CHAPTER

16 FUNDAMENTAL OF QBASIC

As already discussed, you can enter the program without line numbers in the QBASIC session and begin to run or execute. The pull-down menus help in writing program. Some basic features of BASIC constant, variable, etc. are discussed below.

16.1 Variable

A variable is an entity that can be changed during program execution. More precisely, the content of a variable can be changed during program RUN. Some variable have different meanings in different programs. There are two kinds of variable. They are:

(a) Numeric variable
(b) String variable

16.1.1 Numeric Variable: The name given to a number can be called as numeric variable since the actual number referred to is changeable. Numeric variables may be single letter of the alphabet followed by one of the numerals from 0 to 9. Thus, any of the following 286 entities could be used as numeric variables.

A, B, C, ..., Z (single character = 26)
A0, A1,....., A9
B0, B1, ..., B9
.....
.....
Z0, Z1, ..., Z9 (two characters = 260)

Example 1

```
LET AMT = 100
```

Here, AMT is the variable, and 100 is the constant. The "=" sign is assignment operator, that does not imply equality. Numeric variables can also be **an integer**, **a single precision** or **a double precision** numeric variable. Precision indicates accuracy to a specified number of digits.

Example 2

```
A%= 555
AMT% = 100
```

Here, a variable name followed with % sign indicates an integer.

Single precision is sufficiently accurate for most applications. However, the *seventh significant digit of a single precision number will not always be accurate*. The following variables are single precision variables.

A = 300
A! = 567
B = 456
B! = 90, etc.

A variable name followed with ! sign indicates single precision variable.

Double precision variables, though being accurate, uses more memory space and take more calculation time. The following variables are double precision variables.

A# = 45678901
B# = 456234

A variable name followed with # sign indicates double precision.
Different type of variables require different amounts of memory storage.

Variable	Required bytes of storage
Integer	2
Single precision	4
Double precision	8

16.1.2 String Variable: In addition to storing numeric values in variables, strings of alphabetic characters, special characters and digits can also be stored. Variables that store alphanumeric data such as this are called **string variables** and are named differently from numeric variables.

String variable names must have a dollar sign as the last (rightmost) character of the variable name. Following is a summary of naming rules for string variables. Variable names:

- Must be terminated by a dollar sign ($).
- May be any length, with only the first forty characters being significant.
- May use combinations of letters, digits, and periods.
- Must begin with a letter.
- Must not have reserved word such as LET or PRINT.

Using LET$ as a variable name will cause a syntax error. Hence, a string variable is declared with a dollar sign '$' as a suffix to it. It occupies 1 byte of storage for each character.

When a numeric or string value is used *literally* and *not* referred to by a scalar variable name, then its value cannot be changed and is therefore known as a **constant or literal**. In the expression **Y = 2*X +0.5** ; *X* and *Y* are scalar variables and 2 and 0.5 are numeric literals. In the expression **PRINT "ADDRESS";A$**, *ADDRESS* is a string literal and *A$* is a string variable.

16.2 Constant

A constant is a fixed entity that does not change. A constant is a fixed notation. The notation may be a number or string. There are two types of constants. They are:

1. Numeric constant &
2. String constant

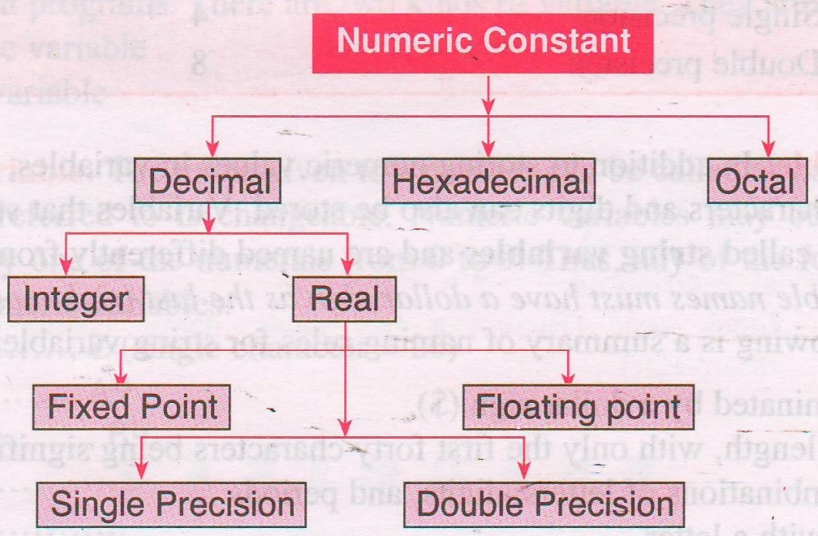
16.2.1 Numeric Constant: Numeric constants are the values. It is another name for numbers. Actual numbers assigned to variables are called constants. In the example

