16 Programming Concepts

Objectives

After completing this chapter, you will be able to

- understand the use of variables and the various ways to change their values
- trace the logical flow of a program using pseudo code
- identify the values of variables during execution of a program flowchart or a program segment
- recognise the basic constructs of a computer program, including branching, conditional and iteration statements

In this chapter, you will learn basic programming concepts, such as arithmetic operations and logic flow of a program. In order to avoid dealing with the intricacies of a real programming language, you will learn programming concepts through a **pseudocode**, which is a condensed form of English.

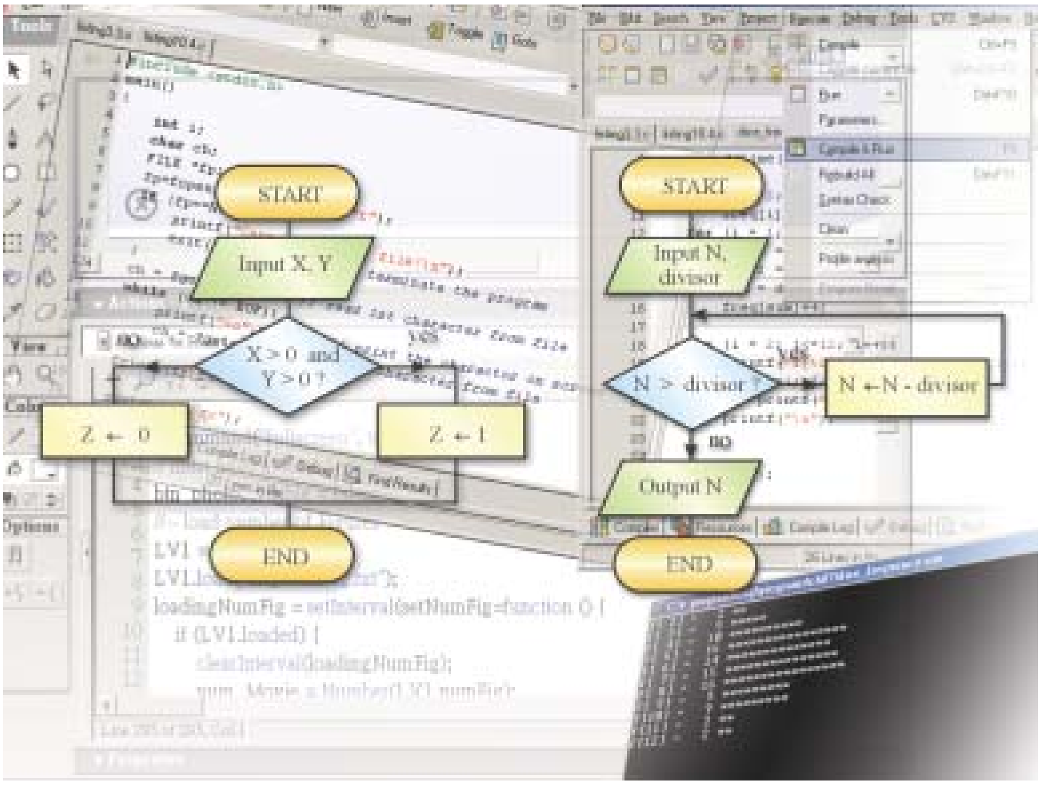


Fig.1 Programming concept

16.1 Variables and Expressions

A. Variables

Keypoints

A variable is a memory location that holds data. An assignment statement assigns value to a variable, overwriting its contents.

When a program is running, the contents of certain memory locations are changed. These memory locations are identified by variable names. A **variable** may hold data like, a number, a constant, a string of characters etc.

An **assignment statement** assigns a value to a variable. It would overwrite the previous content of the variable. In this book, we use an arrow symbol "←" to assign a value to a variable on its left side. The corresponding symbol is ":=" in PASCAL and "=" in other languages, like C and BASIC.

Keypoints

Common data types are
1. numeric
2. character

The format of an assignment statement is

```
variable ← {value or expression}
```

Table 1 shows some examples of assignment statements:

	Assignment statement	Description
1.	`X ← 5`	An integer `5` is placed in variable `X`.
2.	`Y ← 2.5`	A real number `2.5` is placed in variable `Y`.
3.	`name ← "Peter"`	A string `"Peter"` is placed in variable `name`.

