to the fare.

```

10 INPUT "KILOMETER", K
20 IF K <= 5 THEN PRINT "RS.6" : END
30 CHARGE! = 6 + (K-5) *.75!
40 PRINT CHARGE!
50 END
  
```

Here in line number 20, if K is less than or equal to 2 then it will print Rs. 6 and the program will END, in continuation to the line number 20 next statement is END. Otherwise if K is not less than or equal to 2 then line number 30,40 and then 50 will be executed.

3.3.2 The IF...THEN...ELSE Statement

The keyword ELSE is used to specify two different alternatives with IF statement. Based on a condition, one of the two alternatives is executed.

Syntax:

IF (<i>expression</i>) THEN Statements (<i>true task</i>) ELSE Statements (<i>false task</i>)	IF (<i>expression</i>) THEN <i>line number</i> ELSE Statements (<i>false task</i>)
--	--

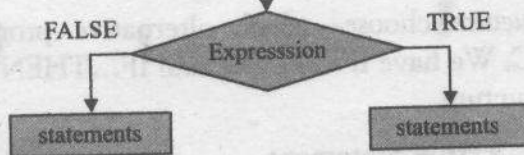


Fig. 3.2: Flowchart of IF...THEN...ELSE structure

The IF...THEN...ELSE statement is a decision making statement as it decides the path of the program. It helps in making

comparisons and testing whether a condition is true or not. IF is always followed by a valid BASIC condition or expression. If the condition is found true then the line number or Instruction after THEN is performed otherwise line number or instruction after ELSE is performed.

Example: Ages of different candidates appearing before a selection board of PIA are accepted through the keyboard. If the age is below 17 the candidate is not eligible for the announced post, otherwise he can appear for test and interview. We are asked to write a program for this problem.

```
10 INPUT "AGE"; A
20 IF A >= 17 THEN 30 ELSE 50
30 PRINT "Candidate is eligible"
40 GO TO 60
50 PRINT "Candidate is not eligible"
60 INPUT "Would you like to input again (Y/N)"; Y$
70 IF Y$ = "Y" THEN 10
80 END
```

The line number 10 will display the message AGE? on screen. The user enters the age (say 18). Line number 20 tests whether A is greater than or equal to 17 or not. Since A is equal to 18 (which is greater than 17), statement at line number 30 is executed. Line number 30 prints "Candidate is eligible"; line number 40 causes the control to pass to line number 60. Line number 60 causes the message "Would you like to input again (Y/N)?" We input either Y or N. In line number 70 if input is Y then control goes to line number 10, otherwise goes to the line number 80 i.e., END.

Now if at line number 60 we enter Y then control will pass again to line number 10, we give another age, say 13. In line number 20 value of A (i.e., 13) is not greater than 17, therefore ELSE part will be executed and control will go to line number 50. Line number 50 will print "Candidate is not eligible". Then line number 60 as before will be executed. In this way a large number of candidates' age can be tested. When we want to stop we should input N in line 60 for Y\$.

Use of Logical Operators

We have studied the three logical operators i.e., AND, OR, and NOT. These operators play an important role in constructing conditions to be used with IF statement. Until now, we have been using simple conditions with IF statement. In this section, we shall observe how complex program logic can be simplified using logical operators.

Example: Let us consider a program to find the smallest of three given numbers.

```
10 INPUT A,B,C
20 IF A<B AND A<C THEN 50 ELSE 30
30 IF B<A AND B<C THEN 60 ELSE 40
40 IF C<A AND C<B THEN 70
50 PRINT "A Is The Smallest Number": GO TO 80
60 PRINT "B Is The Smallest Number": GO TO 80
70 PRINT "C Is The Smallest Number"
80 INPUT "Would you like to input again (Y/N)"; Y$
90 IF Y$ = "Y" THEN 10
100 END
```

There are many such situations where the use of logical operators would simplify the program logic. We just have to concentrate on the underlying problem. In this book, we shall see more examples of using logical operators in next chapters.

3.5 LOOPS

We often face problems whose solution may require executing a set of statements repeatedly. In such situations, we need a structure that would allow repeating a set of statements up to fixed number of times or until a certain criterion is satisfied. Loop structure fulfills this basic requirement.

3.5.1 FOR...NEXT Loop

When it is known in advance how many times the loop must be repeated the FOR...NEXT loop is the most effective option. FOR loop is used to repeat a set of statements to a specific number of times.

Syntax:

```
FOR variable = x TO y [STEP z]
```

```
NEXT [variable]
```

Interpretation

The numeric *variable* name following FOR is called the *loop control variable* or *loop variable*, *x* and *y* are numeric constants where *x* gives the initial or starting value of the loop and *y* gives the final value, *z* followed by keyword STEP gives the increment in *x* till *y* is reached. The increment can be negative also.

x, *y* and *z* can be numeric variable names. In such cases their numeric values should be assigned before the starting of the loop, i.e., before coming to FOR statement. The keyword NEXT should have the same *control variable* as followed by the keyword FOR.

Example: Let us consider a Program that prints the sum of the following series:

```
2,4,6,8,.....100
10  N = 0
20  FOR I = 2 TO 100 STEP 2
30  N = N + I
40  NEXT I
50  PRINT " SUM OF SERIES = "; N
60  END
```

3.5.2 WHILE...WEND Loop

The While loop keeps repeating an action until an associated condition becomes false. This is useful where the programmer does not know in advance how many times the loop will be executed.

Syntax:

```
WHILE expression
```

```
[loop statements]
```

```
WEND
```

Interpretation

The **expression** in the While loop controls the loop repetition. The statements, which are executed when the given condition is true; form the *body of the loop*. The body of the loop is executed until the condition is **true**. As soon as it becomes false, the loop terminates immediately and the program control transfers to the statement next to the WEND statement.

It must be noted that the loop control variable in WHILE loop is always initialized outside the body of the loop and is incremented or decremented, according to the program logic, inside the body of the loop. Whereas in FOR loop, the loop control variable is initialized and incremented or decremented within the FOR statement.

Example: Write a program to print digits from 1 to 10

```
10  N = 1
20  WHILE N <= 10
30  PRINT N
40  N = N + 1
50  WEND
60  END
```

4.5.3 Nested Loop

Inside a loop (FOR or WHILE) there can be one or more loops (FOR or WHILE), such type of structure is known as nested loop.

Example: Suppose we want to print the output in the following format:

```
*****
*****
***
**
*
10  FOR Y = 5 TO 1 STEP-1
20  FOR X = 1 TO Y
30  PRINT "*";
40  NEXT X
50  PRINT
60  NEXT Y
70  END
```

This program contains two loops: the outer loop is from the line number 10 to 60 and inner loop from line numbers 20 to 40. In line number 10, initially Y is assigned a value 5. Since the value of Y is greater than 1, control is transferred to line number 20, which causes the inner loop to execute 5 times resulting into printing of 5 stars (*) in one row. The statement at line number 50 will transfer the printer control to the beginning of the next line. When line 60 is encountered, the control goes back to line number 10. Now the value of Y becomes 4 and once again the inner loop is executed 4 times resulting in printing of 4 (*) stars in second row. This process will continue till the value of Y becomes 1. After that it will come to an end.

Exercise

1. Fill in the blanks:

- (i) It is recommended to avoid the use of _____ statement.
- (ii) GOTO statement provides _____ transfer point.
- (iii) The FOR...NEXT loop transfers control based on a _____.
- (iv) Maximum _____ transfer points can be specified with ON...GOTO statement.
- (v) ON ERROR GOTO statement _____ error trapping features of GW-BASIC.
- (vi) In GW-BASIC, every error is assigned a _____ code.
- (vii) A special variable named _____ is assigned an error code on occurrence of an error in BASIC.
- (viii) A special variable named _____ is assigned the line number where an error occurs in a BASIC program.

- (ix) When an error occurs in a BASIC program, the control may transfer to a(n) _____ routine.
- (x) A _____ structure chooses the alternative program statement(s) to execute.
2. Choose the correct answer.
- (i) Which of the following is not a logical operator:
 (a) AND (b) OR (c) NEITHER (d) NOT
- (ii) Which of the following can not be used to exit from an error handling routine:
 (a) RESUME (b) RESUME NEXT (c) END (d) STOP
- (iii) Which one is a multiple branching statement?
 (a) IF...ELSE (b) GOTO
 (c) ON....GOTO (d) ON ERROR GOTO line_number
- (iv) If the integer value of the numeric expression following the keyword ON, in ON...GOTO statement, is greater than 255, which type of error occurs?
 (a) Syntax error (b) Logical error
 (c) Runtime error (d) Its not an error
- (v) FOR...NEXT is used to implement:
 (a) Iteration (b) Selection
 (c) Sequence (d) All of the above
3. Write T for True and F for False statements.
- (i) The control can exit from an error handling routine using a RESUME statement.
- (ii) There is no difference between END and STOP statements.
- (iii) A false condition evaluates to NULL.
- (iv) A WHILE...WEND loop executes as long as the specified condition is true.
- (v) In WHILE...WEND loop, the loop control variable is always updated outside the body of the loop.
- (vi) Specifying an IF statement within the body of a loop is referred to as a nested loop.
- (vii) The GOTO statement increases the complexity of the program.
- (viii) A condition specified in IF statement may neither be true nor false.
- (ix) There is no need to update the loop control variable in FOR...NEXT loop, the loop itself manage it.
- (x) Specifying an IF statement within the block of another IF statement causes a syntax error.

4. Define control structure. How many control structures are available in BASIC, discuss briefly.
5. Define nested loop. Write the syntax of FOR...NEXT and WHILE...WEND loop and explain with examples.
6. What does it mean by transfer of control? Briefly describe conditional and unconditional transfer of control in BASIC.
7. Differentiate WHILE...WEND and FOR...NEXT loop. Which one is better in a situation where you don't know the number of iteration prior to the execution of the loop?
8. Write a program to calculate the area of a triangle. The program should get the values for base and altitude of the triangle from the user, and display the result. [Hint: $\text{area} = \frac{1}{2} \times \text{base} \times \text{altitude}$]
9. Write a program to calculate area and circumference of a circle. The program should get the radius of the circle from the user and display result. [Hint: $\text{area} = 3.14 \times \text{radius} \times \text{radius}$, and $\text{circumference} = 2 \times 3.14 \times \text{radius}$]
10. Write a program to print first ten odd numbers using WHILE...WEND loop.
11. Write a program to print the sum of squares of first five even numbers using FOR...NEXT loop.
12. Write a program to find the larger of two numbers. The program should get the numbers from the user.
13. Write a program to print the table of a given number. The program should get the number from the user.
14. Write a program that should accept obtained marks of a student in an examination. It should then calculate the percentage and assign a grade to the student. The grade should be assigned according to the following criteria.

Percentage	Grade
≥ 80	A1
≥ 70 , but < 80	A
≥ 60 , but < 70	B
≥ 50 , but < 60	C
≥ 40 , but < 50	D
< 40	F

Answers

1. (i) GOTO (ii) One (iii) Condition (iv) 255 (v) Enables
(vi) Unique (vii) ERR (viii) ERL (ix) Error handling (x) Selection
2. (i) c (ii) d (iii) c (iv) c (v) a
3. (i) T (ii) F (iii) F (iv) T (v) F
(vi) F (vii) T (viii) F (ix) T (x) F

ARRAYS

4.1 INTRODUCTION

In previous chapter of this book, we have written simple programs in which less number of variables has been used as numeric or string variables. If we have to use hundred or thousand different variables in a program, it will become difficult to handle them. For example a program has to store names of hundred students in a computer. It needs hundred different variable names. So an array can solve these type of problems by defining names instead of contiguous memory location defining several names for string different values. Arrays are used to process a large amount of data of same type.

4.1.1 What is an Array?

An array is a collection of variables that can store data of same type. Each memory location holds a single value which is called an element of an array. The array is represented in the computer's memory by a set of consecutive memory locations. The memory locations are referred to as elements of the array. Each array is given a name and the elements of the array are accessed with reference to their position or location number. This position number is called index or subscript. The subscript or index value is written in parentheses with the name of array. The first element of the array has an index value of 0 unless specified otherwise, and the index is incremented for each next element.

General format Syntax: *array name (size of array)*

An array is referred by its name followed by a subscript enclosed in parentheses. Where *array name (array variable)* is the variable name of the array, *size of array (subscript/index value)* is specified number of data items that it will contain. Each element is given a unique storage location in the array, and a subscript or index value is used to identify the position of a particular element. The subscript, which may be an integral expression or simple variable, is enclosed in parentheses after the array name. We have only dealt with unsubscripted variables, so far which are simple variables that are only capable of storing one value. An array is called a subscripted variable because when we need to access a certain element; we must use a subscript to point to that element so we can differentiate it from the rest. All array elements in an array have the same variable name, which is also the name of the entire array. An array variable can be numeric or string. For example of string N\$ is consisting of five elements N\$(0), N\$(1),....N\$(4), it can be represented as below:

N\$(0)	Saleem
N\$(1)	Asma
N\$(2)	Majeed
N\$(3)	Bushra
N\$(4)	Mehmood

where 0, 1, 24 are the subscripts used to identify the array location, so that N\$(0), N\$(1), ...N\$(4), are addresses in the memory where the values will be stored. Values can be assigned to the subscripted variables by LET statement, READ ...DATA statement or INPUT statement. The "size of subscript / index value" can be a variable or number or a numeric expression.

4.1.2 Filling and Printing of an Array

Data (String and numbers) is entered in an array by using LET, READ or INPUT statements. Data may be assigned to the subscripted variables in an array. The following program is to store the string values. It is illustrated by both READ... DATA statement and INPUT statement.

Example 1:

```

10 FOR K= 1 TO 4
20 READ N$(K)
30 PRINT N$(K)
40 NEXT K
50 DATA Shaista, Mehmood, Saleem, Hina
60 END
RUN
Shaista,
Mehmood
Saleem
Hina

```

Note: index value can be given a non- zero integral value but by default it is zero.

When this program is RUN, the subscripted variable N\$(K) automatically represents each of four elements in the array, the first time through the loop when K=1 the string Shaista is assigned to N\$(1) and so on.

Example 2:

```

10 FOR K= 1 TO 4
20 INPUT "Enter the name of student", N$(K)
30 NEXT K
40 FOR K = 1 TO 4
50 PRINT N$(K)
60 NEXT K
70 END

```


RUN

Enter the name of student Shaista

Enter the name of student Mehmood

Enter the name of student Saleem

Enter the name of student Hina

Shaista,

Mehmood

Saleem

Hina

4.1.3 DIM Statement

By default GW-Basic provides an array for string 10 elements, from subscripts 0 to 9.

We have to use the DIM statement to specify a maximum subscript different. If subscript greater than the maximum specified is used, a "Subscript out of range" error occurs. The maximum number of dimensions for an array is 255.

Syntax:

Line No. DIM subscripted variable1, subscripted variable2...

The keyword DIM is a shortened form of the word **dimension**. There is no DIM statement in the above programs because these programs handle less than 11 values.

DIM is used to create memory variable. It specifies the maximum value for array variable subscript and allocates storage accordingly. When subscript variables are used in a program, certain information must be considered before applying.

- Name of subscript variable
- Size of subscript variable
- We can define more than one array variable in one DIM statement (as mentioned in above syntax)
- Subscripted variable may be string or numeric.

The following problem can find the largest number from the list of numbers, which is given by the user.

Example 3:

```
10 DIM NUM(100)
20 INPUT "How many numbers you want to enter; Max: 100: ",LIMIT
30 FOR I= 1 TO LIMIT
40 INPUT "Enter any number", NUM(I)
50 NEXT I
60 LARGE = NUM(I)
```

```

70 FOR I = 1 TO LIMIT
80 IF LARGE < NUM(I) THEN LARGE = NUM(I)
90 NEXT I
100 PRINT "Largest number of list is "; LARGE
110 END
RUN

```

```

How many numbers you want to enter; Max: 100:      14
Enter any number      56
Enter any number      88
Enter any number      2
Enter any number      69
Enter any number      5
Enter any number      14
Enter any number      34
Enter any number      55
Enter any number      76
Enter any number      54
Enter any number      35
Enter any number      29
Enter any number      81
Enter any number      7
Largest number of list is 88

```

4.2 Types of Array

Array can be divided in two major categories.

- One-dimensional array
- Two-dimensional array

4.2.1 One-Dimensional Array

One-dimensional array is also known as linear array or vector array. It consists of only one row or column. It is also called 1-D array. For example, a class of 5 students has taken marks in a subject and wants to find the average of marks of a subject. The following problem can find the average of marks in a subject for the students of a class. The general syntax to declare One-Dimensional array is:

Line No. DIM array name/ variable (n)

Where *array name (variable)* represents the name of array variable, *n* represents the size of elements.

Example 4:

```

10 FOR I= 1 TO 5
20 READ MARK (I)
30 SUM=SUM + MARK(I)

```

```

40 NEXT I
50 AVG = SUM / 5
60 FOR I = 1 TO 5
70 PRINT MARK(I)
80 NEXT I
90 PRINT "SUM OF MARKS OF A CLASS I N A SUBJECT =
"; SUM
100 PRINT "AVERAGE OF MARKS = "; AVG
110 DATA 88, 66, 49, 55, 78
120 END
RUN
88
66
49
55
78
SUM OF MARKS OF A CLASS I N A SUBJECT = 336
AVERAGE OF MARKS = 67.2

```

4.2.3 Two-Dimensional Array

The two dimensional array consists of rows and columns. It is also known as table or matrix. Two-Dimensional array is also defined as: array of One-Dimensional arrays. The element of Two-Dimensional array is referenced by two index values. One index value represents the row and the second represents the column. An array that requires two subscripts to identify a particular element is also known as the double-subscripted array. For example if **A** is a two-dimensional array having 4 rows and 3 columns, its first element is **A (0,0)** and the last element is **A(3,2)**.