LET AMT = 100

AMT is the variable, and 100 is the constant. Numeric constants are formed with combinations of the numeric digits (0-9) and an optional decimal point. An operational sing (+ or -) may precede the number, but it is optional for positive number. No other characters are allowed in constants, which rule out the use of commas, dollar sign, or percent signs. Examples valid numeric constants are:

100	positive integer
12.6	positive real number
-124	negative integer
-126.8	negative real number
1.5E+7	1.5×10^7
1.2E-3	1.2×10^{-3}

Above listed numeric constants can be detailed as below.



Every constant is a unique. The constants are tabulated below.

Integer	Whole numbers between –32768 and +32767. They do not contain any decimal points.
Fixed point	Positive or negative real numbers that contain decimal points.
Floating point	Positive or negative numbers represented in exponential form (similar to scientific notation). A floating point constant number is represented as a value between 1 and 10 multiplied by the power of 10.23450000 which is 2.345×10^7 is written as 2.345E7 in GWBASIC. The allowable range for floating point constants is 3.0×10^{-39} to 1.7×10^{38}.
Hexadecimal	Hexadecimal numbers are numbers with prefix &H. for example, &H45, &H20F, etc.
Octal	Octal numbers are numbers with prefix &O or &. For example, &O123, &123, etc.

A numeric constant can be either integer or real number. The real number can be sub-divided into two more sub groups, single precision and double precision numbers. Inside computer's memory integer numbers are stored as whole numbers. 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 (only 16 digits may be accurate).

Note: ***A single precision constant*** *is a numeric constant with seven or fewer digits and or with exponential form using* ***E*** *and/or with a trailing* ***exclamation point (!)***. ***A double precision constant*** *is a numeric constant with eight or more digits and/or with exponential form using* ***D*** *and/or with a trailing number OR* ***hash sign (#).***

Example 3: *Single precision constants*

456
-1.02E-05
11.5!
1234.0

Example 4: *Double Precision constants*

123456789
-1.02345D-05
1234.0#
1234567.9876

16.2.2 String Constant: Any combination of alphabets, numbers, blank spaces and other special characters available enclosed inside double quotation marks is a string constant. A string constant can be of maximum 255 characters length. String constants are also called *string literal.*

Example 5: Examples of string constants are:

"CLASS 10"
"COMPUTER STUDIES"
"KATHMANDU, NEPAL"
"!@#$%^&*"
"1234567890"
etc.

16.3 BASIC Words

There are two types of words in BASIC. They are:

(a) Reserved word and
(b) User defined word

16.3.1 Reserved Word: Each reserved word has a specific meaning to the BASIC interpreter and compiler. The reserved words of BASIC are statements, commands and functions. The words are defined in BASIC interpreter. The user cannot use them as variable. A reserved word has the same meaning even if it is used in different programs.

16.3.2 User Defined Word: User defined words are the variables that user uses in his program. To represent numbers and string literal, the user may use variables. These are called user defined words. A user-defined word may have different meaning in different programs. In one program, the user defined word P may be denoted as principal amount. In another program, P may be referred to as pressure. That means, user defined words are program dependent, whereas reserved words are program independent. User defined words are classified as numeric variable and string variable.

16.4 Operator

An operator is a symbol indicating an operation. It indicates the process to be performed. It remains between two operands. Three kinds of operators are basically used in BASIC. They are:

(a) Arithmetic operator
(b) Relational operator
(c) Logical operator

16.4.1 Arithmetic Operator: Arithmetic operators are the symbols with the help of which calculations are carried out. The following arithmetic operators are used in BASIC.

Operation	Operator	Example	Meaning
Addition	+	9+6	Add 9 to 6, result will be 15
Subtraction	-	9-6	Subtract 6 from 9, result will be 3
Multiplication	*	9*(-6)	Multiply -6 by 9, result will be -54
Division	/	25/5	Divide 25 by 5, result will be 5
Integer division	\	25\4	Divide 25 by 4 and give integer part of result, result will be 6.25 = 6
Modulus	MOD	36 MOD 7	Divide 36 by 7 and give remainder as result, result will be 1
Exponentiation	^	2^4	Take 2 to the 4^{th} power, result will be 16.

Note: *GWBASIC uses MOD as an arithmetic operator. Similarly, it can be used as an operator in QBASIC.*

Above mentioned ^ is named as **circumflex**. In some software, it is named as caret, like in Lotus 123. The circumflex is used for exponentiation (raised to the power).