Table 1 *Examples of assignment statements*

Common data includes two types: *numeric* and *character*. Numeric data, which can be an integer or a real number, are used for calculation. Examples are `2`, `-3`, `5.2`, `0.0006`. Character data, which are placed inside quotes, are used to show messages or other contents. Anything that can be typed using the keyboard can be a character. Examples of character data are "`A`", "`2.3`", "`Peter`".

2. Arithmetic Operations

Keypoints

Computers perform multiplication and division before addition and subtraction.

Arithmetic operations include addition (+), subtraction (-), multiplication (*) and division (/). Consider the following examples:

	Statements	Results
1.	X ← 5 + 8	X stores 13
2.	X ← 5 - 7	X stores -2
3.	X ← 3 * 2	X stores 6
4.	X ← 8 / 2	X stores 4
5.	X ← 5 / 2	X stores 2.5

Table 2 *Examples of arithmetical expressions*

The calculation on the right hand side of the arrow is called an **expression**. The computer evaluates the expression and places the result to the variable on the left side.

C. Order of Precedence

A computer always follows the proper *order of precedence* as we do mathematics. It means that multiplication and division are performed before addition and subtraction. Consider the following examples:

	Statements	Results
1.	X ← 4 * 2 + 3	Since 4 * 2 is equal to 8, the statement becomes X ← 8 + 3. Hence, X stores 11.
2.	X ← 4 + 2 * 3	Multiplication is performed before addition. Since 2 * 3 is equal to 6, the statement becomes X ← 4 + 6. Hence, X stores 10.
3.	X ← 4 * 2 / 8	Since 4 * 2 is equal to 8, the statement becomes X ← 8 / 8. Hence, X stores 1.
4.	X ← 4 + 8 / 4	Division is performed before addition. Since 8 / 4 is equal to 2, the statement becomes X ← 4 + 2. Hence, X stores 6.
5.	X ← 3 * 7 - 2 / 5	Multiplication and division are performed before subtraction. The statement becomes X ← 21 - 0.4. Hence, X stores 20.6.

Table 3 *Order of precedence*

Operations inside a pair of parentheses are performed first. Consider the following examples:

	Statements	Results
1.	X ← 7 - (2 + 3)	The statement becomes X ← 7 - 5. Hence, X stores 2.
2.	X ← (-2 * 5) + 6	The statement becomes X ← -10 + 6. Hence, X stores -4.
3.	X ← 3 * (-2 - 3)	The statement becomes X ← 3 * (-5). Hence, X stores -15.

Table 4 *Expressions inside parenthesis are performed first*

D. More Complex expressions

Sometimes, an expression may include one or more variables. Suppose X stores 5 and Y stores -3 before execution.

	Statements	Results
1.	X ← X + 2	The statement becomes X ← 5 + 2. Hence, X stores 7.
2.	X ← Y * 2	The statement becomes X ← (-3) * 2. Hence, X stores -6.
3.	Y ← X * Y + Y	The statement becomes Y ← 5 * (-3) + (-2). Hence, Y stores -17.

4.	X ← Y * 2
	Y ← X + Y
		The first statement becomes X ← (-3) * 2, i.e. X stores -6. The second statement becomes Y ← -6 + (-3). Hence, Y stores -9 and X stores -6.

5.	X ← X + Y
	Y ← X + Y
		The first statement becomes X ← 5 + (-3), i.e. X stores 2. The second statement becomes Y ← 2 + (-3). Hence, Y stores -1 and X stores 2.

Table 5 *Examples of assignment statements with more than one variable*

E. String concatenation

String concatenation means to join two strings of characters together. The operator is “+”.

	Statements	Results
1.	A ← "2" + "3"	A stores "23".
2.	A ← "Hi" + "!"	A stores "Hi!".
3.	A ← "Pet" + "er Pan"	A stores "Peter Pan".

Table 6 *String concatenation*

Suppose the variable A stored "3", B stored "27" and C stores "Peter".