The general syntax to declare Two-Dimensional array is:

Line No DIM array variable (row, col)

where, array variable represents the name of the two-dimensional array. "row" represents the total number of rows of table. It is an unsigned number. "col" represents the total number of columns of the table. For example, to declare **A** having 4 rows and 3 columns, the declaration statement is written as: **A(3,2)** the total number of elements of the above table "A" are $4 \times 3 = 12$.

R/C	0	1	2
0	A(0,0)	A(0,1)	A(0,2)
1	A(1,0)	A(1,1)	A(1,2)
2	A(2,0)	A(2,1)	A(2,2)
3	A(3,0)	A(3,1)	A(3,2)

The above table has four rows and three columns. It can be read and processed in a two – dimensional array.

Filling and Printing of Two –Dimensional Arrays

Data is entered into individual elements of a two-dimensional array. To enter data, the element is referenced by its index or subscript value. Similarly, data is retrieved from an array from individual elements of the array. Usually, nested loops are used to access elements of the two-dimensional array. The following example inputs data into two array and then prints out the sum of array on the computer screen in tabular form.

Nested loop has been used to enter data into the elements of the table. The outer loop has been used to change the index values of rows and inner loop has been used to change the index values of columns. Similar method is used to print out data from the elements of the array.

Example 6:

```
10 DIM A(2,2), B(2,2), Z(2,2)
20 FOR R = 1 TO 2
30 FOR C = 1 TO 2
40 READ A(R,C), B(R,C)
50 Z(R,C) = A(R,C) + B(R,C)
60 PRINT Z(R,C),
70 NEXT C
80 PRINT
90 NEXT R
100 DATA 8,4,3,5
110 DATA 6,4,5,5
120 END
RUN
12      8
10     10
```

Array Manipulation

There are different operations can be performed by using array, like searching a particular element in an array, matching elements from two different arrays, sorting array, finding a largest and smallest number from an array and rearranging the array.

0	A(0,0)	A(0,1)	A(0,2)
1	A(1,0)	A(1,1)	A(1,2)
2	A(2,0)	A(2,1)	A(2,2)
3	A(3,0)	A(3,1)	A(3,2)

Exercise

1. Fill in the Blanks:

- (i) Two-dimensional array is also known as _____.
- (ii) Rearranging the list of an array is called _____.
- (iii) The DIM statement is an optional if the number of memory location is less than _____ locations.
- (iv) An array is a collection of _____ variables.
- (v) In two-dimensional array the subscripts are separated by _____.
- (vi) The minimum value for a subscript is always assumed to be _____.
- (vii) Z(2,2) is an example of _____ dimension array.
- (viii) A string array is declared by using its name, preceded by _____ sign.
- (ix) A collection of subscripted variables with same valuable name is _____.
- (x) The values given in the parenthesis in an array are called _____.

2. Choose the correct answer.

- (i) There are _____ types of array.
(a) 1 (b) 2 (c) 3 (d) 4
- (ii) The statement x(30) will reserve _____ memory locations:
(a) 29 (b) 30 (c) 31 (d) None of the above
- (iii) In two dimension array, when dimension is not mentioned, the array should not have more than _____ elements:
(a) 10 (b) 100 (c) 110 (d) 121
- (iv) Which of the following statement is used to find the largest value from an array?
(a) INPUT (b) READ...DATA
(c) ON-ERROR - GOTO (d) None of them
- (v) Which of the following is not a valid subscript?
(a) NUM(10) (b) A(2) (c) B(4) (d) A(-2)
- (vi) An element of an array is mentioned by its:
(a) Subscript (b) array (c) object (d) name of element
- (vii) Dimension statement uses the keyword _____.
(a) DMS (b) DS (c) DIM (d) DM
- (viii) Maximum number of elements per dimension is:
(a) 10 (b) 255 (c) 32767 (d) None of them
- (ix) The doubly subscripted variable p(3,2) specifies the data element present in
(a) Column 3 and Row 2 (b) Column 3 and Column 2
(c) Column 2 and Row 3 (d) Row 3 and Row 2
- (x) The statement DIM C(30,50) would reserves.
(a) 80 Locations (b) 1500 Locations
(c) 1800 Locations (d) 150 Locations

3. Write T for True and F for False statements:
- An array is a collection of subscripted variables with the different variable names.
 - An array is a set of unlike variables.
 - Items in a list are represented by a single variable name as array.
 - In programming, lists and table are called array.
 - 12 memory spaces are reserved for the statement DIM P(4,3).
 - DIM is a reserved word.
 - One demission array is also known as table.
 - Once an array is dimensioned, cannot be re-dimensioned with in the program.
 - The biggest number that BASIC automatically assigns as a subscript is 100.
 - Two dimensional arrays are named the same way as one dimensional arrays.
- What is meant by DIM statement?
 - Describe the use of subscript variable in array.
 - How would you Fill and Print the array?
 - What is meant by Manipulation of array?
 - Differentiate between 1-D-Array and 2-D-Array.
 - Describe about printing two dimensional array with the help of an example.
 - Write a program in BASIC to enter integer type data into an array and then to print the values in reverse order.
 - Differentiate between simple and subscript variable.
 - Draw a flowchart for the Q.NO. 16 program.
 - Write an algorithm to sum array A elements and array B elements.
 - Write a program to print a list of odd numbers from the given numbers.
6, 42, 4, 77, 32, 9, 21, 22, 8, 45, 15, 46
 - Write a program that read an array N with 20 numbers and find the product of the elements of array.
 - Write a program that read an array Z having 12 numbers given by user then print the sum and average of all array elements.
 - Find out the errors in the following program segments if any.

(a) 10 DIM N\$(10)	(b) 10 FOR J = K TO 15
20 FOR K= 4 TO 15	20 K(J) = J
30 INPUT N\$	30 PRINT K(J)
40 NEXT I	40 NEXT J
 - Write a program to sort the list of 20 names in descending order.

Answers

- | | | | | |
|--------------|--------------|-----------|----------------|----------------|
| 1. (i) Table | (ii) Sorting | (iii) 11 | (iv) Subscript | (v) Comma |
| (vi) 0 | (vii) Tow | (viii) \$ | (ix) Array | (x) Subscripts |
| 2. (i) b | (ii) c | (iii) a | (iv) d | (v) d |
| (vi) a | (vii) c | (viii) b | (ix) c | (x) b |
| 3. (i) F | (ii) F | (iii) T | (iv) T | (v) F |
| (vi) T | (vii) F | (viii) F | (ix) F | (x) T |

SUB-PROGRAM AND FILE HANDLING

5.1 INTRODUCTION

As a program gets longer, it becomes more difficult to handle most computer languages excluding GW-Basic features that facilitates such situation. A separate large program is divided into smaller, manageable parts called subprogram or modules. It is designed to perform a specific task and return a value. In BASIC, there are two types of subprogram: standard or "built-in" and user-defined. Intrinsic Function or built-in functions are provided by the BASIC, and allows the programmer to use them so that s(he) does not have to write code to handle certain situations. A user-defined function is written and specified by the programmer to accomplish a particular task. The function will always return a value to the "calling" module. Let us begin with the standard functions.

As previously mentioned, standard functions are provided by BASIC and simply have to be "called" upon for use. The called function will have a particular name followed by parentheses. An argument is "passed" to the function by inserting a constant, variable, expression, or another function inside the parentheses. The argument is what the function will be operating on. A function if used has highest priority in a statement so therefore it will be evaluated before anything else in the statement.

5.1.1. Built-in-Function

These functions perform operations on their operands and return values. Basically these are programs that have been written by developers of language and have been incorporated in it. These functions are known as built in functions or standard functions or Library functions or Intrinsic functions. BASIC has many Pre-defined functions built into the language that can be used by "calling" them. Basic library functions are divided into two general categories, numeric functions and String.

a) Numeric Functions

These functions are applicable on numeric values only, and produce the numeric results. There are too many functions available, but here most important functions will be discussed.

1. ABS FUNCTION

PURPOSE: The purpose of ABS function is to return the absolute value of the expression x i.e., the value without any sign.

FORMAT: ABS (x)

Example:

```
10 PRINT ABS (-15)
20 PRINT ABS (-12.45)
RUN
15
12.45
```

2. INT FUNCTION

PURPOSE: Returns the lowest integer less than or equal to x. In case of whole number it returns the same number.

FORMAT: INT (x)

Example:

```
10 J= INT (3.9999)
20 PRINT J
RUN
3
```

3. SQR FUNCTION

PURPOSE: Returns the square root of a positive number x. x must be greater than or equal to 0.

FORMAT: SQR (x), when $x \geq 0$

Examples:

```
10 FOR X=10 TO 25 STEP 5
20 PRINT X, SQR(X)
30 NEXT X
RUN
10 3.162278
15 3.872984
20 4.472136
25 5
```

4. SIN FUNCTION

PURPOSE: The purpose of SIN function is to find the trigonometric ratio called sine of an angle x expressed in radians. The following formula is used for converting angle in radians.

$SIN(x * \pi / 180)$

SIN(x) is calculated in single precision

FORMAT: SIN (x)

Example:

```
10 PI= 3.142857
20 PRINT SIN (PI * 30/180)
```


RUN
0.9999

Similarly there are other trigonometric functions given in the following list

Function	BASIC equivalent
Cosine	COS(x)
Secant	SEC(x)
Cosecant	COSEC(x)
Tangent	TAN(x)
Cotangent	COT(x)

5. FIX FUNCTION

PURPOSE: The purpose of this function is to obtain an integer value by simply dropping of the decimal part. Fix function dose not round the number.

FORMAT: FIX(x)

Example:

```
PRINT FIX (-7.09)
RUN
-7
```

6. TAB FUNCTION

PURPOSE: To print at certain column position x on the screen. If the current print position is already beyond space x, TAB goes to given position on the next line. Space 0 is the leftmost position. The rightmost position is the screen width. X must be within the range of 1 to 255.

FORMAT: TAB (x)

Example:

```
PRINT "PAKISTAN" TAB(2) "IS MY" TAB(4)
"COUNTRY"
RUN
PAKISTAN      IS MY                COUNTRY
```

7. RND FUNCTION

PURPOSE: To return a random number between 0 and 1.

FORMAT: RND [(x)]

The same sequence of random number is generated each time the problem is run unless the random number generator is reseeded. If x is equal to zero, then the last number is repeated. To get a random number within the range of zero through n, use the following formula:

$$\text{INT}(\text{RND}*(n+1))$$

Example:

```

10 FOR I = 1 to 5
20 PRINT INT (RND *101),
30 Next I
40 END
RUN
53 30 31 51 5

```

8. LOG FUNCTION

PURPOSE: To return a natural logarithm (LOG in BASIC is a logarithm to the base $e = 2.718282$).

FORMAT: LOG(x)

Example:

```

PRINT LOG (10)
Run
2.302585

```

9. SPC FUNCTION

PURPOSE: Skips x spaces in a PRINT statement. SPC may only be used with PRINT and LPRINT statement. The argument n must be in the range 0 to 255. If x is greater than the defined width of the printer or the screen, the value used will be $n \text{ MOD width}$.

FORMAT: SPC (x)

Example:

```

PRINT "OVER" SPC(15) "THERE"
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
OVER      THERE

```

10. BEEP Function

PURPOSE : To sound the speaker at 800 Hz (800 cycles per second) for one-quarter of a second

FORMAT: BEEP

Example:

```

BEEP (type BEEP at command level)

```

11. DATE\$ Function

This statement is used to set or retrieve the current date.

FORMAT:

DATE\$ = v\$

OR

v\$ = DATE\$

where, v\$ is a valid string literal or variable.

```
Ok
PRINT DATE$
11-14-2005
Ok
DATE$ = "12-03-2005
Ok
PRINT DATE$
12-03-2005
Ok
```

Fig. 5.1: Example of DATE\$ function

v\$ can be any of the following formats when assigning the date:

mm-dd-yy
mm/dd/yy
mm-dd-yyyy
mm/dd/yyyy

Examples:

Figure 5.1 shows the use of DATE\$ statement

b). SRTING FUNCTIONS:

String functions are used to process character strings and produce the numeric values or string values.

1. LEN FUNCTION

PURPOSE: To return the number of characters in string x\$. The x\$ is any string expression. Nonprinting characters and blanks are counted in the number of characters.

FORMAT: LEN(x\$)

Example:

```
10 A$ = "PAKISTAN"
```

```
20 PRINT LEN (A$)
```

Run

8

2. VAL FUNCTION

PURPOSE: To return the numerical value of string x\$. The VAL function also strips leading blanks, tabs and line feeds from the argument string. If the first character of x\$ is not numeric, the VAL(x\$) function will return zero.

FORMAT: VAL(x\$)

Example:

```
10 PRINT VAL ("78, city Lahore")
RUN
78
```

In this example VAL is used to extract the number from an address.

3. MID\$ FUNCTION

PURPOSE: To return the requested part from of a given string. n is an integer expression in the range 1 to 255, m is an integer expression in the range 0 to 255. The function returns a string of length m characters from X\$ beginning with the nth characters. If m is omitted or if fewer than m characters are to the right of n, all right most characters beginning with n function characters are returned. If m is equal to 0, or if n is greater than LEN(x\$), then MID\$ returns as null string.
FORMAT: V\$ = MID\$(x\$,n[,m])

Example:

```
10 A$ = "WE LOVE PAKISTAN"
PRINT MID$(A$, 4, 4)
PRINT MID$(A$, 9, 8)
RUN
LOVE
PAKISTAN
```

4. SPACE\$ FUNCTION

Purpose: To return a string of x spaces. x is rounded to an integer and must be within the range of 0 to 255.
FORMAT: SPACE\$(x)

Examples:

```
10 FOR N=1 TO 5
20 X$=SPACE$(N)
30 PRINT X$; N
40 NEXT N
RUN
1
2
3
4
5
```

Line 20 adds one space for each iteration.

5. RIGHT\$ FUNCTION

PURPOSE: To return the specified right most characters of strings x\$.

FORMAT: RIGHT\$ (x\$, i)

If i is equal to or greater than LEN(x\$), RIGHT\$ returns x\$. If i equals zero, the null string (length zero) is returned.

Example:

```
10 A$="Disk Operator"
20 PRINT RIGHT$(A$,5)
RUN
rator
```

6. LEFT\$ FUNCTION

PURPOSE: To return a string that comprises the leftmost n characters of x\$.

FORMAT: LEFT\$ (x\$, n)

n must be with in the range of 0 to 255. If n is greater than LEN\$(x\$), the entire (x\$) is returned. If n equals zero, the null string (length zero) is returned.

Example:

```
10 A$="Disk Operator"
20 PRINT LEFT$(A$,4)
RUN
Disk
```

7. CHR\$ FUNCTION

PURPOSE: To converts an ASCII Code value to its equivalent character. The Function CHR serves the opposite to ASC and is useful in getting output ASCII control characters. The PRINT statement is used to print these characters.

FORMAT: CHR\$ (n)

Example:

```
10 PRINT CHR$(65)
RUN
A
```

5.1.2. User-Defined Functions

A user-defined function is completely defined and customized by the programmer to solve some problems. Functions that we write are called *user-define functions*. User-defined functions return a single value and are generally used to perform an operation that will be needed numerous of times in a program. In BASIC, user-defined functions are referred to as procedures; similar to SUB procedures except function procedures return one value. The

user can define the function with **DEF FN Statement**. These functions are defined only for the life of a given program and are not part of the BASIC language.

A single line function can be defined by DEF FN statement that is executing same codes more than once in the different place of program.

Syntax: **Line No** **DEF FN name [arguments] expression**

Where "name" must be a valid variable name. This name preceded by FN becomes the name of function.

"arguments" consist of those variable names in the function definition that is to be replaced when the function is called. The items in the list are separated by commas.

"expression" is an expression that performs the operation of the function. It is limited to single statement.

The variables in the argument represent, on a one-to-one basis, the argument variables or values that are to be given in the function call.

User-defined functions may be numeric or string. If a type is specified in the function name, the value of the expression is forced to that type before it is returned to the calling statement. If a type is specified in the function name and the argument type does not match, a "Type Mismatch" error occurs.

A user-defined function may be defined more than once in a program by repeating the DEF FN statement.

A DEF FN statement must be executed before the function it defines may be called. If a function is called before it has been defined, an "Undefined User Function" error occurs.

DEF FN is illegal in the direct mode. Recursive functions are not supported in the DEF FN statement. DEF FN can define either numeric or string function. DEF FN returns a string if name is string variable name and a numeric value if name is a numeric variable name.

Example:

```
10  REM
20  DEF FNX(Y) = (Y ^ 3 + Y ^ 2) / Y
30  INPUT "Enter any two numbers", A, B
40  C = DEF FNX(A) + DEF FNX(B)
50  PRINT C
RUN
Enter any two numbers      4      5
50
```

Example:

```
10  CLS
20  PI = 3.1415
30  DEF FNR(X) = PI * X ^ 2
```

```

40 INPUT "Radius="; RAD
50 PRINT "Circle Area is "; FNR(RAD)
60 END
RUN
Radius = 2
Circle Area is 12.556697

```

5.2 Subroutines

A *subroutine* is a self-contained set of statements that can be used from anywhere in a program. The subroutine performs its specific task, and then returns control to the part of the program that calls the subroutine. Sometime, it is more convenient to structure a sequence of statements as a subroutine than as a function. Subroutines are similar to functions in the sense that they can be referenced from other places in a program. Unlike a function, however, a subroutine is not given a name, and it can be used to determine more than one numeric and or string quantity. In BASIC language GOSUB-RETURN and ON-GOSUB statements are used for subroutine.

GOSUB ... RETURN Statement

Purpose: To branch to, and return from, a subroutine.

A subroutine may be called any number of times in a program, and a subroutine may be called from within another subroutine. Such nesting of subroutines is limited only by available memory.

