# Control Structures

Decision structures - IFs and SELECT CASE

A powerful programming tool that will be used in virtually all programs from this point forward are control structures. Control structures allow programmers to determine whether or not specific statements are executed. The two types of control structures are ***decision structures*** and ***loops***. Decision structures are used to make comparisons. Two types of decision structures are IF statements and the SELECT CASE statements.

## Decision structures

### (#1) The Inline IF Statement

In real life, all of us are constantly making decisions. Many decisions are based on a particular situation, and some are based on past experiences that are taken into account. We start making decisions when we wake up in the morning. Is there time for a shower and breakfast? If it's raining outside, what coat should you wear? If I don't feel well, should I stay in bed and blow-off my programming class?

Computer programs also need to handle decisions. The most simple form of IF statement is the inline IF. This structure statement allows us to make a decision based on some criteria then, immediately act on that decision.

```
IF age = 18 THEN PRINT "Be sure to register to vote."
```

Although the in-line IF is a very effective structure for decision making, it often falls short of our need to process multiple statement lines after a decision is made.

### (#2) The Block IF Statement

In QBasic the block IF statement is useful if we need to process multiple statement lines after a decision is made. For example:

```
IF age = 18 THEN
      PRINT "Be sure to register to vote."
END IF
```

The statement between the keywords THEN and END IF -- PRINT "Be sure to register to vote." -- is executed only if age equals 18. Otherwise, no action is taken. Execution then continues to the next statement after the END IF. Any number of statements can be included in the body of the block IF; this group of statements is referred to as a statement block. The following block IF has two statements in its statement block; both are executed if "age = 18"; is TRUE.

```
IF age = 18 THEN
      PRINT "Happy 18th Birthday!"
      PRINT "Be sure to register to vote."
END IF
```

**Notice** that the statement block has been indented. QBasic does not require this indentation; however, it is a standard convention and greatly improves readability.

## Relational/Comparison Operators

The execution of the block IF is controlled by a Boolean (or logical) expression which is an expression that is either TRUE or FALSE. The block IF uses a **relational/comparison operator** to compare two expressions, determining whether the first expression is greater than, equal to, or less than the second one. The following table shows the relational/comparison operators used in QBasic.

| | | |
|---|---|---|
| **<** | Less than | 1 < 10 |
| **<=** | Less than or equal to | "Y" <= "Z" |
| **>** | Greater than | 1043.4 > 1043 |
| **>=** | Greater than or equal to | "SAMUAL" >= "SAM" |
| **=** | Equal to | 10 + 4 = 14 |
| **<>** | Not equal to | "Jones <> "James" |

The values of the expressions can be either numeric or character strings. However, both expressions must be of the same type. Therefore, the following statement comparing two character strings is ***valid***:

```
IF "Jon" < "Jonathan" THEN
     PRINT "This is a nickname."
END IF
```

However, the next statement is ***invalid*** because it attempts to compare a character string to a number:

```
IF "Jon" < 5 THEN
     PRINT "Wrong name."
END IF
```

It's easy to understand how the computer can compare numbers, but you may be wondering how it compares character strings. All computers assign an internal ordering to the set of characters they are able to recognize. This ordering is referred to as the computer collating sequence. Many different collating sequences are available, depending on the type of computer being used. Most computers use the **ASCII** (American Standard Code for Information Interchange) standard. From the **ASCII** table for example, we can determine that an uppercase A is less than an uppercase D because the **ASCII** value of A (65) is less than the **ASCII** value of D (68). In addition, notice that the **ASCII** value of all uppercase letters are always less than lowercase letters.

When the computer compares two character strings it compares each character, from left to right. The first character of one string is compared to the first character of the other string, then the second character of each string is compared, and so on until a different character in the second string (or the end of either string) is reached. For example, the expression

```
"Chase" < "Chasz"
```

is TRUE because the value of e is less than the value of z in the **ASCII** table.

When two strings of unequal length are compared, and all the letters of the shorter string match the corresponding letters of the longer string, the shorter string is considered to be less than the longer string. Thus, the following expression is TRUE:

```
"HOPE" < "HOPEFUL"
```

Be aware that leading and trailing blanks are significant. Because a blank has a smaller **ASCII** value (32) than any letter or digit, the following expressions are TRUE:

```
" CAT" < "CAT"          (Blank < C)
"PAY" < "PAY  "         (Second string has 5 characters)
```

## (#3) the block IF statement : as a single-alternative decision structure

The block IF statement is used to check a single condition. It is called a single-alternative decision structure because it checks only one condition. Action is taken only if that condition is TRUE.