	Statements	**Results**
1.	X ← A + "2"	The statement becomes X ← "3" + "2". Hence, X stores "32". Note: Contents of A remains unchanged
2.	X ← "2" + A	The statement becomes X ← "2" + "3". Hence, X stores "23".
3.	X ← A + A + A	The statement becomes X ← "3" + "3" + "3". Hence, X stores "333".
4.	X ← A + B	The statement becomes X ← "3" + "27". Hence X stores "327".
5.	C ← C + B	The statement becomes C ← "Peter" + "27". Hence C stores "Peter27".

Table 7 *More examples on string concatenation*

Keypoints

Operation between a string and a number is meaningless.

The following statement is *not* allowed:

```
X ← "A" + 2
```

because the operation between a string and a number is meaningless.

16.2 Input and Output Statements

Keypoints

An input statement allows user to change the value of a variable interactively.

An output statement puts the value on the screen.

A. Input Statement

An **input statement** obtains input from user and store the data in a variable. For example,

```
INPUT X
```

is an input statement. When the user types 2.5 on the keyboard, this value will be stored in X.

The format of the input statement is

```
INPUT {variable}
```

B. Output Statement

An **output statement** would put the values on the screen. For example,

```
OUTPUT 2.5 * 2
```

will put the product of 2.5 and 2, i.e. 5 on the screen.

The format of the output statement is

```
OUTPUT {expression}
```

In the following examples, assume X stores 3.

	Statements	Results
1.	`OUTPUT X`	The output is 3
2.	`OUTPUT 2 * 4 + 5`	The output is 13.
3.	`OUTPUT 2 * X + 4`	The computer will evaluate 2 * 3 + 4. The output is 10.

Table 8 *Examples of output statements*

Example 1 The following program segment will find the area of a triangle.

```
10   INPUT HEIGHT
20   INPUT BASE
30   AREA ← HEIGHT * BASE / 2
40   OUTPUT AREA
```

$$\text{area} = \frac{1}{2} \times \text{height} \times \text{base}$$

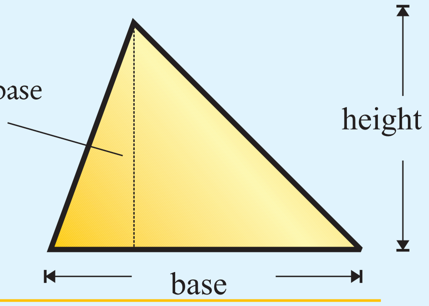


Fig.2 *Calculating the area of a triangle*

If the user inputs 5 and 4 in response to lines 10 and 20 respectively, the output would be 10.

Example 2 The following program segment will interchange the contents of two input data

```
10   INPUT X
20   INPUT Y
30   Z ← X
40   X ← Y      } Swap X and Y
50   Y ← Z
60   OUTPUT X
70   OUTPUT Y
```

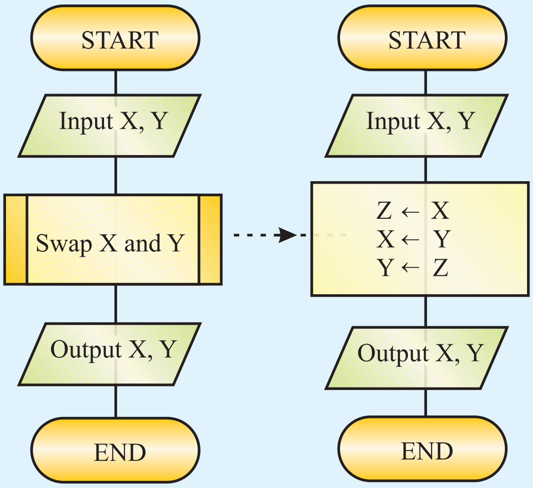


Fig.3 *Swapping two variables*

Lines 30 to 50 are to interchange (or *swap*) the contents of X and Y. Suppose the inputs are 3, 6. After line 30, a copy of X is stored in Z. So, Z stores 3. After line 40, the value of Y is copied to X. So, X stores 6. After line 50, the value of Z is copied to Y. So, Y stores 3. See the illustration below. The outputs of the above program will be 6, 3.