A RETURN statement in a subroutine causes GW-BASIC to return to the statement following the most recent GOSUB statement. A subroutine can contain more than one RETURN statement, should logic dictate a RETURN at different points in the subroutine.

Subroutines can appear anywhere in the program, but must be readily distinguishable from the main program.

To prevent inadvertent entry, precede the subroutine by a STOP, END, or GOTO statement to direct program control around the subroutine.

Syntax:

```

Line No          GOSUB line number
.
.
.

```

```

Line No          RETURN [line number]

```

line number is the starting line number of the subroutine and optional in RETURN statement.

Examples:

```

10 GOSUB 40
20 PRINT "BACK FROM SUBROUTINE"
30 END

```

```
40 PRINT "SUBROUTINE";
50 PRINT " IN";
60 PRINT " PROGRESS"
70 RETURN
RUN
SUBROUTINE IN PROGRESS
BACK FROM SUBROUTINE
```

The END statement in line 30 prevents re-execution of the subroutine.

5.3 FILE HANDLING

A computer can process large amount of data. To know how data can be written and read from files, we must know a brief concept about files. Thus we need to know about data items and records.

Characters consist of alphabets, digits and special characters. These are represented inside the computer as a sequence of 1s and 0s.

Data Fields are group of related characters to have a unit of information. For example a student name, his rolls no are two different data fields.

Record a group of related fields is called record. For example the combination of name, father name, rolls no, age, address is a record of a student.

Up till now we used different techniques to get data in a program. For example LET, INPUT and READ/DATA statement. It is always useful to store input and output in a data file. The most important files are program files and data files.

Program files: Program files contain the program or instructions for the computer.

Data files: Data files contain data and information needed for programs to perform execution successfully. Data files are "linked" to or "included" with a program file during run-time or compilation time. These two access methods are called sequential and random.

By definition, sequential access means that the data contained in the targeted file will be accessed in the order in which it is physically stored on the disk. In other words, if you want to access the 25th record within a file, records 1 to 24 must be accessed first in order to reach the 25th (target) record.

Random access allows the programmer to directly access a specific record within a particular file. This obviously makes the search for a record contained in a file extremely faster than sequential access.

Sequential Files

It is important to note that a file position pointer is used by a file to "point to" the next record to be processed in the file. At first, the pointer starts

at the very beginning of the file therefore pointing to the first record, and then it is incremented to each next record while "cycling" through the file. This is how sequential access is able to work. The imaginary file position pointer is actually sequentially moved to each record within the file until the target record is reached.

Opening a File

When creating or accessing a sequential file, the first thing the programmer must do is open the file using the OPEN statement. The OPEN statement has the following form:

```
OPEN "fileName.ext" FOR mode AS #[buffer]
```

The OPEN statement will provide the name of the file, the way in which it will be used (mode), and the buffer number of the file. The same rules apply for naming data files as they do for naming "regular" BASIC program files, except a data file should normally be given an extension (.ext) of .DAT where as a program file is given an extension of .BAS. This could mean that the program using the data file could have the same name as the data file. The extensions would be the only way to distinguish the files from each other; this is a customary thing to do when naming program and data files that will only be associated with each other.

You can use a file as either OUTPUT, INPUT, or APPEND; these are the mode options. OUTPUT means that the program will eventually write data to the file. INPUT means that the data contained in the file will eventually be read into the calling program. APPEND means that the data will be added to the end of an existing file on disk.

A buffer is a reserved section of primary storage used for storing temporary data being written to or read from a file. The programmer specifies the buffer number of a file, and it is used by the program to recognize and identify the file that is attempting to be manipulated. The buffer number of the file must be known by the programmer when attempting to write data to or read data from the file.

If you try to open a file that doesn't exist, a new file will be created. If you try to open a pre-existing file as OUTPUT, the file will be overwritten and will lose all of its contents or data. For example, consider the following OPEN statement:

```
OPEN "STUDATA.DAT" FOR OUTPUT AS #1
```

The above OPEN statement would open the file *STUDATA.DAT* for output using 1# as a buffer number. If *STUDATA.DAT* pre-existed on disk, then it will be overwritten and created as an empty file. Once you have successfully opened a file, you can either write data to it, read data from it, or

append records to the file; this depends on what mode you specified for the file in the OPEN statement.

Writing to a File

Writing to a file that has been previously opened in OUTPUT mode is accomplished using the WRITE# statement. The WRITE# statement works much like the PRINT statement, except instead of sending output to the screen, the WRITE# statement will send data to the opened file. Another difference is that when using the WRITE# statement, you must first indicate the buffer number of the file being written to. For example, consider the following WRITE# statement:

```
WRITE #1, name, address, phone
```

The above statement would write the data contained in the name, address, and phone variables to the file using buffer number 1. The WRITE# statement actually only transfers the data into buffer 1's primary storage location. The transfer from primary memory to secondary memory actually occurs when "closing" the file, which is covered further in the article.

Reading From a File

Reading data from a file that has been previously opened as INPUT is accomplished using the INPUT# statement. The INPUT# statement is similar to the INPUT statement, except instead of reading user data input from the keyboard, the INPUT# statement reads data input from a file on disk. For example, consider the following:

```
INPUT #1, stuName, gpa, grade
```

The above statement would take values associated with the file with buffer number 1, and assign the values to each of the specified variables in the statement one value at a time. It is important to note that the INPUT# statement reads data from the record that is currently being pointed to by the file position pointer. After the INPUT# statement is executed, the file position pointer will be incremented to the next record in the file, and then the process is repeated if trying to read more data in the file.

It is also important to realize that there is an *end-of-file* (EOF) character at the end of every file. This allows the program accessing the file to recognize when there is no more data left in the file. The *end-of-file* character comes in handy when reading data from a file because it will let you know when all data has been read. The function EOF() that is used to check for the end of a file during the reading process. The EOF() function will return a value of false (0) if there are more records in the file, or it will return a value of true (1) if the end of the file has been encountered. Therefore, the EOF() function is actually evaluated like a Boolean expression. The argument sent

into the EOF() function will be the buffer number of the file being manipulated. Because the EOF() is a Boolean expression that returns a value of true or false, it is an efficient way of controlling the execution of a loop during reading data from a file. For example, consider:

```
10 WHILE NOT EOF( 1 )
20 INPUT #1, stuName, gpa, grade
30 PRINT "STUDENT: "; stuName; SPC(5); gpa; SPC(5); grade
40 WHILE
```

The above code would read all data and records contained in the file with buffer number 1 and print the data for each record until the *end-of-file* marker is reached. If you want to access and print data sequentially access and print data contained in a particular record in a file, you would have to read in (but not print) all the records preceding the "target" record. This can be accomplished by creating a "dummy" variable to keep track of how many records have been read from the file until the record you are trying to access is the next record to be read. This is time consuming, you reach to desired record in-efficient, and a better solution to this problem would be to use a random access method file. We dive into the details of random access later.

Closing a File

After use of a file, it must be closed. When writing data to a file on disk, the CLOSE statement is responsible for transferring the data currently in primary memory to secondary memory (file on disk). So, it is imperative that you close open files. This is accomplished using the CLOSE statement as follows:

```
CLOSE #buffer
```

Data is written to file (physical) when COLSE Statement encountered. It writes all temporary data that is currently in the corresponding buffer to the file. If we want to close our previous *STUDATA.DAT* file, we use it as:

```
CLOSE #1
```

You can also close many opened files using a single CLOSE statement as follows:

```
CLOSE #1, #2, #3, ..., #n
```

You can also close all currently opened files using a single CLOSE statement with no parameters by simply issuing:

```
CLOSE
```

Study the following complete program illustrating how to write/read data using a sequential file before moving on to learn about random files:

```
REM TOPIC CONCENTRATION: SEQUENTIAL FILES
```

```
REM This program will create a sequential file named INFO.DAT
```

```
REM The file will then be used to store input data by user
```

```

REM and then to retrieve the data shared in file.
OPEN "INFO.DAT" FOR OUTPUT AS #1
LS
REM Get input from user
INPUT "Do you wish to enter student information (Y/N)"; ANSWER$
IF ANSWER$ = "n" OR ANSWER$ = "N" THEN NODATA = 1 ELSE
NODATA = 2
WHILE ANSWER$ = "Y" OR ANSWER$ = "y"
INPUT "Enter student name: ", NAME$
INPUT "Enter student id: ", ID
INPUT "Enter student GPA: ", GPA
REM Write current input to temporary buffer storage 1
WRITE #1, NAME$, ID, GPA
INPUT "Do you wish to enter new student information (Y/N)"; ANSWER$
WEND
REM Transfer data from temporary buffer 1 to actual INFO.DAT
CLOSE #1
OPEN "INFO.DAT" FOR INPUT AS #2
IF NODATA <> 1 THEN
PRINT
PRINT "STUDENT LISTING"
PRINT "-----"
END IF
WHILE NOT EOF(2)
INPUT #2, NAME$, ID, GPA
PRINT
PRINT "STUDENT NAME: "; NAME$
PRINT " STUDENT ID: "; ID
PRINT " STUDENT GPA: "; GPA
WEND
CLOSE #2
END

```

Random Files

If the solution to the problem you are working with deals with directly accessing a particular record in a file, then the most efficient algorithm to use would involve using a random file access method. As previously mentioned, random files have the capability of accessing a record within a file without the need to access all preceding records like in sequential file. Specifying that a file be treated as random uses a form similar to when specifying different mode operations for a sequential file. As with all file types, the first thing the

programmer must do before attempting to use the file is open it. The form of opening a random file is as follows:

```
OPEN "fileName.ext" FOR RANDOM AS #[buffer] LEN =  
LEN(recordVariable)
```

Notice that we opened the file as being *RANDOM* and a buffer file number is used as usual (if this does not look normal. The major difference is the `LEN = LEN()` function initialization. The argument, *recordVariable*, that is sent into the `LEN()` function is a variable of a record type. The `LEN()` function will return the size of *recordVariable* in order to allocate space for each record to be stored in the file.

Writing or Storing Records

When dealing with random files, you can't send data from each field in a record one-by-one into the file. Instead, random files let you send the contents of the entire record to the file in one statement. This is accomplished by using the `PUT` statement, and the `PUT` statement also specifies the location in the file where the record is being sent. The `PUT` statement has the following form:

```
PUT #[buffer], recordNumber, recordVariable
```

In the above statement, the buffer file number must be first specified, the number of the record or the location of the record to be placed in the file is next, and finally the actual record variable is specified so the program knows which record to send to the file.

Reading Records

The complete opposite operation of the `PUT` statement would be to retrieve or get data contained in a data file. BASIC has a built-in function to handle this situation also. It is called `GET` and has a form identical to the `PUT` statement. The form of the `GET` statement is as follows:

```
GET #[buffer], record Number, record Variable
```

When using the `GET` statement, any record can be directly accessed by specifying the record's number or location file number. This is what makes random files much more powerful than sequential files. We can create random files but at this stage it is beyond the scope of the book.

Exercise

1. Fill in the blanks:
- (i) A subroutine itself is a _____ program.
 - (ii) A function is used to calculate and return a _____ value.
 - (iii) GOSUB and GOTO statements are _____.
 - (iv) A GOSUB is used to invoke a _____.
 - (v) User-defined function may be numeric or _____.
 - (vi) $\text{INT}(7.6) =$ _____.
 - (vii) A user defined function is started with a keyword _____.
 - (viii) GOSUB statement must be ended with _____ statement.
 - (ix) The RND function is referred to as a _____ number generator.
 - (x) _____ is a collection of fields to provide information about an entity in a file.
2. Choose the correct answer.
- (i) $\text{LEFT\$}('Pakistan',3) =$ _____
(a) 'Pak' (b) 'PAK' (c) ' Pa' (d) 'kis'
 - (ii) The outer of a function $\text{INT}(-5.7)$:
(a) -5 (b) -4 (c) -6 (d) 5
 - (iii) $\text{RIGHT\$}(x\$,n)$ will
(a) Leave 'n' spaces at the right of string x\$
(b) Leave 'n' spaces at the left of string x\$
(c) select 'n' characters from the right of side
(d) select 'n' spaces from the right of side.
 - (iv) The _____ function is used to convert ASCII codes to its character equivalent.
(a) $\text{CHAR\$}()$ (b) $\text{CHR}()$ (c) CHAR. (d) $\text{CHR\$}()$
 - (v) Instructions that are written once in the main program or independently and can be called more than one time in the main program is called.
(a) Control Statement (b) Loop
(c) Subprogram (d) None of them
 - (vi) $\text{TAN}(x) =$
(a) $\text{SIN}(x)/\text{COT}(x)$ (b) $\text{COT}(x)$
(c) $\text{COS}(x) / \text{SIN}(x)$ (d) $\text{SIN}(x)/\text{COS}(x)$
 - (vii) The output of $\text{SGN}(-4)$ is:
(a) '-' sign (b) '+' sign
(c) 0 (d) None of them
 - (viii) A file is activated with one of the following statement before its use:
(a) WRITE (b) READ
(c) PRINT (d) None of them

- (ix) A file can be organized in the following ways:
 (a) One way (b) Two ways
 (c) Three ways (d) Four ways
- (x) To read information from a file, it must be opened for:
 (a) Input (b) Output
 (c) Both a and b (d) None of them
3. Write T for True and F for False statements.
- (i) There are three techniques to organize files.
 - (ii) Statement END must be placed at the end of main program.
 - (iii) FIX function gives an integer value by rounding off the fractional part.
 - (iv) Subroutines are not easy to design and debug and modify..
 - (v) A function is associated with two statements GOSUB and RETURN
 - (vi) A file is an organized collection of records.
 - (vii) An OPEN statement must contain a reference to the file.
 - (viii) Built-in function can also be defined as library function.
 - (ix) A file opened for output is used to read data from it.
 - (x) The keywords used for using subroutines are GOSUB and RETURN
4. What is the difference between user-defined function and built-in functions?
5. Differentiate between Sub-routine and Function.
6. Describe the use of GOSUB...RETURN statement.
7. What is the difference between Sequential and Random files?
8. Describe the way of opening, closing, reading and writing to a sequential file.
9. Differentiate between data file and program file.
10. Write down the purpose of the following functions:
 (i) ABS() (ii) INT()
 (iii) SQR() (iv) SIN()
 (v) TAB()
11. Write a program to get full name of any person and return the number of characters in his first name.
12. Write a program that print ASCII characters from 1 to 255.
13. Write a program that is used for the conversion of temperature from Celsius scale to Fahrenheit scale with the help of DEF FN function.
14. Write a program to calculate and print following formula by using user-defined function. $Combination = \frac{n!}{k!(n-k)!}$
15. Write a program to implement a telephone directory using sequential access files. Your program should be capable of writing the name, telephone number and address of your friends to a sequential file.

Answers

1. (i) Small (ii) Single (iii) Different
(iv) Subroutine (v) String (vi) 7
(vii) DEF FN (viii) RETURN (ix) Random
(x) Record
2. (i) c (ii) c (iii) c
(iv) d (v) c (vi) d
(vii) d (viii) d (ix) b
(x) c
3. (i) F (ii) T (iii) F
(iv) F (v) F (vi) T
(vii) T (viii) T (ix) F
(x) T

GRAPHICS IN BASIC

6.1 INTRODUCTION

Graphics is an art to design and produce pictorial representation of information. This facility is provided in almost all the versions of BASIC language. It is displaying information on screen. Graphic is that area of computer programming which is highly in use these days. It depends on the hardware such as input, output and graphic card (Color Graphic Adaptor, Video Graphic Array).

Let us start from the beginning. Your screen is made up of hundreds of pixels. The number of pixels horizontally and vertically determines the resolution of your monitor. In GW-Basic, we can be in any number of graphics modes, which define the current graphics resolution (pixels), text resolution (characters), number of colors, and number of video pages. There are 13 screen graphics modes, and each has its different purpose. There are several ways of drawing to the screen. Each uses things called coordinates to define what area of the screen to use. A coordinate is a specific pixel (Picture element: A pixel is one dot on the screen). Your computer screen is made with 1 million little square of color (pixel) you can determine any coordinate. A coordinate can be determined by counting the numbers of pixels down and the number of pixels to the right the pixel.

BASIC provides three modes of displaying data.

- Text Mode
- Medium-Resolution Graphic mode
- High-resolution Graphic mode

Text Mode is used for only textual data. In text based graphic, text and lines can be drawn on the screen. The whole screen is divided into 80 column and 25 rows. It has 16 colors out of any 2 colors palettes (table 6.1). Columns are counted from 0 to 39 or 79 and rows from 0 to 24.

Medium-Resolution Graphic Mode is used in drawing graphic. The display screen is divided into a matrix consisting of 320 columns and 200 rows of pixels. Thus the position of each and every pixel will be determined by its coordinates on x-axis and y-axis of the screen. This graphical mode works with 4 colors. The different four colors are 0,1,2,3, one of 16 color can chosen for background and one for foreground. Each of the foreground and background palettes is given in table 6.1

High Resolution Graphic Mode is used in drawing graphics with matrix of 640 x 200 pixels. We are able to display the text characters in 25 lines of 80 characters in each line.

6.1.1 SCREEN Statement

The SCREEN statement is commonly used to select a screen mode appropriate for a particular display-hardware configuration. For example the supported hardware configuration like IBM Monochrome Display and Printer Adapter (MDPA) with mode 0 is used to connect only to a monochrome display. Programs written for this configuration must be in text mode only.

Syntax:

SCREEN [*mode*] [, [*colorswitch*]]

Where screen *mode* is a numeric value from 0, 1, 2, 7, 8, 9, 10. It may not be run on each type of computer or monitor, because it depends on the display card and monitor's type. Different modes of screen are below.