16.4.2 Relational Operator: Control of the loop execution is based on a condition. To form the conditions, there are six relational operators used to compare two values. The result of the comparison is either true or false.

The relational operator brings about the comparison between two numeric expressions or string expressions. No comparison can be made between two different kinds of expression such as numeric constant and string constant or numeric variable and string variable. The comparison can be made between two similar kind of expressions. The operators and their meanings are given below.

Operator	Meaning	Numeric Exp	String Exp
=	Equal to	A = B	A$=B$
<	Less than	A < B	A$ < B$
<=	Less than or equal to	A<=B	A$<=B$
>	Greater than	A>B	A$>B$
>=	Greater than or equal to	A>=B	A$>=B$
<>	Not equal to	A<>B	A$<>B$

16.4.3 Logical Operator: Like relational operators, BASIC also supports logical operators. In decision making, logical operators are used. The logical operators are:

(a) AND conjunction
(b) OR disjunction and
(c) NOT logical

In case of AND operator, both conditions must be true for the entire condition to be true. For OR operator, if one or other condition is true, or both are true, the entire condition is true. NOT logical operator reverses the condition, so that a true condition will evaluate false and vice versa.

(a) AND Operator

Syntax

result=numeric-expression1 AND numeric-expression2

The AND logical-conjunction operator compares corresponding bits in two numeric expressions and sets the corresponding bit in the result to 1 if both bits are 1. The AND operator use this "truth table":

First condition	Second condition	Result
F	T	F
T	F	F
F	F	F
T	T	T

Example 6

```
WHILE ANS$ <> "YES" AND ANS$ <> "NO"
INPUT "ENTER 'YES' OR 'NO'"; ANS$
WEND
```

In this example, if ANS$ contains *neither* YES or NO, then the condition will be true. IF ANS$ contains NO, then the first condition is true, the second condition is false, and the entire condition is false. The compound condition also evaluates false if ANS$ contains YES.

When two conditions are joined by AND, each condition is evaluated independently. Then, if both conditions are true, the entire compound condition is true. If one or the other is false, the entire condition tests false.

(b) OR Operator

Syntax

result = numeric-expression1 OR numeric-expression2

The logical "inclusive or" operator compares corresponding bits in numeric-expression1 and numeric-expression2, then sets the corresponding bit in the result according to the following table:

Bit in First Expression	Bit in Second Expression	Bit in Result
1	1	1
1	0	1
0	1	1
0	0	0

Example 7

```
INPUT "ENTER A NUMBER BETWEEN 1 & 10";L
WHILE NUM <1 OR NUM > 10
INPUT "VALUE NOT IN RANGE 1-10, PLEASE  RE-ENTER";L
WEND
```

If the value of NUM is less than 1 or greater than 10, then the message will print. *Note that the NUM must be repeated in the second condition.* If this were written IF NUM <1 OR >10, BASIC would *not* assume that you meant NUM>10, but produce a SYNTAX ERROR.

When a compound condition contains an OR, the two conditions are tested independently. Then if one condition *or* the entire condition evaluates true.

(c) NOT Operator

Syntax

result = NOT numeric-expression

The logical-complement operator evaluates each bit in numeric-expression, then sets the corresponding bit in the result according to the following table:

Bit in Expression	Bit in Result
1	0
0	1

This inverts the bit values of any variable. If an integer variable has the value 0 (false), the variable becomes -1 (true), and vice-versa.

Example 8

```
WHILE NOT RESPONSE$ ="QUIT"
GOSUB AA                    'CALCULATE
GOSUB AB                    'PRINT
INPUT 'ENTER NEXT ONE ('QUIT' TO END)";RESPONSE$
WEND
```

In the example, if RESPONSE$ holds anything other than QUIT the condition evaluates true. The NOT operator reverses the truth of a condition and applies only to the condition it precedes.

The use of the NOT operator is generally confusing and difficult to evaluate and is therefore not recommended. Conditions usually can be constructed without the use of NOT (e.g., RESPONSE$ <> "QUIT" rather than RESPONSE$ ="QUIT").

16.4.4 Combining the Logical Operators: Combinations of AND and OR can be used in a single compound condition. When the operators are combined, the order of evaluation becomes important. The condition surrounding the AND is evaluated first, then the condition surrounding the OR is tested.

WHILE SCORE >=0 AND SCORE <=100 OR SCORE =999

If the SCORE entered is in the range 0 - 100 or if 999 is entered, then the entire condition is true. Compound conditions can be useful, but should only be used when thoroughly understood. The logic should be carefully worked out for all possible combinations of multiple ANDs, ORs, and NOTs before use. Parenthesis may be used to alter the order of evaluation, with the conditions within the parentheses being evaluated first. Extra use of parentheses can also make the statement easier to understand.