	X	Y	Z
Initial values	3	6	
30 Z ← X	3	6	3
40 X ← Y	6	6	3
50 Y ← Z	6	3	3

Fig.4 *Swapping two variables X and Y*

16.3 Conditional Statements

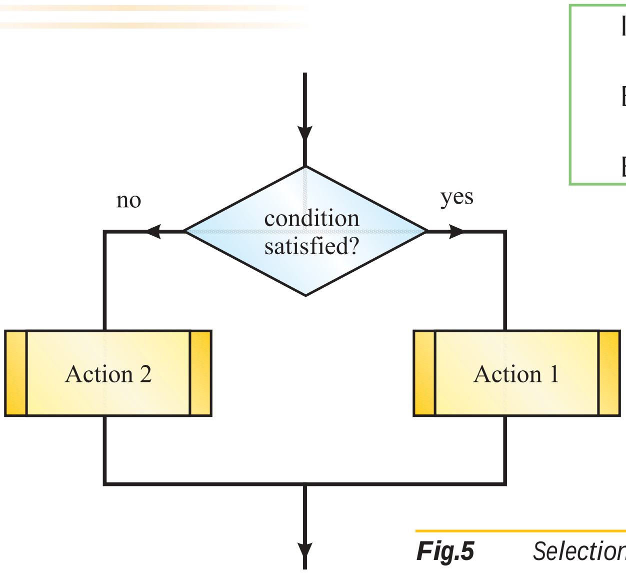
Keypoints

A conditional statement can be IF...THEN...ELSE... ENDIF or IF...THEN...ENDIF

A **conditional statement** will carry out an action if a specific condition is satisfied. It enables a program to select from one or more alternatives. The flowchart in Fig.5 is called a **selection control structure**.

The format of a conditional statement can be

```
IF {conditional expression} THEN
   {action 1}
ELSE
   {action 2}
END IF
```

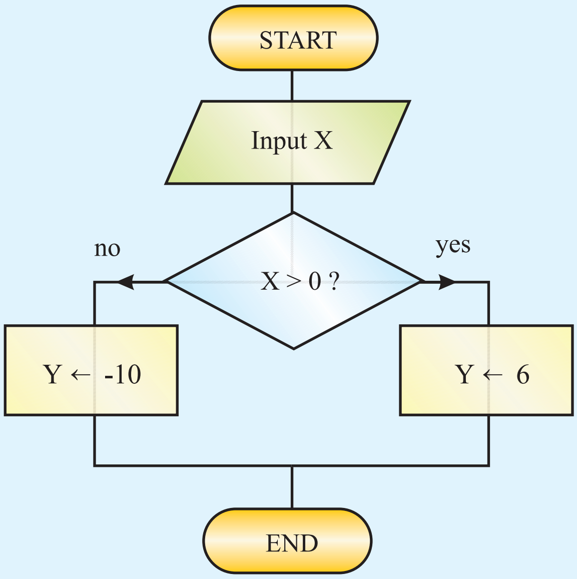


The result of the conditional expression is either *true* or *false*. If the result is true then "*Action 1*" will take place. Otherwise, "*Action 2*" will take place. The action may consist of more than one statement.

Fig.5 *Selection control structure with two alternatives*

Example 3 The following program segment would assign a different value to Y depending on the input value.

```
10  INPUT X
20  IF X > 0 THEN
30        Y ← 6
40  ELSE
50        Y ← -10
60  END IF
```



If the user enters a positive number, such as 5 or 12, in response to the input statement, the value of Y would be 6. If the user enters 0 or a negative number, such as -5 or -12, the value of Y would be -10.

Fig.6 *Selection based on the sign of X*

A. Relational Operators

The symbol “>” in the conditional expression of the above example means “greater than”. It is called a **relational operator**. Other relational operators are “=”, “>=”, “<=”, “<“, “<>”

Relational operator	Meaning
=	equal to
<>	not equal to
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

Table 9 *Relational operators*

Example 4 The following program segment has the same effect as Example 3.

```
10  INPUT X
20  IF X <= 0 THEN
30          Y ← -10
40  ELSE
50          Y ← 6
60  END IF
```

When the input is less than or equal to zero, the value of Y would be -10. When the input is greater than 0, the value of Y would be 6.