Screen modes 0, 1, 2, 7, 8, 9, and 10

In BASIC language, screen mode 0 is by default mode. It is only text base, and it can be viewed on the screen. Screen mode 1 activates medium resolution graphic mode. Graphic up to resolutions 320 x 200 pixels can be obtained in it. Screen mode 2 activates high resolution graphic mode. Graphic of resolution 640 x 200 pixels can be obtained in it. It also needs good quality monitors like CGA(Color Graphics Adapter), (EGA) Enhanced Graphics Adapter ,VGA etc.

Screen modes 7,8,9,10 are also used to draw medium resolution and high resolution graphics. All of them support graphics and good quality result can be obtained in them. Some major screen modes describe below.

SCREEN MODE	DESCRIPTION
-------------	-------------

- | | |
|----|---|
| 0 | Text mode only, you can't use any graphics |
| 1 | 320 x 200, only has 4 colors |
| 2 | 640 x 200, only 2 colors (Black and White) |
| 7 | 320 x 200, 16 colors and supports pages (get to later)
This screen mode is very useful |
| 8 | 640 x 200, 16 colors no pages |
| 9 | 640 x 350, 16 colors supports pages (very useful!) |
| 10 | 640 x 350, 2 colors (black and White) |
| 11 | 640 x 480, 2 colors |
| 12 | 640 x 480, 16 colors (very useful) |
| 13 | 320 x 200, 256 colors (extremely useful) |

For various screen modes and display hardware configurations, different attribute and color settings exist. (See the PALETTE statement for a discussion of attribute and color number.) The Default Attributes and Colors for Most Screen Modes are:

Attributes for Mode			Color Display	
1,9	2	0,7,8,9	Number	Color
0	0	0	0	Black
		1	1	Blue
		2	2	Green
		3	3	Cyan
		4	4	Red
		5	5	Magenta
		6	6	Brown
		7	7	White
		8	8	Gray
		9	9	Light Blue
		10	10	Light Green
1		11	11	Light Cyan
		12	12	Light Red
2		13	13	Light Magenta
		14	14	Yellow
3	1	15	15	High-intensity White

Table 6.1 Color attributes for screen except screen 10

6.1.2 COLOR Statement

After screen modes the next important thing used in graphics is the COLOR statement. The purpose of COLOR statement is to select display colors.

Syntax:

COLOR [*foreground*][,*background*][,*border*]

COLOR [*background*][,*palette*]

COLOR [*foreground*][,*background*]

In general, COLOR allows you to select the foreground and background colors for the display. In SCREEN 0 a border color can also be selected. In SCREEN 1 no foreground color can be selected, but one of two four-color palettes can be selected for use with graphics statements. The different syntaxes and effects that apply to the various screen modes are described below:

Mode	Effect
SCREEN 0	<p>Modifies the current default text foreground and background colors, and the screen border. The <i>foreground</i> color must be an integer expression in the range 0-31. It is used to determine the "foreground" color in text mode, which is the default color of text. Sixteen colors can be selected with the integers 0-15. A blinking version of each color can be selected by adding 16 to the color number; for example, a blinking color 7 is equal to 7 + 16, or 23. Thus, the legal integer range for <i>foreground</i> is 0-31.</p> <p>The <i>background</i> color must be an integer expression in the range 0-7, and is the color of the background for each text character. Blinking colors are not permitted.</p> <p>The <i>border</i> color is an integer expression in the range 0-15, and is the color used when drawing the screen border. Blinking colors are not permitted.</p> <p>If no arguments are provided to COLOR, then the default color for background and border is black (COLOR 0), and for foreground, is as described in the SCREEN statement reference pages.</p>
SCREEN 1	<p>In mode 1, the COLOR statement has a unique syntax that includes a <i>palette</i> argument, which is an odd or even integer expression. This argument determines the set of display colors to use when displaying particular color numbers.</p> <p>For hardware configurations the default color settings for the <i>palette</i> parameter are equivalent to the following:</p> <p>COLOR ,0 'Same as the next three PALETTE statements '1 = green, 2 = red, 3 = yellow</p> <p>COLOR ,1 'Same as the next three PALETTE statements '1 = cyan, 2 = magenta, 3 = white</p>
SCREEN 2	<p>No effect. An "Illegal function call" error message occurs if COLOR is used in this mode.</p>
SCREEN 7- SCREEN 10	<p>In these modes, no <i>border</i> color can be specified. The graphics background is given by the <i>background</i> color</p>

Note: Arguments outside valid numeric ranges result in "Illegal function call" errors.

Examples:

The following series of examples show COLOR statements and their effects in the various screen modes:

SCREEN 0

COLOR 1, 2, 3 'foreground=1, background=2, border=3

SCREEN 1

COLOR 1, 0 'foreground=1, even palette number

COLOR 2, 1 'foreground=2, odd palette number

SCREEN 7

COLOR 3, 5 'foreground=3, background=5

SCREEN 8

COLOR 6, 7 'foreground=6, background=7

SCREEN 9

COLOR 1, 2 'foreground=1, background=2

6.1.3 PALETTE

It is used to select from one of two color sets that are already available such that these colors sets will be used by color parameter of a LINE, CIRCLE, PSET, DRAW or other graphics utilities.

Syntax:

PALETTE [*attribute*, *color*]

The PALETTE statement works only for systems equipped with the Enhanced Graphics Adapter (EGA). A GW-BASIC palette contains a set of colors, with each color specified by an *attribute*. Each *attribute* is paired with an actual **display color** (see in Table 6.1). This *color* determines the actual visual color on the screen, and is dependent on the setting of your screen mode and your actual physical hardware display.

The value of palette may be either 0, 1. If palette 0 or 1, then the colors are:

Palette	Numbers	Color
0	0	Back ground Color
0	1	Green
0	2	Red
0	3	Brown
1	0	Back ground Color
1	1	Cyan
1	2	Magenta
1	3	White

6.2 PSET Statements

Purpose: To display a point at a specified place on the screen during use of the graphics mode.

Coordinates values can be beyond the edge of the screen. However, values outside the integer range (-32768 to 32767) cause an "Overflow" error. (0,0) is always the upper-left corner and (0,199) is the lower-left corner in both high resolution and medium resolution. If the value for color is greater than 3, an "Illegal function call" error is returned.

Syntax:

PSET(x,y)[,color]

PSET (x offset, y offset) is a point relative to the most recent point referenced. For example:

PSET(10,10)

Example 1:

The following program clears out the line by setting each pixel to 0.

```
40 FOR I=100 TO 0 STEP -1
```

```
50 PSET(I,I),0
```

```
60 NEXT I
```

6.3 LINE Statement

The purpose of LINE statement is to draw lines and boxes on the screen. We can use the LINE statement to generate a line between any two statements.

Syntax:

LINE [(x1,y1)]-(x2,y2) [,attribute][,B[F]][,style]

In the syntax the values (x1,y1) and (x2,y2) are used to specify the coordinate positions of starting and ending point of the line. These two points positions are separated by a minus or dash sign. The attribute specifies the color or intensity of the display pixel.

B(box) to draw a box with the points (x1,y1) and (x2, y2) at opposite corners. **BF(filled box)** to draw a box as B and fills in the interior with points. The style is a 16-bit integer mask used when putting down pixels on the screen. This is called line-styling. Style can one of 0,1, 2, 3, 4, and 5. It is used for normal lines and boxes, but invalid for filled boxes.

The simplest form of LINE is the following:

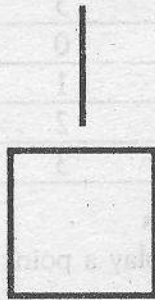
LINE -(x2,y2)

This draws a line from the last point referenced to the point (x2,y2) in the foreground color.

Examples:

LINE (160,0)-(160,199)

LINE (0,0)-(100,175),,B



a square box in the upper left corner of the screen.

6.4 CIRCLE Statement

The purpose of CIRCLE statement is to draw a circle, ellipse on the screen during use of the Graphics mode

Syntax:

CIRCLE(*x*, *y*), *radius* [, *color*] [, *start*], *end*] [, *aspect*]]

The (*x*, *y*) are the x-axis and y-axis coordinates of the center of the circle or ellipse, and *radius* is the radius (measured along the major axis) of the circle or ellipse. The quantities *x* and *y* can be expressions.

The *color* specifies the color of the circle or ellipse. Its value depends on the current screen mode. In the high-resolution mode, 0 indicates black and 1 indicates white.

The *start* and *end* angle parameters are radian arguments between -2π and 2π which specify where the drawing of the ellipse is to begin and end

The *aspect* describes the ratio of the x-axis to the y-axis (*x*:*y*). The default aspect ratio depends on the screen mode.

Example 1:

```
10 SCREEN1: CIRCLE(100,100), 50
```



Ellipse is a mathematical term used for shapes of oval type by using one more parameter to the CIRCLE statement, ellipse can be drawn.

Example 2:

```
10 REM draw an ellipses
20 CLS
30 SCREEN 2
40 CIRCLE (40,80),30,1,,1
50 END
```



In this case, height of ellipse will be greater than its width.

Example 3:

```
10 REM draw an ellipses
20 CLS
30 SCREEN 2
40 CIRCLE (40,80),30,1,,2
50 END
```



The ellipse drawn from the above program will be less in width than that of example 2.

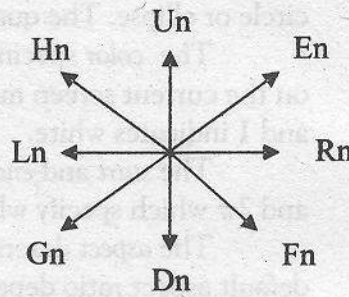
6.5 DRAW Statement

The DRAW statement is used to draw lines and other shapes on the screen. It is valid only in graphics mode. DRAW statement is used to a object of the shape other than circular. The object would be drawn as specified by a string containing drawing commands.

Syntax: DRAW string

A string consists of single character command followed by a prefix that controls the size, direction etc. of the line and enclosed in the quotation marks. Each of the following movement commands begins movement from the current graphics position. Movement commands move for a distance of scale factor *n, where the default for n is 1; thus, they move one point if n is omitted and the default scale factor is used.

Command	Moves
Un	up
Dn	down
Ln	left
Rn	right
En	diagonally up and right
Fn	diagonally down and right
Gn	diagonally down and left
Hn	diagonally up and left



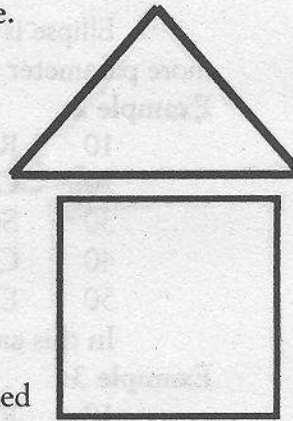
The following prefix commands may precede any of the above movement commands:

- B Move, but plot no points.
- N Move, but return to original position when done.

REM PROGRAM TO DRAW A TRIANGLE

```
20 SCREEN 2
30 PSET(250,50)
40 DRAW "G50 R100 H50"
50 REM PROGRAM TO DREW A SQUARE
60 SCREEN 2
70 PSET(250,50)
80 DRAW "R50 D50 L50 U50"
```

When you run the program, a square will be obtained



Exercise

1. Fill in the Blanks:

- (i) SCREEN 2 is _____ resolution graphic mode.
- (ii) In the text mode characters are displayed on the screen in _____ columns multiplied by _____ rows.
- (iii) In medium resolution mode, the screen is divided into a matrix of _____ pixel.
- (iv) A string consists of double character command followed by prefix that control the size, direction in _____ statement.

- (ix) In text mode characters are displayed on the screen in 40 column and 25 rows.
 (x) CIRCLE statement can also be used to draw an ellipse.
4. Define Graphic. Give the names of graphic modes.
 5. What are the coordinates of text mode, medium resolution mode and higher resolution mode?
 6. Describe the SCREEN Statement.
 7. Write the syntax of CIRCLE statement. Also give example for explanation.
 8. Compare and differentiate LINE and DRAW statements.
 9. Find out the errors in the following if any?
 - (a) LINE (140,100)-(300-100),2,BF,4
 - (b) 10 SCREEN 2
 - (c) 10 SCREEN 1
 - 20 COLOR 1, 2
 - 20 A=20
 - 30 DRAW "U10 R10 D10 L10" 30 DRAW "U=A R=A, D=A L=A"
 10. What will be the output of the following.
 - (a) 10 SCREEN 2
 - 20 PSET(250, 50)
 - 30 DRAW "G50 R100 H50"
 - 40 END
 - (b) 10 SCREEN 2
 - 20 FOR I = 30 TO 180
 - 30 CIRCLE(1,100), 50
 - 40 NEXT I
 11. Write a program to draw a star.
 12. Define the COLOR statement. How many color backgrounds are available with color statement?
 13. Define the Palette.
 14. Briefly discuss the PIXEL.
 15. Write a program to produce five concentric circles of different radii.
 16. Write a program to draw a parallelogram by using DRAW statement.

Answers

1. (i) High (ii) 40, 25 (iii) 300 x 200 (iv) DRAW
 (v) DRAW (vi) Brown (vii) 640, 200 (viii) Pixels
 (ix) 0,1 (x) High
2. (i) c (ii) d (iii) a (iv) d (v) b
 (vi) c (vii) c (viii) a (ix) c (x) a
3. (i) T (ii) F (iii) T (iv) T (v) F
 (vi) T (vii) T (viii) F (ix) T (x) T

MICROSOFT WORD

7.1 INTRODUCTION

Initially the past people use to write document with the help of pen or pencil and paper. Then after that they have started to use typewriter but people still face too many problems such as erasing or editing etc. the documents. Computer software solved these problems. There is variety of software that performs specific task. For example, word processor is an application that is used to write documents. *Microsoft word* is a powerful *word processing* program.

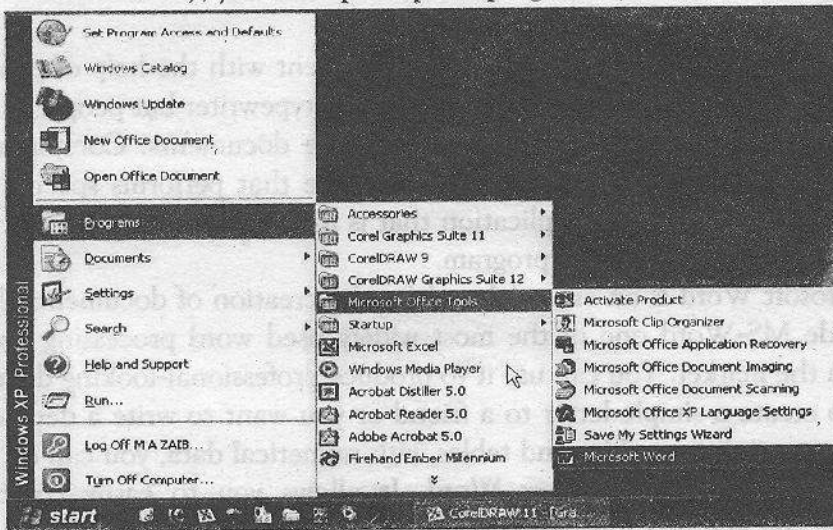
Microsoft Word is an essential tool for the creation of documents. Its ease of use has made MS-Word one of the most widely used word processing applications currently on the market. You can use it to produce professional-looking documents. If you want to create a simple letter to a friend or you want to write a detailed, multi-page report containing graphics and tables with numerical data, you can do it quickly and easily done in the Microsoft Word. It allows you to easily combine text, spreadsheets, and graphics into a single application. You can also use Word to create your own Web pages. Word's variety of pull-down menus, toolbars, and buttons make learning and using it remarkably easy.

Therefore, it is important to become familiar with the various features of latest MS-WORD 2003 software. The characteristics of MS-Word are:

- To create a fast and easy documents quickly.
 - To provide built-in templates for creating, letters, journals, faxes, calendars and many more.
 - To give built-in spell checker and grammar.
 - To provide auto-correction by just mouse clicking.
 - To insert an AutoText entry so as to insert a short cut key for a long text or paragraph.
 - To take multiple hard copies of a document.
 - To give the facility to open an existing document
 - To save document for future use
 - To make easily be edited and formatted.
 - To provide facility to undo and redo what we have typed or taken an action.
 - To give a quick finding and replacing the text.
 - To provide facility creating tables, pictures and even one can applying simple formulas etc.
 - One of the best features is to save the document in HTML format.
- Some of the features may not available in lower version of MS-Word.

7.1 LOADING MICROSOFT WORD

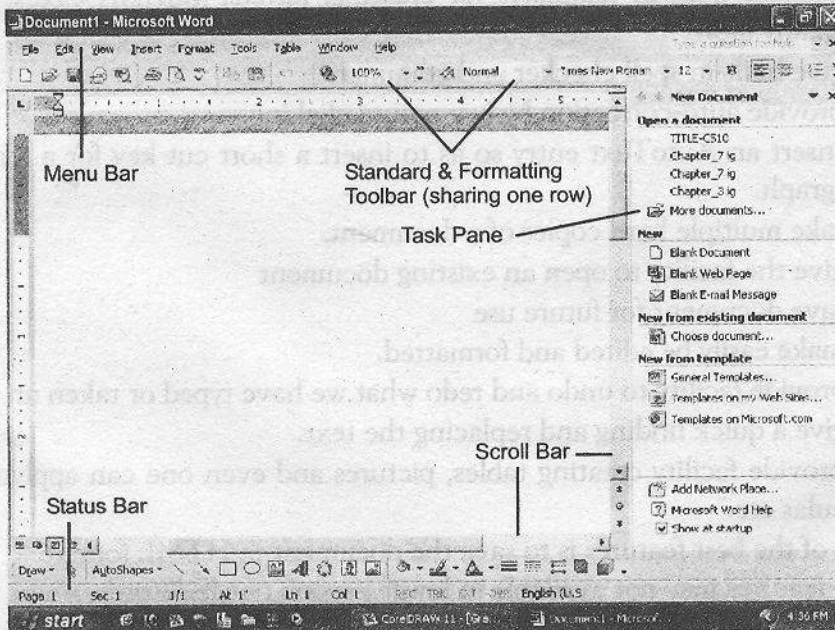
Loading the Word processor is the process to run Word on your computer: “Start” >> “Programs” >> “Microsoft Office” >> “Microsoft Office Word .” If there is an icon of Microsoft Word available on your desktop (shaped like a square with a “W” in the middle), you can open up the program by double-clicking it, as well.