Let's look at developing a program using the block IF statement:

The local music store is having a sale. All CDs are marked down to $10.00. If you buy six or more CDs, you get an additional 10 percent discount off the total price. The number of CDs being purchased should be entered by the user during program execution. Next, the program determines the regular price (number * 10). The following block IF snippet charges the customer only 90 percent of the regular price if more than five CDs are being purchased:

```
IF number >= 6 THEN
      cost = cost * .9
END IF
```

This statement could also be written:

```
IF number > 5 THEN
      cost = cost * .9
END IF
```

The end result will be the same either way.

Following is the full program code for this example:

```
DIM number AS INTEGER
DIM cost AS INTEGER
DIM format1 AS STRING
CLS
INPUT "Enter the number of CDs: ", number
cost = number * 10
IF number >= 6 THEN
    cost = cost * .9
END IF
format1 = "\                       \ $$##.##"
PRINT USING format1; "The cost of the CDs is: "; cost
END
```

## (#4) The Double-Alternative Decision Structure (IF ELSE) : one action is taken if the condition is TRUE and another if it is FALSE

The block IF statements discussed so far have been single-alternative decision structures. In a single-alternative decision structure, an action is taken only if the condition is TRUE; otherwise, execution simply continues to the next statement. In double-alternative decision structures one action is taken if the condition is TRUE and another if it is FALSE. Here's an example of how a block IF statement can be used to write a double-alternative decision structure:

```
IF speed <= 65 THEN
     PRINT “You are going”; speed; “miles an hour.”
ELSE
     PRINT “Pull over!”
     tickets = tickets + 1
END IF
```

The statement following the THEN is executed if the condition is TRUE; otherwise, the condition following the ELSE is executed. Notice that in this example the THEN statement block contains a single statement whereas the ELSE statement block consists of two statements. Any number of statements can be contained in either block.

Let's alter the program by changing the pricing arrangement used in the music store sale:

| | |
|---|---|
| 1 to 5 CDs | $10.00 each |
| 6 or more CDs | $ 9.75 each |

This is one way the new code can be written:

```
IF number >= 6 THEN
     cost = number * 9.75
ELSE
     cost = number * 10
END IF
```

If six or more CDs are purchased, the cost is $9.75 each; otherwise, the cost is $10.00 each.

## (#5) The ELSEIF Clause: checking for one of several conditions

Inserting one or more ELSEIF clauses into a block IF statement allows the statement to check for one of several conditions. The following example shows how an appropriate message could be displayed depending on how many points a player earned on a video game:

```
IF playerScore > 50000 THEN
     PRINT “Congratulations! You earned the rank of Intergalactic
            Warrior.”
ELSEIF playerScore > 35000 THEN
     PRINT “You earned the rank of Star Fleet Commander.”
ELSEIF playerScore > 20000 THEN
     PRINT “You earned the rank of Space Ship Captain.”
ELSE
     PRINT “You earned the rank of Space Cadet.”
END IF
```

It is important to realize that when this program segment is executed, something will always happen. If none of the specified conditions is `TRUE`, the `ELSE` clause will be executed.

Once again, let’s alter the sale prices for the music store CDs:

| | |
|---|---|
| 1 to 5 CDs | $10.00 |
| 6 to 9 CDS | $ 9.75 |
| 10 or more CDs | $ 9.50 |

The following block `IF` statement is an effective solution to this type of problem:

```
IF number >= 10 THEN
     cost = number * 9.5
ELSEIF number >= 6 THEN
     cost = number * 9.75
ELSE
     cost = number * 10
END IF
```

If number is greater than or equal to `10`, the statement following the `THEN` is executed, and if it is less than `10` but greater than or equal to `6`, the statement in the `ELSEIF` clause is executed; otherwise, the statement in the `ELSE` clause is executed.