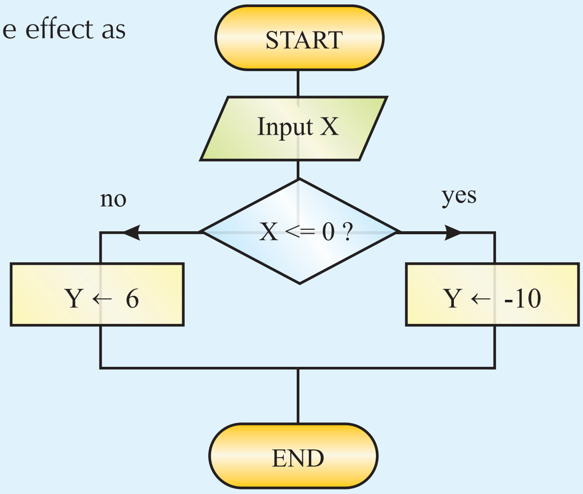


Fig.7 *Selection based on the sign of X*

Example 5 The following program segment will put the largest input into Z.

```
10  INPUT X
20  INPUT Y
30  IF X > Y THEN
40          Z ← X
50  ELSE
60          Z ← Y
70  END IF
```

When the user inputs 5 and 8, the condition in line 30 is not satisfied. Therefore, the action in line 60 will take place, i.e. Z ← Y. Thus, Z would store 8 which is the largest of the inputs.

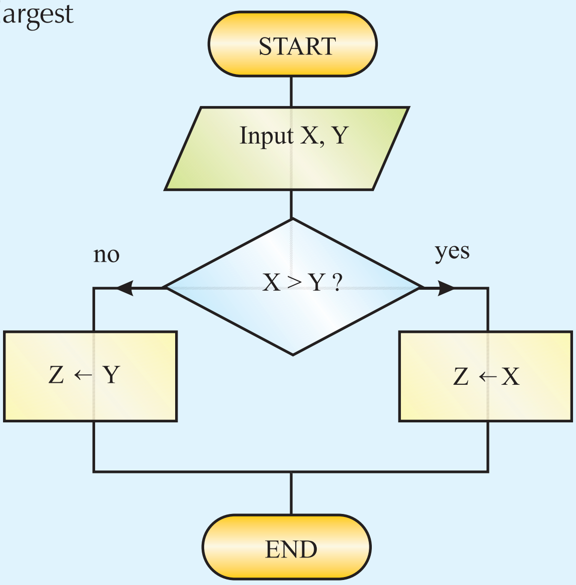


Fig.8 *Selection based on the relative values of X and Y*

B. Conditional statements with One Alternative

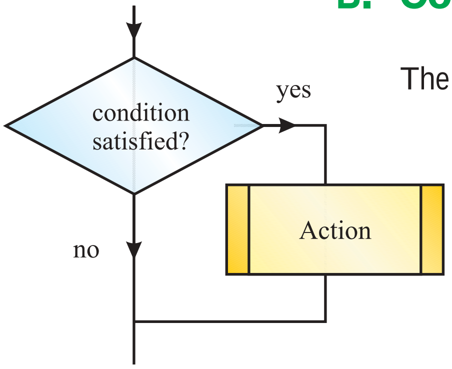


Fig.9 *Selection control structure with one alternative*

The format of a conditional statement can also be

```
IF {conditional expression} THEN
   {action 1}
END IF
```

If the result of the conditional expression is true, then action 1 will take place. Otherwise, no action will take place.

Example 6 The following program segment will output the absolute value of the input data, i.e. the magnitude of the input.

```
10  INPUT X
20  IF X < 0 THEN
30      X ← -X
40  END IF
50  OUTPUT X
```

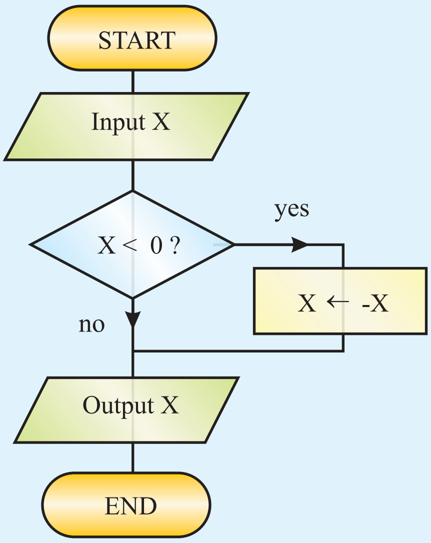


Fig.10 *Finding absolute value*

The "-" in line 30 is called an **unary operator** and changes the sign of the variable X. It is equivalent to

```
30  X ← -1 * X
```

When the input is less than zero, e.g. -5, it would be multiplied by -1, giving 5.