Once the MS-Word is loaded we will be able to create, edit and save a document. Here we will learn how to create new document, writing text into the document, saving and exiting form it.

Screen Layout

The application window provides the space for the Word document. The components of the application window are:



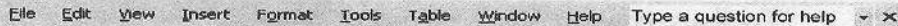
- Title bar
- Menu bar
- Tool bar
- Document Window
- Status bar
- View Button

The Title Bar

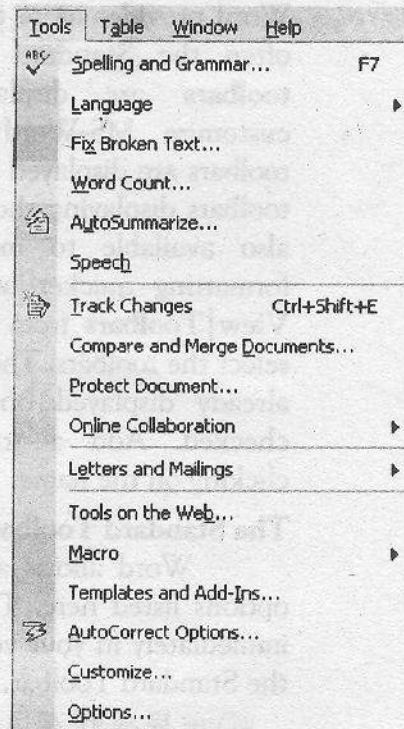


We will start with the Title bar, which is located at the very top of the screen. On the Title bar, Microsoft Word displays the name of the document in which you are currently working. At the top of your screen, you should see "Microsoft Word - Document1" or a specified name.

The Menu Bar and Drop Down Menus



The Menu bar is generally found directly below the Title bar. The Menu bar displays the menu. The Menu bar begins with the word File and continues with Edit, View, Insert, Format, Tools, Table, Window, and Help. You use the menus to give instructions to the software. Point with your mouse to a menu option and click the left mouse button to open a drop-down menu. You can now use the left and right arrow keys on your keyboard to move left and right across the Menu bar options. You can use the up and down arrow keys to move up and down the drop-down menu.



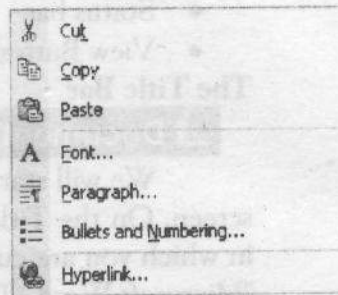
When you begin to explore Word, you will notice a significant change in the menu structure if you are familiar with previous versions of Word. The menus in Word display only the commands you have recently used. To view all options in each menu, you must click the double arrows at the bottom of the menu. The images below show the Tool menu collapsed (left) and expanded (right) after the double arrows at the bottom of the menu were clicked:

Drop Down Menus: Clicking on any of the menu bar items will cause MS-Word to display a drop down menu, which in turn contains further items that may be selected. In some cases the items within a drop down menu may appear "grayed out", this means that option is not currently available. To access the

drop down menus using the keyboard, depress the **Alt** key and enter the letter that is underlined within the drop down menu command, i.e., to display the **Tool** drop down menu list, press **Alt + T**

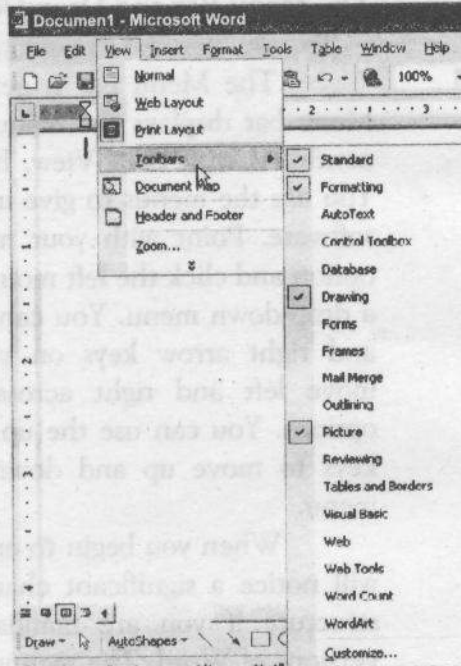
Shortcut Menu

These features allow you to access various Word commands faster than using the options on the menu bar. View shortcut menus by right-clicking with the mouse at work area of document. The options on this menu will vary depending on the element that was right-clicked. For example, the shortcut menu below is produced by right-clicking on a bulleted list.



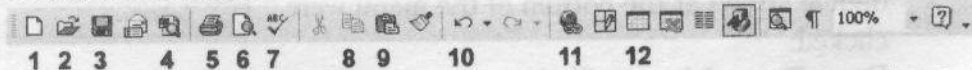
Toolbar

Toolbars contain small pictures called tool icons which, when clicked on, provide a shortcut method or performing MS-Word action. MS - Word provides more than 18 tool bars, often the **Standard** and **Formatting** toolbars are displayed. You can customize MS-Word so that other toolbars are displayed by default. Many toolbars displaying shortcut buttons are also available to make editing and formatting quicker and easier. Select **View|Toolbars** from the menu bar to select the toolbars. The toolbars that are already displayed on the screen are checked. Add a toolbar simply by clicking on the name.



The Standard Toolbar

Word allows all toolbars to be customized, so you may not find all options listed here. There are several buttons that may or may not appear immediately in your version of Word. Use the following graphic as a guide to the Standard Toolbar.



1. **New Blank Document:** To begin a new document, click on the New Blank Document icon, shaped like a blank sheet of paper.
2. **Open:** Clicking on this icon opens up a previously saved document on your computer.


3. **Save:** Clicking on the Save icon saves the document you are currently working on. If you are saving a document for the first time, you can click on this button. However, if you want to save a new file from a pre-existing document, then you must go to the menu bar and select **"File" >> "Save As"** and give the file a new name. When working on any document, you should be sure to save frequently, so that you don't lose any work.
4. **Permission:** Microsoft has enabled Information Rights Management (IRM) within the new version of Word, which can help protect sensitive documents from being copied or forwarded. Click this for more information and options. This option is not available in word 2000.
5. **Print:** Clicking on the Print icon automatically prints the document currently active in Word. If you wish to explore more print options, then go to the menu bar and select **"File" >> "Print."**
6. **Print Preview:** To get an idea of the appearance of your document in print before you actually print it out, you can click on this icon to view your document from a zoom-out distance.
7. **Spelling and Grammar:** Clicking begins a review of your document in search of spelling and grammatical errors that may need to be corrected.
8. **Copy:** Copy the current selection to the clipboard, which can then be pasted elsewhere in the document, or into a completely separate program/document. To copy text, choose **Edit|Copy**, click the **Copy** button on the standard toolbar, or press **CTRL+C** to copy the text to the clipboard.
9. **Paste:** Clicking on the Paste button inserts the text that has been most recently added to the Clipboard (the text would have been added there by Cutting or Copying). With Paste, you can either insert the copied text into a document or replace selected text.
10. **Undo Typing:** The Undo Typing button goes back and removes the last addition or change made to your document.
11. **Insert Hyperlink:** You may find that you want to make links to a particular web site, web page, or some other kind of online file in your Word document. Using the Insert Hyperlink button, you can turn selected text into hyperlinks. When the icon is clicked, a window will appear that will allow you to insert the URL (web address) of the web page you want to link to.
12. **Insert Table:** When this icon is clicked, a small window will appear in the form of a grid of squares. Use this window as a guide to indicate how many rows and columns you would like your table to contain.

Once selected, a table will automatically appear in Word. Clicking the Tables and Borders button will allow you to modify the table. To modify an aspect of the table, select, or place the cursor in, the area and apply changes such as borders and colors.

The Formatting Toolbar

Word allows all toolbars to be customized, so you may not find all options listed here. There are several buttons that may or may not appear immediately in your version of Word. Use the following graphic as a guide to the Formatting Toolbar.



1. **Style:** Styles in Word are used to quickly format portions of text. For example, you could use the "Normal" or "Default Paragraph Font" for the body text in a document. There are also three preset styles made for headings.
2. **Font:** Font is a simple but important factor in Word documents. The choice of font (the style of the text itself) can influence the way others view documents, either on the screen or in print. For example, Arial font looks better on screen, while Times New Roman is clearer in print. To apply a font to text, select desired text with your cursor, and choose a font from the font drop down menu. Scroll down to the font you want and select it by clicking on the name once with the mouse. A serif font (one with "feet" circled in the illustration below) is recommended for paragraphs of text that will be printed on paper as they are most readable. The following graphic demonstrates the difference between *serif* (Times New Roman on the left) and *sans-serif* ("no feet", Arial on the right) fonts. 
3. **Font Size:** You may encounter times in which you need to display some text larger or smaller than other text. Selecting desired text with the cursor and choosing a font size from the drop down menu changes the size of text. Select a size by clicking on it once. A font size of 10 or 12 is best for paragraphs of text.
4. **Bold:** Places the text in bold.
5. **Italic:** Places the text in *italics*.
6. **Underline:** Underlines the text.
7. **Align Left:** Aligns the selection to the left of the screen/paper.
8. **Center:** Aligns the selection to the center of the screen/paper.
9. **Align Right:** Aligns the selection to the right of the screen/paper.
10. **Justify:** Aligns the selection to both the left and right of the screen/paper.

11. **Line Spacing:** Adjust the line spacing (single-spaced, double-spaced, etc.)
12. **Numbering:** Create a numbered list.
13. **Bullets:** Create an unordered, bulleted list.
14. **Decrease Indent:** Decreases the indentation of the current selection (to the left).
15. **Increase Indent:** Increases the indentation of the current selection (to the right).
16. **Outside Border:** Places a border around the current selection; click the drop-down for a wide selection of bordering options.
17. **Highlight:** Highlight the current selection; default color is yellow.
18. **Font Color:** Change the font color; the default/automatic color is black.

Document Window

The area in which text is written, edited, formatted, etc. is called document window. It comprises the largest area of the Word application window. The document window also contains the following:

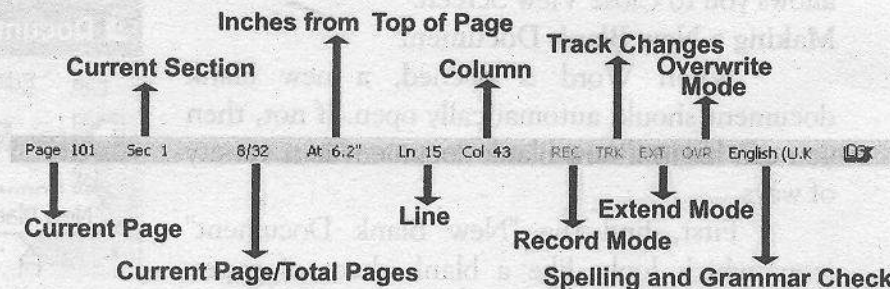
Insertion: The blinking vertical bar is called insertion point. It indicates the place where typed text will be written in the document.

End Mark: The thick horizontal line specifies end of the document. It is called the end mark. The end mark is displayed in **Normal view** only.

I-Beam: The mouse pointer appears as I-shaped object when it is placed in the document window. It is also called I-beam.

Status Bar

The Status bar appears at the very bottom of the screen and provides such information as the current page, current section, total number of pages, inches from the top of the page, current line number, and current column number. The Status bar also provides options that enable you to track changes or turn on the Record mode, the Extension mode, the Overtyping mode, and the Spelling and Grammar check.



Scroll Bar

The document window has scroll bars. At the right edge of the document window is a vertical scroll bar, and at the bottom of the document

window is the horizontal scroll bar. The scroll bars are used to move up and down, right and left in the document window.

The Ruler



The ruler is generally found below the main toolbars. The ruler is used to change the format of your document quickly. To display the ruler:

- Click View on the Menu bar.
- The option Ruler should have a check mark next to it. If it has a check mark next to it, press Esc to close the menu. If it does *not* have a check mark next to it, continue to the next step.
- Click Ruler. The ruler now appears below the toolbars.

View Buttons

Changes the layout view of the document to normal view, web layout view, print layout view, or outline view.

Change in View

In an effort to provide various ways in which to view your work in progress and remain organized, Word XP offers five different views for your document. The five views are Normal view, Print Layout view, Web Layout view, Outline view, and Full Screen View.

Normal view is best used for typing, editing, formatting and proofreading. It provides a maximum amount of space without rulers or page numbers cluttering your view.

Web Layout view shows you what your text will look like on a web page.

Print Layout view shows you what your document will look like when it is printed. Under Print Layout view you can see all elements of the page. Print Preview shows you this as well.

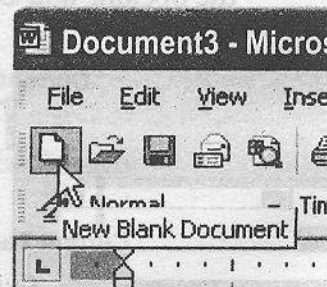
Outline view is used to create and edit outlines. Outline view only shows the headings in a document. This view is particularly handy when making notes.

Full Screen view displays **ONLY** the document that you are working on. All the other pieces of the Word window are removed except for one button that allows you to Close View Screen.

Making a New Blank Document


When Word is opened, a new blank document should automatically open. If not, then you can begin a new blank document in a variety of ways.

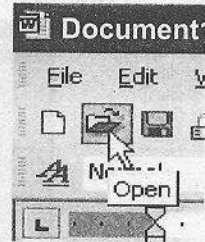
First, find the "New Blank Document" icon, which looks like a blank sheet of paper, located underneath the menu bar in Word in what is called the "standard toolbar." Click on the icon to bring up a new blank document.



Also, you can go to the menu bar and select **File >> New...** (shortcut: **Ctrl+N**). To begin typing, just click the cursor anywhere within the new blank document.

Opening a Document


To open to view, edit, or print a document, you must first open up that file in Word. You can open a file by clicking on the "Open" folder icon  (with a picture of a folder) located in the standard toolbar. Or, you can use the menu bar and navigate to **File >> Open...** (shortcut: **Ctrl+O**).

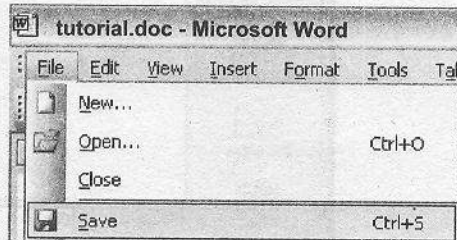


Saving a Document

When you are working with any sort of media in any software, you should be sure to save your work often. In Word, there are numerous options for saving documents in a variety of file types.

To save a new, unsaved document, you

can Click on the Save icon , shaped like a disk located on the standard toolbar. Or, you can go to the menu bar and select **File >> Save...** (shortcut: **Ctrl+S**).



Note: The save in drop-down list box lists folder options where you can save the document. The default folder that appears is "Documents." If you don't want to save it to this folder, or if you want to save your document to another disk, you can select another one. Click on the down arrow to browse.

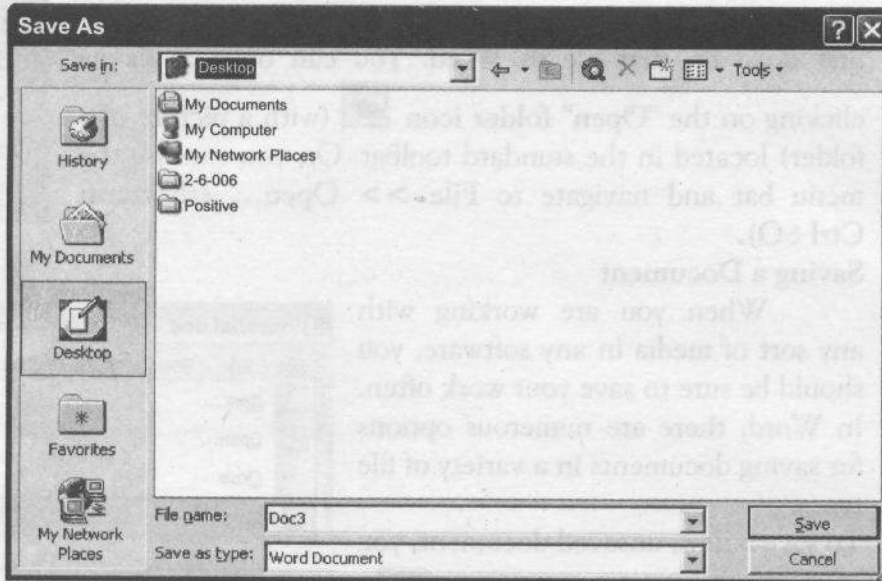
When you first create a document, it has no name. If you want to use that document later, it must have a name so Word can find it. Word asks for a name that first time you save the document, and after that, the name you give it will appear in the title bar at the top of the screen.

A dialogue box should appear, offering you a number of options. To save the document in the desired location on your computer, locate and select the folder on your computer. Give your document a name in the file name text box. While you can give your document long names, make sure you save it with a name you can remember.

Please note that it's good practice not to use spaces or special characters in file names. For example, a long file name may look like this:

To save a completely new document using previously existing (and opened) text, you use the **Save As** option. Open the document that you wish

to save as an entirely new file, go to the menu bar, and click on **File >> Save as**. In the file name text box, give your document a new name. Using this option allows you to save multiple versions (with different file names) of a document based on one original file.



Renaming Documents

To rename a Word document while using the program, select **File|Open** and find the file you want to rename. Right-click on the document name with the mouse and select **Rename** from the shortcut menu. Type the new name for the file and press the **ENTER** key.