There are some more operators frequently used in advanced programming as well as in intermediate level. These are listed below for your reference. These operators are (a) EQV, (b) IMP and (c) XOR.

(a) EQV Operator

Syntax

result = numeric-expression1 EQV numeric-expression2

Use the logical-equivalence operator to compare corresponding bits in numeric-expression1 and numeric-expression2 and then set the corresponding bit in the result according to the following table:

Bit in First Expression	Bit in Second Expression	Bit in Result
1	1	1
1	0	0
0	1	0
0	0	1

(b) IMP Logical Operator

Syntax

result = numeric-expression1 IMP numeric-expression2

The logical-implication operator, IMP, compares corresponding bits in *numeric-expression1* and *numeric-expression2* and then sets the corresponding bit in the result according to the following table:

Bit in First Expression	Bit in Second Expression	Bit in Result
1	1	1
1	0	0
0	1	0
0	0	1

(c) XOR Operator

Syntax

result = numeric-expression1 XOR numeric-expression2

The logical "exclusive or" operator compares corresponding bits in numeric-expression1 and numeric-expression2, then sets the corresponding bit in the result according to the following table:

Bit in First Expression	Bit in Second Expression	Bit in Result
1	1	0
1	0	1
0	1	1
0	0	0

14.5 Order of Preference

While performing the arithmetic calculations, computer follows a fixed order in which these operations are performed. This hierarchy or order of calculation is also called order of precedence. This is detailed below.

1st Order	All the calculations within parenthesis ( ) are calculated first.
2nd Order	All the exponentiations or power are calculated next.
3rd Order	All multiplication and division are executed next. In a sequence multiplication and division are calculated from left to the right of the equation.
4th Order	All integer divisions (\) are executed next.
5th Order	All modulus (MOD) operations are executed next.
6th Order	All the addition and subtraction are done next. In a sequence, they are calculated from left to right.

If you have arithmetic, relational and logical operators in the same statement, then BASIC first evaluates all the arithmetic operation, next it checks all the relational part and at last it evaluates all the logical operators.

For example in a statement like

Example 9:

```
IF X+4 > Y+8 OR X$="A" AND Z*5 = 20
THEN PRINT "VERY GOOD"
```

First, all the mathematical operations X+4, Y+8, Z*5 are calculated. Next, all the relational operators >, <, =, <>, etc,. are checked. And finally AND and OR logical operators are evaluated. If you write the above example as

```
IF (X+4 > Y+8 OR X$="A") AND Z*5 = 20
THEN PRINT "VERY GOOD"
```

You will get a completely different result. Another example explains more about it.

```
LET WHAT = SQR(INT(ABS(A/B)) * 2) + A^2
                ↑ ↑   ↑  ↑   ↑   ↑ ↑
Order of evaluation  5 3   2  1   4   7 6
```

16.6 Mathematical Expression and BASIC Expression

The representation of algebraic expression into BASIC expression needs some rules. The BASIC interpreter understands the rules of arithmetic but the symbols used are different. When programs are written, the algebraic expressions need to be converted into BASIC expression. Some examples are given below.

Algebraic Expression	BASIC Expression
$A \times B$	A*B
$\frac{A}{B}$	A/B
A(A+B)	A*(A+B)
$\frac{A+B}{C}$	(A+B)/C
A^{10}	A^10
$\frac{-b+\sqrt{b^2-4ac}}{2a}$	(-b*+(b^2-4*a*c)^0.5)/(2*a)
$S= ut + \frac{A}{B} at^2$	s = u*t+1/2*a*t^2

MAIN POINTS TO REMEMBER

1. A variable is an entity that can be changed during program execution. There are numeric and string variables. Numeric variables may be of integer, single precision or double precision.
2. String variable can have maximum of 40 character length.
3. A text enclosed in double quotes is known as literal constant.
4. A numeric value used in expression is numeric literal constant.
5. Numeric constants are not enclosed in quotes whereas string literal constants are enclosed in double quotes.
6. Numeric constants are of the following types: integer, single precision & double precision.
7. Hexadecimal numbers are numbers with prefix &H. Eg. &H4F
8. Octal numbers are numbers with prefix &O or &Eg &0123, &123.
9. BASIC reserved words are keywords of BASIC that do not change in any program.
10. User defined words are the variables that have different meaning in different programs.
11. An operator is a sign or symbol between two operands indicating an operation.
12. Arithmetic operators are: +, –, *, /, \, MOD, ^
13. Relational operators are: =, <, <=, >, >=, < >
14. Logical operators are: AND, OR, NOT.