Example 7 The following program segment will input two data and arrange them in ascending order.

```
10  INPUT X
20  INPUT Y
30  IF X > Y THEN
40      Z ← X
50      X ← Y
60      Y ← Z
70  END IF
80  OUTPUT X
90  OUTPUT Y
```

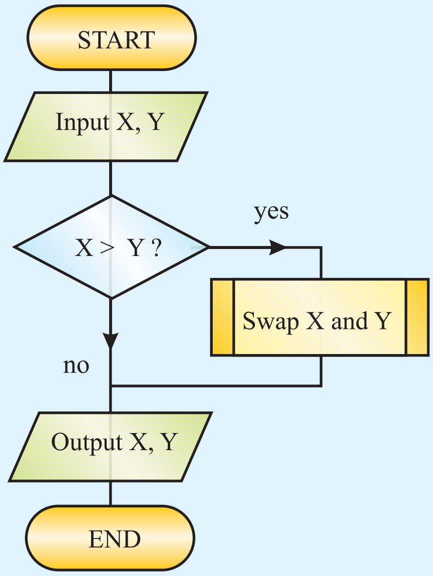


Fig.11 *Sorting in ascending order for 2 numbers*

In line 30, the computer compares the contents of X and Y. If X is greater than Y, the actions in lines 40 to 60 will take place, i.e. the values of X and Y will be interchanged, giving the result X smaller than Y.

C. Boolean Operators

As mentioned above, a conditional expression gives either *true* or *false*. We say that the result is a **Boolean value**.

Sometimes, the result depends on more than one condition. For example, you will buy something if you want that thing and you have enough money. Both conditions must be satisfied. In this example, two Boolean values interact to give a new Boolean value. The "AND" is called a **Boolean operator**. Other Boolean operators are "OR" and "NOT".

Example 8 The following program segment will assign 1 to Z when both inputs are positive. Otherwise, 0 will be assigned to Z.

```
10   INPUT X
20   INPUT Y
30   IF X > 0 AND Y > 0 THEN
40        Z ← 1
50   ELSE
60        Z ← 0
70   END IF
```

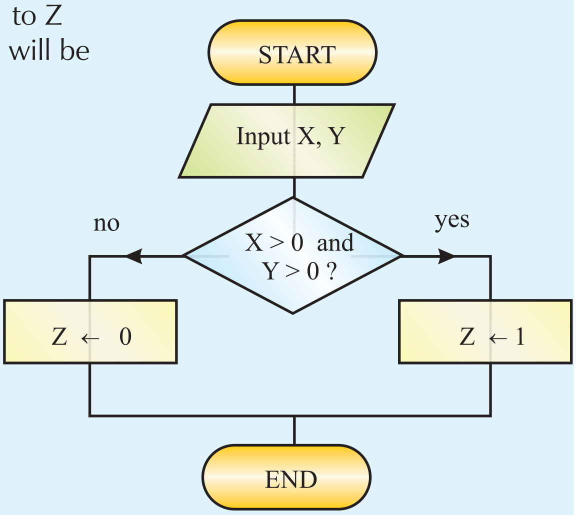


Fig.12 *Testing positive for both numbers*

The result of the **AND** is governed by the following truth table. Suppose p and q represent two Boolean values.

	p	q	p AND q
1.	false	false	false
2.	false	true	false
3.	true	false	false
4.	true	true	true

Table 10 *Truth table for AND operator.*

In example 8, suppose the input data are 5 and -4. The computer will determine the result of each relational operation and then use line 3 of Table 9 to determine the final result. Since the table returns a false value, Z will be assigned with 0.