Close a Document

Close the current document by selecting **File|Close** or click the Close icon if it's visible on the Standard Toolbar.

7.1.1 Editing a Document

The process of adding, inserting, and changing and deleting text in a word document is called text editing. Similarly, inserting and changing contents and appearance of a graphics is called graphics editing.

Following is a list of most commonly used editing procedures in Word:

- Typing and Inserting Text
- The Undo and Redo Commands
- Selecting Text
- Deleting Text
- Moving Text
- Copying Text
- Paste Text

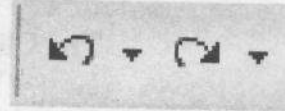
Typing and Inserting Text

To enter text, just start typing! The text will appear where the blinking cursor is located. Move the cursor by using the arrow buttons on the keyboard or positioning the mouse and clicking the left button. The keyboard shortcuts listed below are also helpful when moving through the text of a document:

Move Action	Keystroke
Beginning of the line	HOME
End of the line	END
Top of the document	CTRL+HOME
End of the document	CTRL+END

The Undo and Redo Commands

If you accidentally deleted text or placed the wrong format on a block of text, you can use the Undo command to reverse changes that you've made in your document. When you click on the Undo button, you



Undo Redo

reverse your last action. To undo multiple actions, select them from the Undo drop-down list. Multiple actions are undone in the sequence performed. While most actions can be undone, there are certain actions which cannot, such as saving or printing a document. If you change your mind after using the Undo command, you can reverse it by clicking on the Redo button.

Unless you are perfect typist, you probably have a few mistakes in your document, or perhaps you have changed your mind about some of the text in the document. In a word processing program, correction and changes are easy to make.

Selecting Text

To change any attributes of text it must be highlighted first. Select the text by dragging the mouse over the desired text while keeping the left mouse button depressed, or hold down the **SHIFT** key on the keyboard while using the arrow buttons to highlight the text. The following table contains shortcuts for selecting a portion of the text:

Selection	Technique
Whole word	double-click within the word
Whole paragraph	triple-click within the paragraph
Several words or lines	drag the mouse over the words, or hold down SHIFT while using the arrow keys
Entire document	choose Edit Select All from the menu bar, or press CTRL+A

Deselect the text by clicking anywhere outside of the selection on the page or press an arrow key on the keyboard.

Deleting Text

Use the **BACKSPACE** and **DELETE** keys on the keyboard to delete text. Backspace will delete text to the left of the cursor and Delete will erase text to the right. To delete a large selection of text, highlight it using any of the methods outlined above and press the **DELETE** key.

The Clipboard

You can view the elements on the clipboard by selecting **View|Toolbars|Clipboard** from the menu bar.

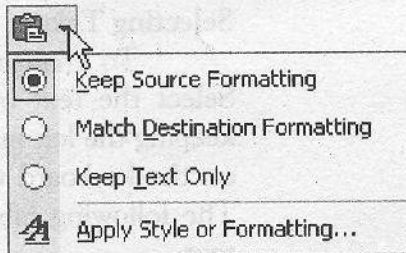
Place the mouse arrow over each element in the clipboard to view the contents of each item and click on an element to add its contents to the document. Click **Paste All** to add all of the items to the document at once. Click the **Clear Clipboard** button (the icon with an "X" over the clipboard image) to clear the contents of the clipboard.

When you copy and paste text, a clipboard icon comes on the screen which gives you an option on how to paste the text.

This is an example of the clipboard feature.

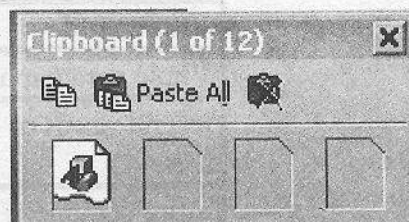
Note:

Word XP provides a Collect and Paste, which provides the means to manage multiple pieces of information. The Microsoft Office XP Clipboard allows you to copy up to twelve pieces of text or pictures from one or more documents, e-mail messages, Web pages, presentations or other files. You can then paste these pieces into your document, either individually, or all at once using the Paste All feature.



Moving (Cutting) Text

Highlight the text that will be moved and select **Edit|Cut** from the menu bar, click the **Cut** button on the standard tool bar, or press **CTRL+X** at once. This will move the text to a clipboard.



To move a small amount of text a short distance, the drag-and-drop method may be quicker. Highlight the text you want to move, click the selection with the mouse, drag the selection to the new location, and release the mouse button.

Copying Text

To copy text, choose **Edit|Copy**, click the **Copy** button on the standard toolbar, or press **CTRL+C** to copy the text to the clipboard.

Paste Text

To paste cut or copied text, move the cursor to the location you want to move the text to and select **Edit|Paste** from the menu bar, click the **Paste** button on the standard toolbar, or press **CTRL+V**.

- Use the same methods to modify the style from the **Modify Style** dialog box that were used for the **New Style** box.
- To only rename the style, type a new name in the **Name** field.
- Click **OK** when you are finished making modifications.
- Click **Apply** to update the style in the document.

7.2.3 Find and Replace

If you need to find a particular word or piece of text, you can use the Find command. If you want to search the entire document, simply execute the Find command. If you want to limit your search to a selected area, highlight that area and then execute the Find command. After you have found the word or piece of text you are searching for, you can replace it with new text by executing the Replace command.

Find - Using the Menu

- Type the following:
Momi is from Punjab . She lives on the east side of Punjab. Her daughter attends Punjab High School.
- Highlight: " Momi is from Punjab. She lives on the east side of town. Her daughter attends Punjab High School."
- Choose *Edit > Find* from the menu.
- Type Punjab in the Find What field.
- Click Find Next.
Note that " Punjab " is highlighted.
- Click Find Next.
Note that the " Punjab " in "Punjab High School " Punjab is highlighted.

- Click Find Next. The following message should appear: "Word has finished searching the selection. Do you want to search the remainder of the document?"
- Click No.
- Click Cancel.

Find by Using Keys

- Highlight: "Momi is from Punjab. She lives on the east side of town. Her daughter attends Punjab High School."
- Press Ctrl-f.
- Follow steps 5 through 10 in the preceding section.

Similarly replace command can be performed by using menu and keys too.

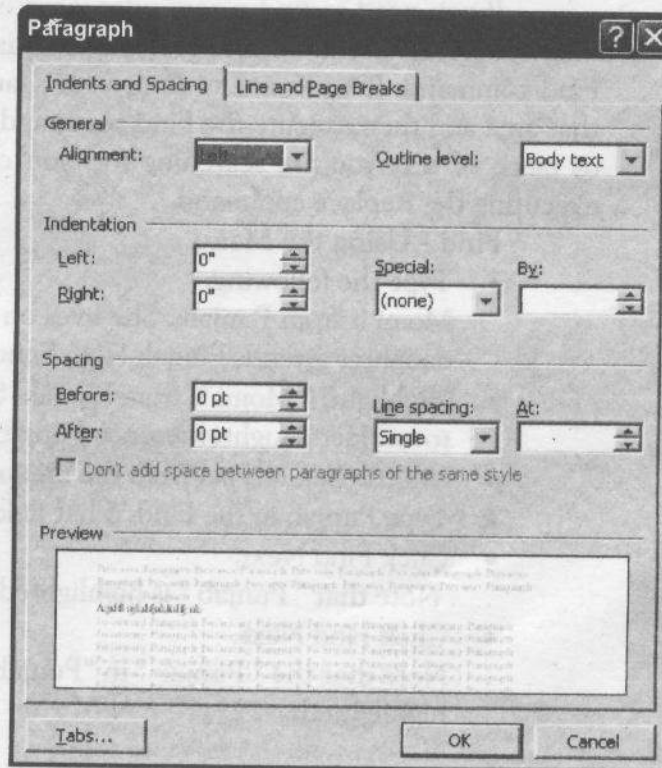
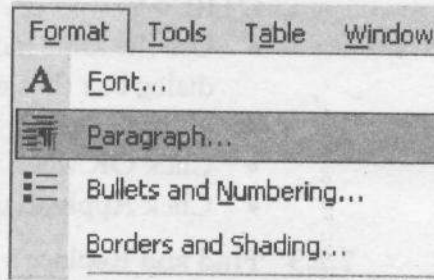
7.3 PARAGRAPH SPACING

To access the Paragraph formatting options, navigate to the menu bar, and select "Format" >> "Paragraph," or right-click within a paragraph.

A window will appear with options for modifying spacing and indenting. Here, you can choose to make the text in your document single or double spaced, as well as edit the margins for the document.

When you are formatting a paragraph, you do not need to highlight the entire paragraph. Placing the cursor anywhere in the paragraph enables you to format it. After you set a paragraph format, subsequent paragraphs will have the same format unless you change their format.

You will need text to work with to perform the exercises for this lesson, so type the following exactly as shown. End paragraphs where you see the end-of-paragraph



marker (¶). Press Enter once to end the paragraph, **but do not leave spaces between paragraphs**. You will set the space between paragraphs during the exercise. Do not press Enter to move to a new line -- Microsoft Word automatically wraps at the end of a line.

Sample Paragraphs ¶

We will use this paragraph to illustrate several Microsoft Word features. It will be used to illustrate Space Before, Space After, and Line Spacing. Space Before tells Microsoft Word how much space to leave before the paragraph. Space After tells Microsoft Word how much space to leave after the paragraph. Line Spacing sets the space between lines within a paragraph.

We will use this paragraph to illustrate some additional Microsoft Word features. It will be used to illustrate first-line indent. With first-line indent, you can indent the first line of your paragraph. We will also look at indentation. Indentation enables you to indent from the left or right margin of your document.

Space Before and Space After

Space Before sets the amount of space before the paragraph. **Space After** sets the amount of space after the paragraph. Following are the sample paragraphs with Space After set to 12 pt. The exercises that follow give you a chance to see how Space Before and Space After work.

Space After

Sample Paragraphs

We will use this paragraph to illustrate several Microsoft Word features. It will be used to illustrate Space Before, Space After, and line spacing. Space Before tells Microsoft Word how much space to leave before the paragraph. Space After tells Microsoft Word how much space to leave after the paragraph. Line Spacing sets the space between lines within a paragraph.

We will use this paragraph to illustrate some additional Word features. It will be used to illustrate first-line indent. With first-line indent, you can indent the first line of your paragraph. We will also look at Indentation. Indentation enables you to indent from the left and/or right margins of your document.

Line Spacing

Line Spacing sets the amount of space between lines within a paragraph. Single spacing is the default. The spacing for each line is set to accommodate the largest font on that line. If there are smaller fonts on the line, there will appear to be extra space between lines where the smaller fonts are located. At 1.5 lines, the Line Spacing is set to one-and-a-half times the

single-space amount. For double-spaced lines, the line spacing is set to two times the single-space amount.

First-Line Indent

This demonstrates how you can indent the left side of the first line of your paragraph, as in the following example.

The first-line indent feature indents the first line of the paragraph. The amount of the indent is specified in the *By* field. The remainder of the paragraph is indented by the amount specified in the *Indentation* field.

Indentation

Indentation allows you to indent your paragraph from the left or right margin. The following examples show different types of indentation.

We will use this paragraph to illustrate several Word features. We will illustrate *Space Before*, *Space After*, and *Line Spacing*. *Space Before* tells Word how much space to leave before the paragraph. *Space After* tells Word how much space to leave after the paragraph. *Line Spacing* sets the space between lines within a paragraph.

We will use this paragraph to illustrate some additional Word features. We will illustrate first-line indent. With first-line indent, you can indent the first line of your paragraph. We will also look at *Indentation*. *Indentation* enables you to indent from the left or right margins of your document.

Alignment

Microsoft Word gives you a choice of several types of alignment. Left-justified text is aligned on the left side. It is the default setting.

Sample Paragraph

This is a sample paragraph. It is used to illustrate alignment. Left-justified text is aligned on the left. Right-justified text is aligned on the right. Centered text is centered between the left and right margins. You can use *Center* to center your titles. Justified text is flush on both sides.

Right-justified text is aligned on the right side.

Right-Justified

Sample Paragraph

This is a sample paragraph. It is used to illustrate alignment. Left-justified text is aligned on the left. Right-justified text is aligned with on the right. Centered text is centered between the left and right margins. You can use *Center* to center your titles. Justified text is flush on both sides.

Centered text is centered between the left and right margins.

Centered

Sample Paragraph

This is a sample paragraph. It is used to illustrate alignment. Left-justified text is aligned on the left. Right-justified text is aligned with on the right. Centered text is centered between the left and right margins. You can use *Center* to center your titles. Justified text is flush on both sides.

Justified text is flush on both sides.

Justified

Sample Paragraph

This is a sample paragraph. It is used to illustrate alignment. Left-justified text is aligned on the left. Right-justified text is aligned with on the right. Centered text is centered between the left and right margins. You can use Center to center your titles. Justified text is flush on both sides.

The Right, Left, center and justified alignment can be performed by using key icon and menu For example if you want to center align of your MS-word document. It can be performed by using three different paragraph alignment methods.

- Paragraph alignment by using keys
- Paragraph alignment by using icon
- Paragraph alignment by using menu



Center - Using the Menu

- Highlight the first paragraph you typed, beginning with "We will use" and ending with "within a paragraph."
- Choose *Format > Paragraph* from the menu.
- Choose the *Indents and Spacing* tab.
- Click to open the Alignment pull-down menu.
- Click Centered.
- Click OK. The paragraph is now centered.

Justify and Center by Using Keys

- Highlight the text.
- Press Ctrl-e. The text is now centered.
- Press Ctrl-j. The text is now justified.

Justify and Center by Using the Icon

- Highlight the text.
- Click the Center icon . The text is now centered.
- Click the Justify icon . The text is now justified.
- Other paragraph alignments can be performed by using these three methods

Hanging Indent

The hanging indent feature indents each line except the first line by the amount specified in the By field, as shown in the example.

Hanging Indent: *The hanging indent feature indents the first line of the paragraph from the margin by the amount specified in the Left field. The amount in the Left field plus the amount specified in the By field indent all subsequent lines.*

When you begin typing the following paragraph, you might find that your paragraph is indented one inch on both sides. When you start a new paragraph in Microsoft Word, the setting from the previous paragraph carries over. If you wish, you can reset the indentation. If you choose not to reset the indentation, it will not affect your ability to perform the exercise.

- Type the following:
Hanging Indent: The hanging indent feature indents the first line by the amount specified in the **Left** field. Subsequent lines are indented by the amount specified in the **Left** field plus the amount specified in the **By** field. Highlight the paragraph you just typed.
- Choose *Format > Paragraph* from the menu.
- Choose the Indents and Spacing tab.
- In the Special field, click to open the pull-down menu.
- Click Hanging.
- In the By box, type 2.0".
- Click OK.
- Place the cursor after the colon following "Hanging Indent."
- Press the Tab key.

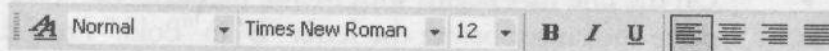
7.4 FONTS

In Microsoft Word, the term font is used to refer to the typeface of your text. You can change the font (the "family" of type you use for your text). This feature is illustrated in the following

Change the Font - Using the Menu

- Type the following:
Arial Courier Times New Roman
- Highlight "Arial."
- Choose *Format > Font* from the menu.
- Choose the Font tab.
- In the box below the Font field, click "Arial."
- Click OK.
- Highlight "Courier."
- Choose *Format > Font* from the menu.
- Choose the Font tab.
- In the box below the Font field, click "Courier New."
- Click OK.
- Highlight "Times New Roman."
- Choose *Format > Font* from the menu.
- Choose the Font tab.
- In the box below the Font field, click "Times New Roman."
- Click OK.
- Your text should now look similar to the following:
"Arial Courier Times New Roman"

Change the Font by Using the Formatting Toolbar



Highlight "Arial Courier Times New Roman."

Press Ctrl-spacebar. Ctrl-spacebar sets the formatting back to the default.

Highlight "Arial."

Click to open the Font pull-down menu on the Formatting toolbar.

Click "Arial."

Next, highlight "Courier New."

Click to open the Font pull-down menu on the Formatting toolbar.

Click "Courier New."

Next, highlight "Times New Roman."

Click to open the Font pull-down menu on the Formatting toolbar.

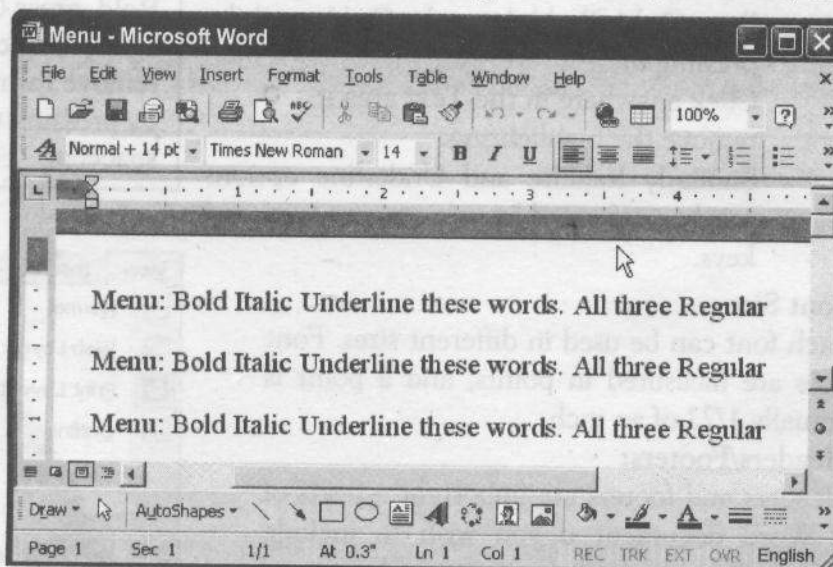
Click "Times New Roman."