## (#6) Nesting Statements: checking several unrelated conditions

It is possible to nest block `IF` statements by placing them inside one another. This allows the programmer to check several unrelated conditions. Let’s study the following example:

```
IF age >= 18 THEN
     IF sex = “F” THEN
          PRINT “Woman”
     ELSE
          PRINT “Man”
     END IF
ELSE
     IF sex = “F” THEN
          PRINT “Girl”
     ELSE
          PRINT “Boy”
     END IF
END IF
```

A number of actions are taken when these nested IF statements are executed. The first IF determines whether age is greater than or equal to 18. If this condition is TRUE, the first inner IF is executed and determines whether sex equals “F”. If both conditions are TRUE, “Woman” is displayed. If only the first one is TRUE, “Man” is displayed. If the outer IF is FALSE, we know that the person is under 18 and execution continues to the outer IF’s ELSE clause. The IF statement nested in the outer IF’s ELSE clause checks to see if sex equals “F”; if TRUE, “Girl” is output; otherwise “Boy” is output. Nesting IF statements in this manner allows the program to check for several unrelated conditions, in this case age and sex.

**Notice** in the preceding nested IF statement that the two inner IFs are indented inside the outer IF. This indentation makes the logic easier to follow. Care must be taken when nesting decision structures. And remember, each IF must have its own END IF.

## The SELECT CASE Statement: selecting from a list of alternatives

The SELECT CASE statement allows an action to be selected from a list of alternatives For example:

```
INPUT “Enter your class (1 – 4)”; class

SELECT CASE class
      CASE 1
            PRINT “Freshman”
      CASE 2
            PRINT “Sophomore”
      CASE 3
            PRINT “Junior”
      CASE 4
            PRINT “Senior”
      CASE ELSE
            PRINT “Invalid class number.”
END SELECT
```

Like the block IF, the SELECT CASE is a block structure and is often used in the place of an ELSEIF code block. It begins with the keywords SELECT CASE and ends with END SELECT. Each CASE clause includes a block of one or more statements that are to be executed if the stated variable equals the listed value. In this example, if class equals 1, the statement following CASE 1 is executed; if class equals 2, the statement following CASE 2 is executed; and so forth. If the value of class is invalid (that is, outside the 1-4 range), the statement in the CASE ELSE clause is executed. The CASE ELSE clause is optional, but is quite useful for checking for invalid input.

SELECT CASE may be used with character string data as well as numeric data, as shown in the following example:

```
SELECT CASE language

      CASE "Spanish"
            PRINT "Buenos dias."
      CASE "English"
            PRINT "Good day."
      CASE "French"
            PRINT "Bonjour."
      CASE "German"
            PRINT "Guten Tag."
      CASE ELSE
            PRINT "Invalid entry."
END SELECT
```

In QBasic several expressions can be listed in a single CASE clause. The next code snippet illustrates this option. The snippet will determine the number of days in a particular month. Notice what happens when the user enters February… the user must then indicate whether this is a leap year; a double-alternative decision IF statement inside the "February" CASE clause statement can then assign the correct number of days for February.

```
SELECT CASE month
      CASE "April", "June", "September", "November"
                  days = 30
      CASE "January", "March", "May", "July", "August",
            "October", "December"
                  days = 31
      CASE "February"
            INPUT "Is this a leap year (Y/N): ", leapYear
            IF leapYear = "Y" THEN
                  days = 29
            ELSE
                  days = 28
            END IF
      CASE ELSE
                  days = 0
END SELECT
```

## Menus: allowing the user to choose a desired function from a list

A menu is a list of options that a program can perform. Just as a customer in a restaurant looks at the menu to choose a meal, a program user can look at a menu displayed on the screen to choose a desired operation. The user makes the selection by entering a code (usually a simple number or letter) at the keyboard, as in the following example:

```
Please enter one of the following numbers:
      1 - Convert to Japanese Yen
      2 - Convert to Egyptian Pounds
      3 - Convert to Mexican Pesos
      4 - Convert to German Marks
```

The SELECT CASE statement is often used in conjunction with menus, such as the one shown above. After entering a number of dollars to be converted, the user enters a 1, 2, 3, or 4 to indicate the type of currency conversion desired. The SELECT CASE determines which calculation should be performed. If the user enters an invalid code, the CASE ELSE clause should display an error message.

## Logical Operators: combining expressions to produce a single value

In addition to arithmetic operators ( +, –, *, /, ^ ) and relational operators ( =, <>, >, >=, <, <= ), there is a third group of operators called Boolean (or logical) operators. A Boolean operator acts on one or more expression(s) that evaluate as TRUE or FALSE to produce a statement with a TRUE or FALSE value. The three most commonly used logical operators are AND, OR, and NOT.