Table 11 Truth table for OR operator.

	p	q	p OR q
1.	false	false	false
2.	false	true	true
3.	true	false	true
4.	true	true	true

Example 9 The following program segment will assign 1 to Z when either input is positive. Otherwise, 0 will be assigned to Z.

```
10  INPUT X
20  INPUT Y
30  IF X > 0 OR Y > 0 THEN
40       Z ← 1
50  ELSE
60       Z ← 0
70  END IF
```

If the user enters 5 and -4, the first condition is true but the second condition is false. The computer will use line 3 of Table 10 to determine the final result. Since the table returns a true value, Z will be assigned with 1.

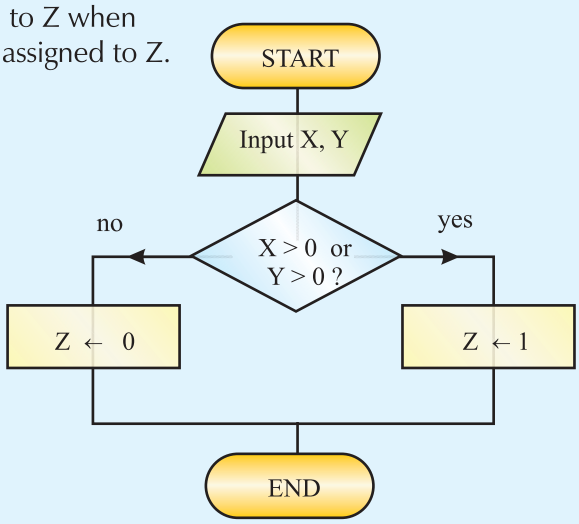


Fig.13 Testing for at least one positive

Table 12 Truth table for NOT operator.

	p	NOT p
1.	false	true
2.	true	false

Example 10 The following program segment will assign 1 to Z when the inputs are not equal. Otherwise, 0 will be assigned to Z.

```
10  INPUT X
20  INPUT Y
30  IF NOT (X = Y) THEN
40       Z ← 1
50  ELSE
60       Z ← 0
70  END IF
```

Line 30 is equivalent to

```
30  IF X <> Y THEN
```

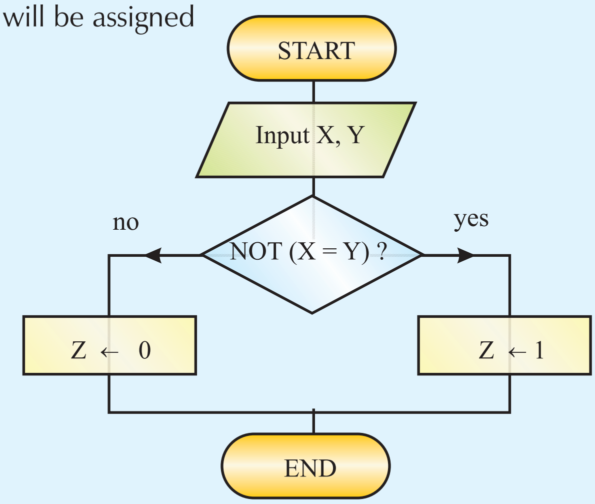


Fig.14 Testing for equality

16.4 Branching Statement

Keypoints

A branching statement changes the sequence of execution.

A **branching statement** will change the sequence of execution. It is implemented by a `GOTO` command. The format of the branching statement is

```
GOTO {line number}
```

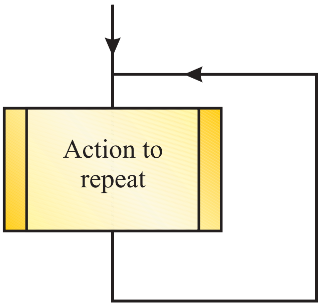
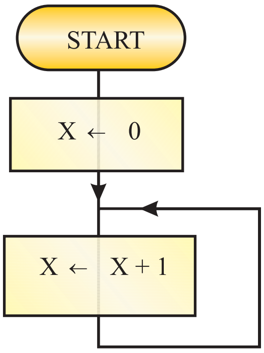