Your text should now look similar to the following:

"Arial Courier Times New Roman"

Bold, Underline, and Italicize

You can bold, underline, or italicize when using Word. You also can combine these features -- in other words, you can bold, underline, and italicize



a single piece of text. In the exercise that follows, you will learn three different methods for bolding, italicizing, or underlining when using Word. You will learn to bold, italicize, or underline by using the menu, an icon, or the keys.

Bold – by Using the Menu, Icon and Keys

- On the line that begins with Menu, highlight the word Bold. To do so, place the cursor before the letter "B" in "Bold." Press the F8 key; then press the right arrow key until the entire word is highlighted.
- Choose *Format > Font* from the menu. The Font Dialog box opens.
- Click Bold in the Font Style box. Click OK to close the dialog box.
- Click anywhere in the text area to remove the highlighting. You have bolded the word bold.
- On the line that begins with "Icon," highlight the word "Bold." To do so, place the cursor before the letter "B" in "Bold." Press the F8 key; then press the right arrow key until the entire word is highlighted.
- Click the Bold icon **B** on the toolbar.
- Click anywhere in the Text area to remove the highlighting.
- On the line that begins with "Keys," highlight the word "Bold." To do so, place the cursor before the letter "B" in "Bold." Press the F8 key; then press the right arrow key until the entire word is highlighted.
- Press Ctrl-b (hold down the Ctrl key while pressing b). Click anywhere in the Text area to remove the highlighting.
- Similarly Italicize and Underline options can be performed by using menu, icon and keys.

Note: You can see the effect of your selection in the Preview window. To turn off the bold, click Regular.

Note: To turn off bold, highlight the text and press the Bold icon again

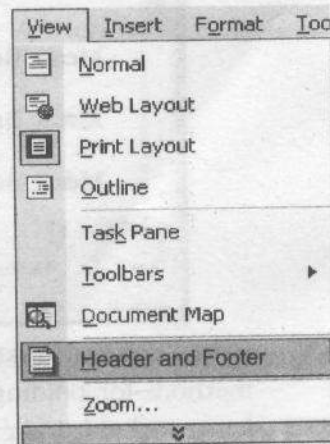
Note: To turn off Bold, press Ctrl-b again. You can also remove formatting by pressing Ctrl-spacebar.

Font Size

Each font can be used in different sizes. Font sizes are measured in points, and a point is actually 1/72 of an inch.

Headers/Footers:

Headers and footers are important aspects of a Word document if you wish to include information such as page numbers and headings on every page. To access the header and footer options, go to the menu bar and select "View" >> "Header and Footer."





Header
Insert header text here |



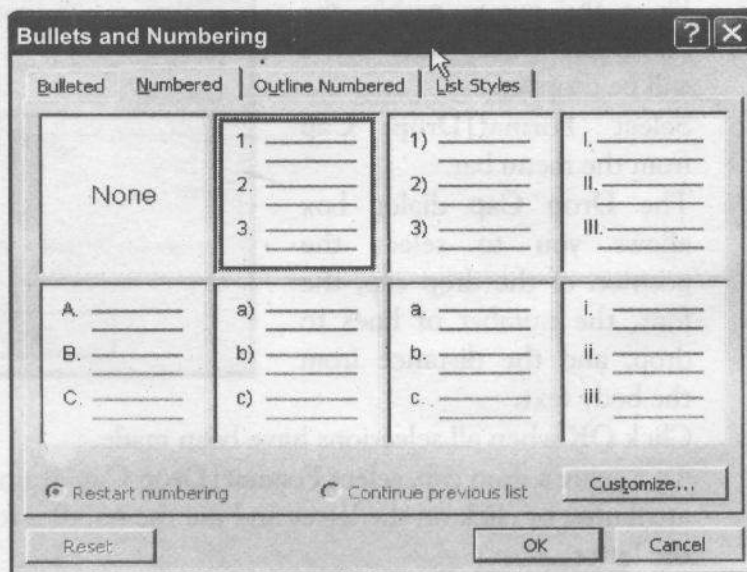
A dotted-line box called "Header" will automatically appear, as well as a sub-menu for formatting header and footer properties. The cursor will already be placed in the Header box. If you scroll down on your current page opened in Word, you will see a dotted-line box called "Footer." To add text in the Header or the Footer, simply click the cursor inside either one of the boxes, and type the text you want. To add page numbers to your document, click your cursor inside of the footer box. Then,



- Click on the icon shaped like a sheet of paper with a "#" inside. The page number will then be inserted and applied to all of the pages in your document.

Bulleted and Numbered Lists

- Click the **Bulleted List** button  or **Numbered List** button  on the formatting toolbar.
- Type the first entry and press **ENTER**. This will create a new bullet or number on the next line. If you want to start a new line without adding another bullet or number, hold down the **SHIFT** key while pressing **ENTER**.

Note: You can also type the text first, highlight the section, and press the **Bulleted List** or **Numbered List** buttons to add the bullets or numbers.



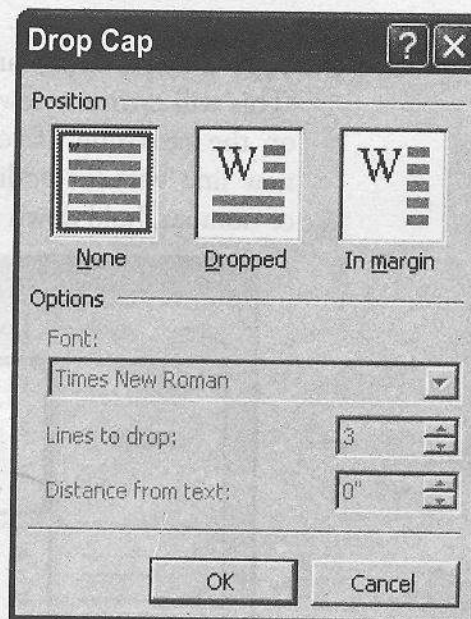
- Continue to typing entries and press **ENTER** twice when you are finished typing to end the list.
Use the **Increase Indent**  and **Decrease Indent**  buttons on the formatting toolbar to create lists of multiple levels. The bullet image and numbering format can be changed by using the **Bullets and Numbering** dialog box.
- Highlight the entire list to change all the bullets or numbers, or place the cursor on one line within the list to change a single bullet.
- Access the dialog box by selecting **Format|Bullets and Numbering** from the menu bar or by right-clicking within the list and selecting **Bullets and Numbering** from the shortcut menu.
- Select the list style from one of the seven choices given, or click the **Picture...** button to choose a different icon. Click the **Numbered** tab to choose a numbered list style.
- Click **OK** when finished.
- Tables are used to display data and there are several ways to build them in Word. Begin by placing the cursor where you want the table to appear in the document and choose one of the following methods.

Drop Caps

A drop cap is a large letter that begins a paragraph and drops through several lines of text as shown below.

Add a drop cap to a paragraph by following these steps:

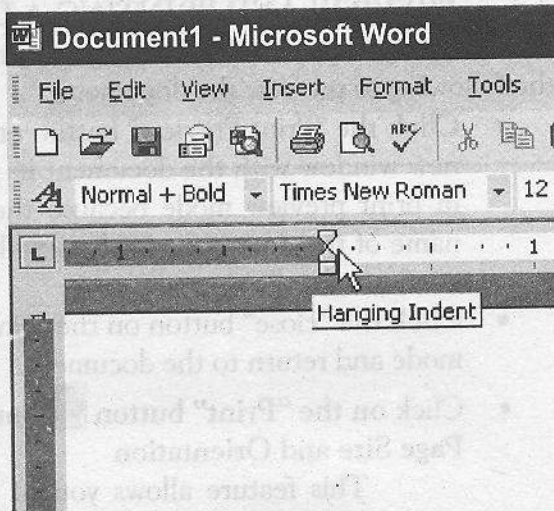
- Place the cursor within the paragraph whose first letter will be dropped.
- Select **Format|Drop Cap** from the menu bar.
- The **Drop Cap** dialog box allows you to select the position of the drop cap, the font, the number of lines to drop, and the distance from the body text.
- Click **OK** when all selections have been made.
- To modify a drop cap, select **Format|Drop Cap** again to change the attributes, or click on the letter and use the handles to move and resize the letter.



Page Margins

The page margins of the document can be changed using the rulers on the page and the **Page Setup** window. The ruler method is discussed first:

- Move the mouse over the area where the white ruler changes to gray.
- When the cursor becomes a double-ended arrow, click with the mouse and drag the margin indicator to the desired location.
- Release the mouse when the margin is set.
- The margins can also be changed using the **Page Setup** dialog box:

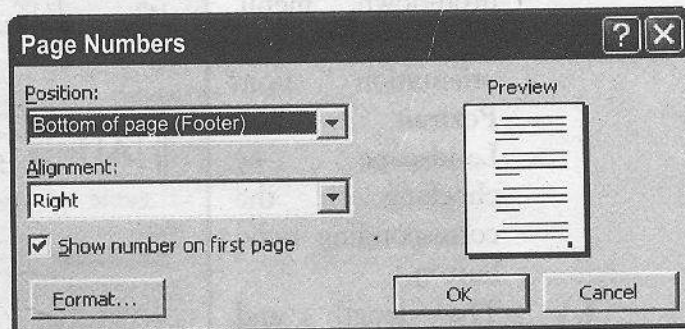


Insert Page Numbering

As noted earlier, you may insert page numbering into a document via the "Header and Footer" properties. You may also insert page numbering without using the header/footer properties.


Click "Insert" on the menu bar and then click "Page Numbers..." The "Page Numbers" dialog box appears on top of the document.

You may use the "Position:" drop box to position the page number at the top or bottom of the page. You may use the "Alignment:" drop box to align the number at the left, centre, right, inside or outside of the document. You may click to check the "Show number on first page" checkbox to allow page one to display number 1 or uncheck it to hide number 1 on the first page. Click the "Alignment" drop box to align the page number to the right. Then click "OK". See that the page number has been re-aligned to the right side of the page.




Non-printing Characters

Word can display special symbols on the screen that show where you have pressed enter, the spacebar or the Tab key in your document. These non-printing characters are useful when you are editing documents and making

sure that the text placement is exactly as you want it to be. To display these special symbols, click the Show/Hide button on the Standard toolbar. 

7.5 PREVIEW AND PRINTING A DOCUMENT

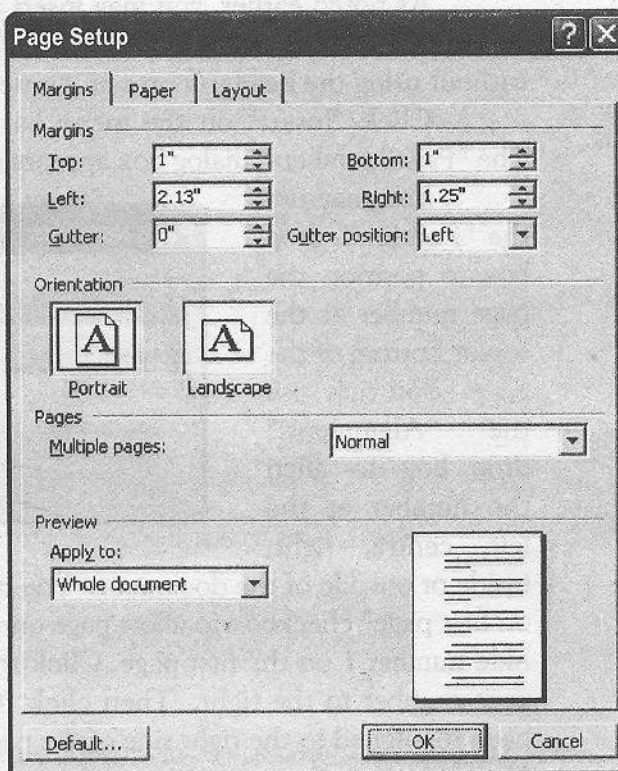
You can preview a document before sending it to the printer for output. Do the following to preview the document:

- Click the "Print Preview" button on the Standard tool bar. MS Word opens a new window with the document in preview mode. You will know that you are in print preview mode because the word "Preview" in parenthesis after the name of the document in the title bar. Notice that the entire page fits in the window as well.
- Click the "close" button on the print preview tool bar to exit print preview mode and return to the document.
- Click on the "Print" button  on the formatting tool bar

Page Size and Orientation

This feature allows you to change and control the orientation page within the Page Setup dialog box.

- Select **File|Page Setup** and choose the **Paper Size** tab.
- Select the proper paper size from the drop-down menu. Change the orientation from **Portrait** or **Landscape** by checking the corresponding radio button.
- The length and breadth of a page of a document is called page size. Page size of a document is defined in the Page Setup dialog box. To define page size:
 - Click File menu and select Page Setup



Select Paper Size tab and select appropriate page size from a list of given standard sizes. Some standard sizes and their dimensions are as follows:

Letter: paper size of 8.5 by 11 inches
Legal: paper size of 8.5 by 14 inches
A4: paper size of 8.27 by 10.5 inches

7.6 AUTOMATIC FEATURES

As you type text into a document, red or green wavy lines may appear under certain words. The **Automatic Spell Check** places non-printing red wavy lines to indicate that the word is not recognized by the Word dictionary. Green wavy lines indicate that the phrase may not be grammatically correct. In the example below, the word 'example' is spelled wrong, indicated by the red wavy line underneath the misspelled word.

Another automatic editing feature of Word is **AutoCorrect**. AutoCorrect automatically corrects commonly misspelled text as it is entered. For example "teh" is replaced with "the" as soon as the space bar is pressed. Word has a number of default AutoCorrect entries. Additional entries can be added by the user.

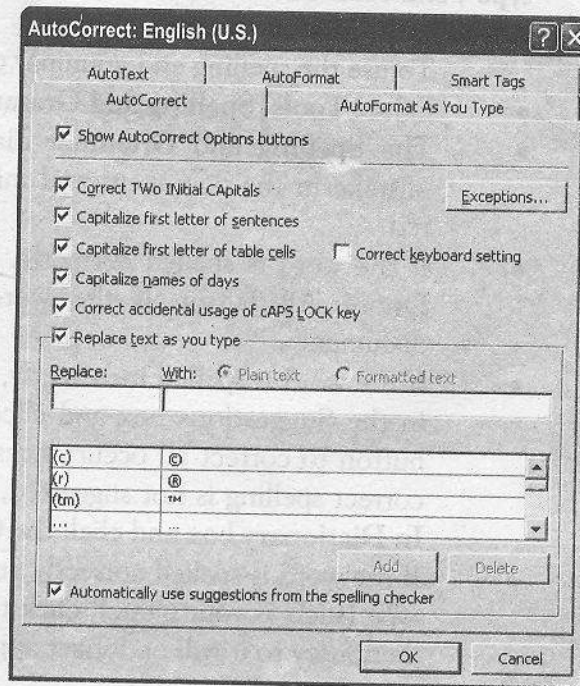
AutoText and AutoComplete

Another automatic feature of Word is AutoText. AutoText entries can be used to quickly assemble a document. As soon as you begin to type a commonly used word, a yellow AutoText flag will appear. To accept the flag, hit the Enter key. To override the AutoText entry, simply continue typing, and the flag will disappear.

AutoComplete

AutoComplete enables you to insert entire items — such as dates and AutoText entries — when you type a few identifying characters. In the example below, the date has become an AutoComplete entry.

Word automatically corrects many commonly misspelled words and punctuation marks with the AutoCorrect feature. To view the list of words that are automatically corrected, select **Tools|AutoCorrect**. This may be a hidden feature so click the



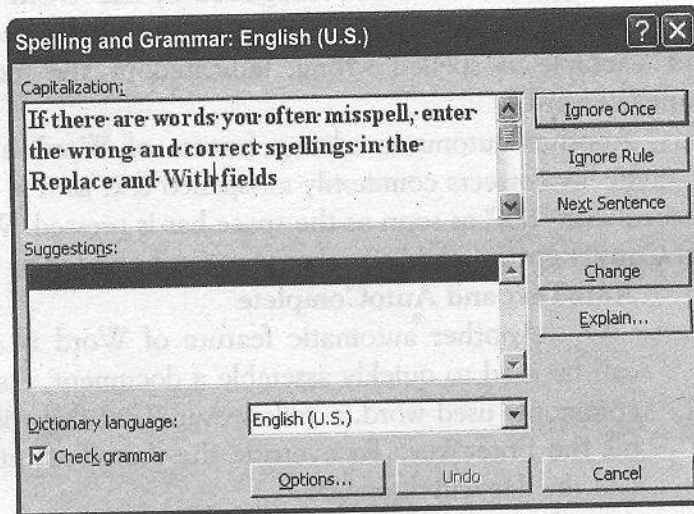
double arrows at the bottom of the **Tools** menu listing if the **AutoCorrect** choice is not listed.

Many options including the accidental capitalization of the first two letters of a word and capitalization of the first word of the sentence can be automatically corrected from this page. If there are words you often misspell, enter the wrong and correct spellings in the **Replace** and **With** fields.

Spelling and Grammar Check

Word will automatically check for spelling and grammar errors as you type unless you turn this feature off. Spelling errors are noted in the document with a red underline.

Grammar errors are indicated by a green underline. To disable this feature, select **Tools|Options** from the menu bar and click the **Spelling and Grammar** tab on the dialog box. Uncheck "Check spelling as you type" and "Check grammar as you type", and click **OK**.