The AND operator combines two expressions and returns a value of TRUE only when both of these conditions evaluate to TRUE. For example, the combined logical expression:

```
IF (score > 75) AND (time < 50) THEN
     PRINT name
END IF
```

evaluates as TRUE only if the expression score > 75 and the expression time < 50 are both TRUE. If one expression or the other is FALSE, the entire statement is FALSE causing the THEN clause of the statement to be ignored. The parentheses in the preceding statement are not necessary, but they improve the readability of the statement.

The logical OR operator also combines two expressions, but only one of the expression needs to evaluate as TRUE for the entire statement to be TRUE. Thus, the statement

```
IF (score > 75) OR (time < 50) THEN
     PRINT name
END IF
```

evaluates as TRUE if either the expression score > 75 or the expression time < 50 is TRUE, or if both are TRUE. The entire condition is FALSE only if the expression score > 75 and the expression time < 50 are both FALSE.

The third logical operator, NOT, is a **unary** operator (an operator used with only one operant) and is used with a single expression. The effect of NOT  is to reverse (negate) the logical value of the expression it precedes. For example, if the variable pet contains the value "Dog", the condition of the following statement is FALSE:

```
IF NOT (pet = "Dog") THEN
     felines = felines + 1
END IF
```

Because the condition pet = "Dog" evaluates as TRUE, the NOT operator reverses this value to FALSE, making the final result of the entire condition FALSE. If pet contained any other value, the condition pet = "Dog" would evaluate as FALSE, and the NOT operator would make the value of the entire condition TRUE.

## Truth tables:

| AND | | |
|---|---|---|
| **Expression #1** | **Expression # 2** | **Result** |
| TRUE | TRUE | **TRUE** |
| TRUE | FALSE | **FALSE** |
| FALSE | TRUE | **FALSE** |
| FALSE | FALSE | **FALSE** |

| OR | | |
|---|---|---|
| **Expression #1** | **Expression # 2** | **Result** |
| TRUE | TRUE | **TRUE** |
| TRUE | FALSE | **TRUE** |
| FALSE | TRUE | **TRUE** |
| FALSE | FALSE | **FALSE** |

When a single statement contains more than one logical operator, the operations are evaluated in the following sequence:

**NOT**<br>
**AND**<br>
**OR**

**Hierarchy of Operations:**

1. Anything in parentheses
2. Exponentiation (**^**)
3. Unary plus or minus sign (a sign used alone in front of a number)
4. Multiplication and division (***, /**)
5. Addition and subtraction (**+, −**)
6. Relational/Comparison operators (**=, <>, <, >, <=, >=**)
7. **NOT**
8. **AND**
9. **OR**

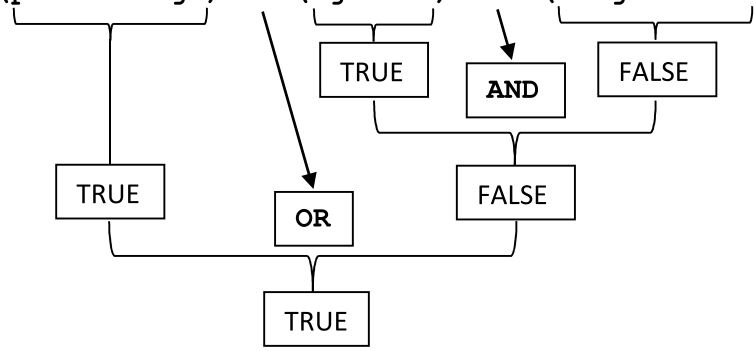
## Combining Multiple Logical Operators

For example, the following expression combines AND and OR:

```
IF (pet = “Dog”) OR (age = 3) AND (weight = 10) THEN
```

Given the predefined order of evaluation, the following shows how the preceding statement would be evaluated given pet = “Dog”, age = 3, and weight = 9:

```
IF (pet = “Dog”) OR (age = 3) AND (weight = 10) THEN
```