To use the spelling and grammar checker, follow these steps:

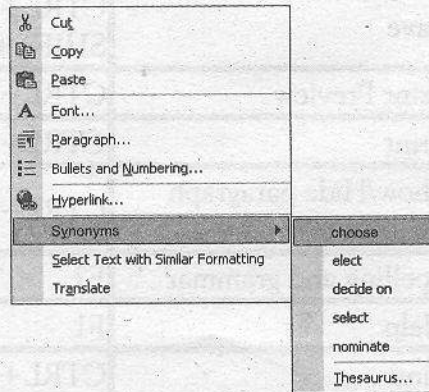
- Select **Tools|Spelling and Grammar** from the menu bar.
- The **Spelling and Grammar** dialog box will notify you of the first mistake in the document and misspelled words will be highlighted in red.
- If the word is spelled correctly, click the **Ignore** button or click the **Ignore All** button if the word appears more than once in the document.
- If the word is spelled incorrectly, choose one of the suggested spellings in the **Suggestions** box and click the **Change** button or **Change All** button to correct all occurrences of the word in the document. If the correct spelling is not suggested, enter the correct spelling in the **Not In Dictionary** box and click the **Change** button.
- If the word is spelled correctly and will appear in many documents you type (such as your name), click the **Add** button to add the word to the dictionary so it will no longer appear as a misspelled word.

As long as the **Check Grammar** box is checked in the **Spelling and Grammar** dialog box, Word will check the grammar of the document in addition to the spelling. If you do not want the grammar checked, remove the checkmark from this box. Otherwise, follow these steps for correcting grammar:

- If Word finds a grammar mistake, it will be shown in the box as the spelling errors. The mistake is highlighted in green text.
- Several suggestions may be given in the **Suggestions** box. Select the correction that best applies and click **Change**.
- If no correction is needed, click the **Ignore** button.

Synonyms

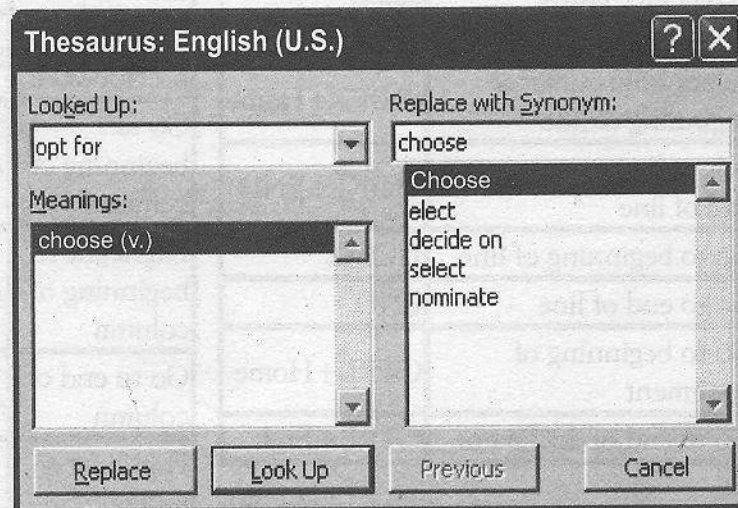
Word 2000 has a new feature for finding synonyms. Simply right-click on the word and select **Synonyms** from the shortcut menu. From the list of suggested words, highlight the word you would like to use or click **Thesaurus...** for more options.



Thesaurus

If you find yourself overusing a particular word and want to find a good synonym for it. To use the thesaurus, select **Tools | Language | Thesaurus** from the menu bar or select it from the **Synonyms** shortcut menu as detailed above. A list of meanings and synonyms are given on the windows. Double-click on the words in the

Meanings box or click the **Look Up** button to view similar words. Double-click words in the **Replace with Synonym** box to view synonyms of those words. Highlight the word you would like to add and click the **Replace** button.



Note: A plus sign indicates that the keys need to be pressed at the same time.

Action	Keystroke	Action	Keystroke
--------	-----------	--------	-----------

Document actions	
Open a file	CTRL+O
New file	CTRL+N
Close a file	CTRL+W
Save As	F12
Save	CTRL+S or SHIFT+F12
Print Preview	CTRL+F2
Print	CTRL+P
Show/Hide paragraph symbols	CTRL+*
Spelling and grammar	F7
Help	F1
Find	CTRL+F
Replace	CTRL+H
Go To	CTRL+G

Cursor movement	
Select all - entire document	CTRL+A
Select from cursor to beginning of line	SHIFT+Home
Select from cursor to end of line	SHIFT+END
Go to beginning of line	HOME
Go to end of line	END
Go to beginning of document	CTRL+Home
Go to end of document	CTRL+End

Formatting	
Cut	CTRL+X

Text Style	
Font face	CTRL+SHIFT+F
Font size	CTRL+SHIFT+P
Bold	CTRL+B
Italics	CTRL+I
Underline	CTRL+U
Double underline	CTRL+SHIFT+D
Word underline	CTRL+SHIFT+W
All caps	CTRL+SHIFT+A
Change case	SHIFT+F3
Subscript	CTRL+=
Superscript	CTRL+SHIFT+=
Make web hyperlink	CTRL+K

Tables	
Go to next cell	Tab
Go to previous cell	SHIFT+Tab
Go to beginning of column	ALT+PageUp
Highlight to beginning of column	ALT+SHIFT+PageUp
Go to end of column	ALT+PageDown
Highlight to end of column	ALT+SHIFT+PageDown
Go to	ALT+Home

Copy	CTRL+C
Paste	CTRL+V
Undo	CTRL+Z
Redo	CTRL+Y
Format painter	CTRL+SHIFT+C
Left alignment	CTRL+L
Center alignment	CTRL+E
Right alignment	CTRL+R
Justified	CTRL+J
Delete previous word	CTRL+Backspace
Apply bulleted list	CTRL+SHIFT+L
Indent	CTRL+M
Page break	CTRL+Enter

beginning of row	
Highlight to beginning of row	ALT+SHIFT+Home
Go to end of row	ALT+End
Highlight to end of row	ALT+SHIFT+End
Column break	CTRL+SHIFT+Enter

Miscellaneous	
Copyright symbol - ©	ALT+CTRL+C
Date field	ALT+SHIFT+D
Go to footnotes	ALT+CTRL+F
Show/Hide ¶	CTRL+SHIFT+8
Thesaurus	SHIFT+F7

Exercise

1. Fill in the Blanks:
- (i) Paper size 8.5 x 11 inches is called _____ size.
 - (ii) Font is also called a _____.
 - (iii) The keyboard shortcut to save a document in MS-Word is _____.
 - (iv) There are _____ items on the menu bar of MS-Word.
 - (v) Start button is on the _____ bar.
 - (vi) Shortcut command used for printing is _____.
 - (vii) In MS-Word's default font size is _____ point.
 - (viii) Ctrl + C is used for _____.
 - (ix) The Thesaurus can be used to look up synonyms and _____.
 - (x) The Office Assistant is _____ by default.

2. Choose the correct answer:
- (i) Which keyboard shortcut is used for double underline?
(a) Ctrl + Shift + D (b) Ctrl + [
(c) Ctrl + Shift + M (d) Shift + F3
 - (ii) MS Word is a _____ based program.
(a) Windows (b) System
(c) DOS (d) None of them
 - (iii) Which of the following bars provide the information about application software?
(a) Menu bar (b) Tool bar
(c) Status bar (d) Scroll bar
 - (iv) Press Ctrl + 2 keys for _____ spacing.
(a) Single (b) Double
(c) Triple (d) None of them
 - (v) Clicking on the Print icon _____ prints the document currently active in Word.
(a) manually (b) automatically
(c) (a) and (b) (d) None of them
 - (vi) Which of the following is used to select the paragraph?
(a) Single-click (b) Double-click
(c) Right-click (d) None of them
 - (vii) Ctrl + Y is used for
(a) Undo (b) Find
(c) Page break (d) Redo
 - (viii) Which one option is not in Edit Menu?
(a) Undo (b) Redo
(c) Find (d) Font
 - (ix) Which keyboard shortcut is used to make the selected word bold?
(a) Ctrl + Shift + B (b) Shift + B
(c) Alt + B (d) Ctrl + B

3. Write T for True and F for False statements.

- (i) A word processing program is used for calculating income, expenditure, balance sheet etc.
- (ii) Word processor is also called electronic typewriter.
- (iii) Ctrl + U underline the selection.
- (iv) Status bar is located at the top of application window.
- (v) Print Setup allows you only to select the printer.
- (vi) Ctrl + X cuts the selection.
- (vii) Normal view of a document does not display information such as header and footer.

- (viii) A drop cap is a large letter that begins a paragraph and drops through several lines of text.
- (ix) F2 key is used for help in MS-Word.
- (x) Spell Checking can be used to quickly check the spelling in a document.

4. What is meant by Title Bar?
5. Describe the use of clipboard in MS-Word.
6. How would you customize your toolbars?
7. What is meant by Drop Cap?
8. Differentiate between Formatting Toolbar and Standard Toolbar
9. Define the different views of the document.
10. How can you run the spell checker of MS-Word?
12. What is the use of Print Preview?
13. What are the methods for changing the margins in a document?
14. Differentiate between Alignment and Indent in MS-Word.

Answers

- | | | | |
|----|---|---|--|
| 1. | (i) Letter
(iv) 9
(vii) 12
(x) Displayed | (ii) Typeface
(v) Taskbar
(viii) Copy | (iii) Ctrl + S
(vi) Ctrl + B
(ix) antonyms |
| 2. | (i) a
(iv) b
(vii) b | (ii) a
(v) c
(viii) d | (iii) c
(vi) b
(ix) d |
| 3. | (i) F
(iv) T
(vii) T
(x) T | (ii) T
(v) F
(viii) T | (iii) T
(vi) T
(ix) F |

Glossary

A

Analysis: At this stage a problem is decomposed into sub-problems.

Algorithm: It is a step-by-step procedure to solve a problem in finite number of steps.

Arithmetic Operators: These operators are used to perform arithmetic operations on values (numbers)

Array: An array is a collection of contiguous memory locations which can store data of same type.

B

Built-in Functions: These are functions packaged with every implementation of BASIC.

C

Coding: The process of converting an algorithm to a program using a programming language is known as coding.

Constant: A constant is a named memory location whose contents can not be changed during the program execution. Like a variable, the value of a constant is accessed by its name.

Control Structures: These are fundamental structures of all high level programming languages. These are used to control the flow of execution of a program.

Conditional Transfer of Control: In this case, the program control switches to a specific line number by skipping one or more program lines depending on a certain condition.

D

Desk Checking: It is the process of carefully observing the working of an algorithm, on the paper, for some test data. Algorithm is provided a variable set of input for which the output is examined

Debugging: It is the process of finding and removing errors in the program

Deployment: Implementation of the program is also referred to as deployment of the program (see implementation also)

Documentation: It is a detailed description of a program's algorithm, design, coding method, testing, and proper usage.

Direct Mode: In this mode GW-BASIC commands are executed as they are typed.

Data files: Data files contain data and information needed for programs.

Drop Cap: It is a large letter that begins a paragraph and drops through several lines of text as shown below.

E

Execution: When a program has been loaded, the CPU follows instructions specified in it one-by-one. This process is called execution of the program.

EOF: At the end of every file, a special character is stored which is used to identify the end of file. This is known as End Of File marker.

F

Flowchart: It is the pictorial representation of an algorithm.

Field: A group of related characters to have a unit of information is called field or data field.

G

GW-BASIC: Genral Work Beginners All Purpose Symbolic Instructions Code. It is referred to as Gray Wolf Beginers All Pupose Symbolic Instructions Code

I

Implementation: Installation of the program in the user's environment is known as implementation of the program.

Indirect Mode: This mode is used to type programs. Programs are explicitly executed using RUN command.

IDE: Integrated Development Environment

Input Statements: These statements are used to provide data to the program.

Intrinsic Functions: Built-in functions are also called intrinsic functions (see also built-in function)

K

Keywords: Reserved words are also called keywords (see reserved words also)

L

Logic Error: This type of error occurs when a program follows a wrong logic.

Loading: Before execution, every program is brought into memory from the secondary storage. This process is known as loading the program.

Logical Operators: These operators are used to combine simple conditions to construct a compound condition.

Loop: Loop structure is used to repeat a number of statements up to a specified number of times or until a specified condition fulfilled.

M

Maintenance: It is an ongoing process of upgrading the program to accommodate new hardware or software requirements.

N

Numeric variable: A variable which can only store a numeric value is called numeric variables.

Numeric Constant: A constant which can only store a numeric value is called numeric constant.

Nested Loop: A loop inside the body of another loop is called a nested loop.

O

Operator's Precedence: It determines the order of evaluation of an operator in an expression.

Output Statements: These statements are used to get data from the program.

One-dimensional Array: This type of array is also known as linear array or vector array. It consists of only one row or column.

P

Problem-solving: Problem-solving is a skill to approach the solution of a given problem and it can be developed by following a well organized approach.

Programming: It is a problem solving activity that uses computer to solve a problem.

Program: A program is a set of instructions given to the computer in a programming language like BASIC, which the computer follow to solve a problem.

Program file: A program files contains instructions for the CPU.

Pixel: It is a picture element: A pixel is represented by one Dot on the screen.

Q

QBASIC: Quick BASIC

R

Requirements Document: It is an abstract description of the software which provides a basis for design and implementation. It also describes the features and restrictions of the system.

Runtime Error: This type of error occurs when the program directs the computer to perform an illegal operation such as dividing a number by zero.

Reserved Words: These are the words, which have predefined meaning in BASIC

Relational Operators: These operators are used to compare two values.

Record: Group of related fields is called record.

Random Access File: The data contained in a random access file is accessed directly from where it is physically stored on the disk.

Resolution: Number of pixels on the screen

S

Syntax: The grammatical rules of a programming language to write programs are referred to as syntax of that programming language.

Syntax Error: This type of error occurs when the program violates one or more grammatical rules of the programming language

String: A string can be defined as a sequence of characters enclosed in double quotations.

String Variable: A variable which can store strings is known as string variable. In GW-BASIC, a dollar sign (\$) is followed by the string variable's name.

String Constant: A constant which can store a string is known as string constant. Like a variable, a dollar sign (\$) is followed by the name of a string constant.

Selection Structure: It chooses which alternative program statement(s) to execute depending on a condition.

Subprogram: A larger program is divided into smaller, manageable piece of codes are called subprogram.

Subroutine: It is a self-contained set of statements that can be used from anywhere in a program. The subroutine performs a specific task, and then returns control to the part of the program that calls the subroutine.

Sequential Access File: The data contained in a sequential access file is accessed in the order in which it is physically stored on the disk.

Serif Font: These fonts have some extra decorative lines on the edges of the characters e.g. Times New Roman.

Sans-Serif Font: These fonts do not have extra decorative lines on the edges of the characters e.g. Arial.

T

Top-Down Design: The design in which a problem is divided into smaller sub-problems, to approach the solution is called top-down design.

Testing: The process of evaluating the program to verify that it works as desired is known as testing.

Two-dimensional Array: It is also known as table or matrix. It can be thought of as a 2×2 matrix consisting of two rows and two columns.

Thesaurus: Dictionary feature in MS-Word

U

Updating: It is the process of introducing minor improvements in the program to meet the changing requirements of the user.

Unconditional Transfer of Control: In this case, the program's control switches to a specific line by skipping one or more lines without depending on a condition.

User-define Function: A user-defined function is written by the programmer to meet certain programming requirements.

V

Variables: These are memory locations which are used to store data and program's computational results during the program execution.

INDEX

A

Analysis, 1, 2
Algorithm, 4-9
Auto, 20
Arithmetic operator, 28
Assignment operator, 30
Array 53-58
ABS function, 61
Auto Text, 113
Auto Complete, 113

B

Built-in functions, 61
BEEP function, 64

C

Coding, 1, 3
Character set of BASIC, 16
Constant, 18
CLEAR, 20
CLS, 20
CONT, 27
Concatenation Operator, 30
Control structures, 41
> CHR\$ function, 67
COLOR statement, 81
CIRCLE statement, 85

D

Desk Checking, 2
Documentation, 4
Debugging, 3, 9
Direct mode, 13-14
DELETE, 21
DATA statement, 33
DIM, 55
DATE\$ function, 65
DRAW statement, 85
Drop Caps, 110

E

EDIT, 21
END, 27

F

Flowchart, 2, 8-10
FILES command, 22
FOR-NEXT, 48
FIX function, 63

G

GW-BASIC, 13
GOTO statement, 41-42
GOSUB...RETURN, 69

I

Implementation, 2, 4
Indirect mode, 13
INPUT, 35
IF Statement, 45
IF-ELSE Statement, 46
INT function, 62

K

Keyword, 17
KILL, 22

L

Logic error, 3
LSIT, 23
LOAD, 23
LLIST, 26
LPRINT, 26
Logical operator, 29
LOG function, 64
LEN function, 65
LEFT\$ function, 67
LINE, 84

M

Maintenance, 4, 9
MKDIR, 24
MID\$ function, 66
Medium Resolution Mode, 79

N

Numeric variable, 18
Numeric constant, 18
NAME, 24
Nested loop, 50

O

Operator precedence, 31
ON GOTO statement, 43
ON ERROR GOTO, 44
One-dimensional array, 56

P

Problem-solving, 1
Program, 3
Programming, 1, 3
PRINT, 36
PRINT USING, 37
Pixel, 79
PALETTE, 83
PSET, 83

Q

QBASIC, 13

R

Requirements Document, 2
Runtime error, 3
RUN command, 25
Reserved word, 17
RENUM, 24
RMDIR, 25
REM, 27
Relational operator, 29
READ, 33
RESTORE, 34
RND function, 63
RIGHT\$ function, 67
Random file, 74

S

Syntax Error, 3
Step-form Algorithm, 5
String variable, 18
String constant, 19
SAVE, 25
SYSTEM, 26
STOP, 28
Sequence structure, 41
SQR function, 62
SIN function, 62
SPC function, 64
SPACE\$ function, 66
Subroutine, 69
Sequential file, 70
SCREEN statement, 80

T

Top-down design, 2
Testing, 3-4
Two-dimensional array, 57
TAB function, 63
Text Mode, 79
Thesaurus, 115

U

Update, 1, 3

V

Variable, 17
VAL function, 65

W

WHILE-WEND, 49
Word processing, 89