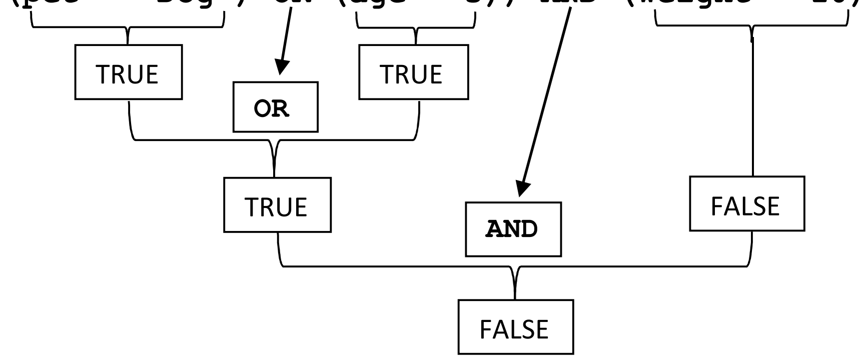
The AND portion of the expression is evaluated first. That result is then combined with the OR portion of the statement to determine the final value of the entire condition. In this case, the statement condition is TRUE, so the THEN clause would be executed.

The precedence of logical operators (like that of arithmetic operators) can be altered by using parentheses. In the previous example, using the same variable values as before, could be rewritten as:

```
IF ((pet = "Dog") OR (age = 3)) AND (weight = 10) THEN
```

In this example, the OR portion of the expression is evaluated before the AND portion. Thus, the parentheses can change the final result of the evaluation, as shown in the following diagram. Compare the evaluation of this statement with the previous diagram.

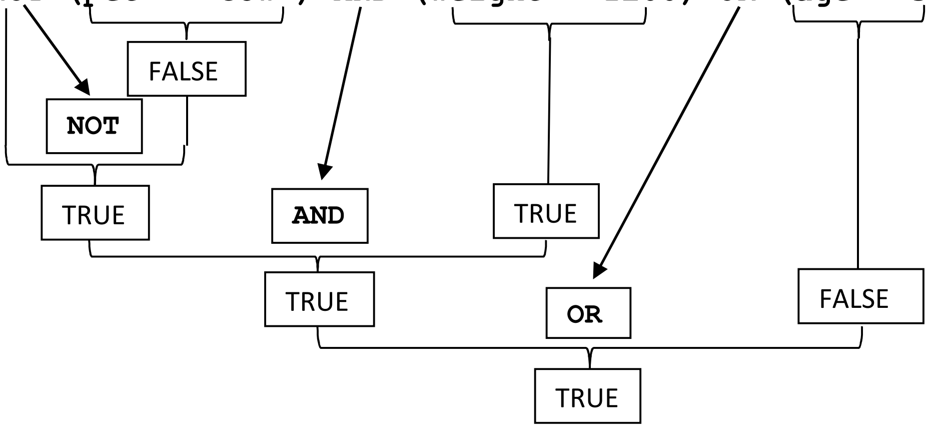
```
IF ((pet = "Dog") OR (age = 3)) AND (weight = 10) THEN
```



Even if the desired order of evaluation is the same as the predefined order, it is good programming practice to use parentheses in order to make the logic clear.

NOT can also be combined with AND and OR in a single statement, as shown in the following diagram. Study the evaluation of the condition, making sure that you understand how the use of parentheses and the predefined order of operators have determined the final result of the evaluation. Assume that pet ="Pig", age = 6, and weight = 1500.

```
IF NOT (pet = "Cow") AND (weight > 1200) OR (age < 3) THEN
```



The code snippet below demonstrates how logical operators can be used to determine if a triangle is scalene, isosceles, or equilateral.

- scalene has no equal sides
- isosceles has two equal sides
- equilateral all three sides are equal

Notice that the first test uses the AND operator to determine if all three sides are equal.

```
IF (side1 = side2) AND (side2 = side3)
```

The test for an isosceles triangle is more complex and involves checking for three different conditions. Only one of these conditions needs to be TRUE for the triangle to be isosceles; therefore, this test involves the OR operators. If none of these conditions is TRUE, the triangle must be scalene. As shown in this snippet, logical operators allow for a variety of conditions to be checked efficiently and simultaneously.

```
IF (side1 = side2) AND (side2 = side3) THEN
     PRINT “equilateral”
ELSEIF (side1 = side2) OR (side2 = side3) OR (side1 = side3) THEN
     PRINT “isosceles”
ELSE
     PRINT “scalene”
END IF
```

| **Examples of Conditions Using Logical Operators** | |
|---|---|
| NOT (1 * 4 = 5) | **True** |
| (18 < 16) OR (7 + 2 = 9) | **True** |
| (18 < 16) AND (7 + 2 = 9) | **False** |
| ((2 + 8 ) <= 11) AND (17 * 2 = 34) | **True** |
| NOT (12 > 8 - 2) | **False** |