Fig.15 *Branching (infinite loop)*

Consider the following example:

```
10   X ← 0
20   X ← X + 1
30   GOTO 20
```

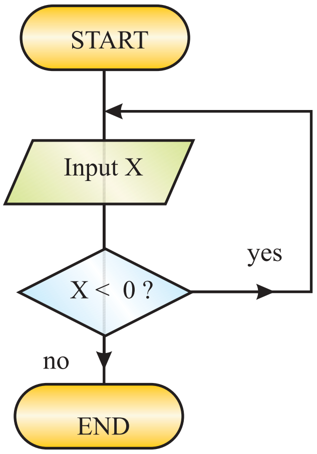


This program will never stop. The computer will keep on add 1 to X so that X takes on values 0, 1, 2, ... successively. We say that the program falls into an **infinite loop**. There will be no response from the computer until the program is manually stopped.

Fig.16 *An example of infinite loop*

A branching statement should be used with a conditional statement. Consider the following example:

```
10   INPUT X
20   IF X < 0 THEN
30        GOTO 10
40   END IF
50   OUTPUT X
```



In this program, if the computer finds that the input data is negative (X < 0), it would request for input again. This ensures that a number greater than or equal to zero is entered from the keyboard.

Fig.17 *Conditional branching*

Example 11 The following program segment will increase the values of X for a number of times.

```
10  X ← 0
20  X ← X + 1
30  IF X < 3 THEN
40      GOTO 20
50  END IF
60  OUTPUT X
```

After the 1st encounter of line 20, X stores 1. Since the condition in line 30 is satisfied, the computer will branch to line 20 again. This will repeat until X is equal to 3.

Thus, output from the program is 3.

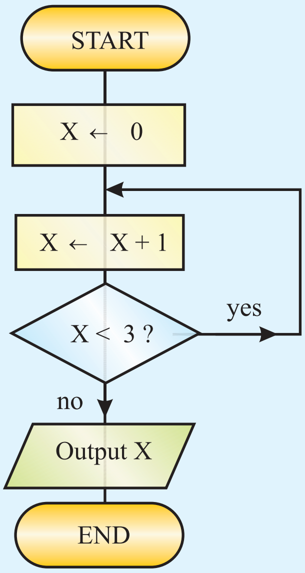


Fig.18 *Iteration for three times*

After executing line 30	Result of X	Condition in line 30	Action
1st time	1	satisfied (1 < 3)	`GOTO 20`
2nd time	2	satisfied (2 < 3)	`GOTO 20`
3rd time	3	not satisfied (3 = 3)	`OUTPUT X`

Since line 20 is executed for three times, we say that there are three iterations. An **iteration** means repeating a sequence of instructions once. We shall learn some constructs designed to carry out iterations for a specified number of times.

Example 12 Consider the following program segment.

```
10  S ← 0
20  X ← 0
30  X ← X + 1
40  S ← S + X
50  IF X < 10 THEN
60        GOTO 30
70  END IF
80  OUTPUT S
```

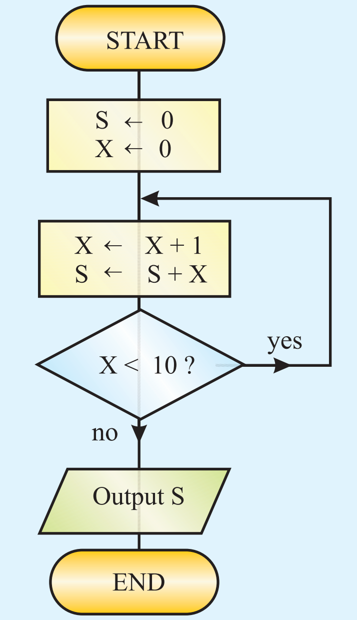


Fig.19 *Iteration for ten times*

In this program, lines 30 and 40 will be repeated as long as X is less than 10. Line 30 is to increase X by 1. Line 40 is to find the sum of X. Since X changes from 1 to 10, this program is to find the sum of 1 + 2 + ... + 10. The output is expected to be 55.

After line 50	Result of X	Result of S	Condition in line 30	Action
1st time	1	1	satisfied (1 < 10)	`GOTO 30`
2nd time	2	3	satisfied (2 < 10)	`GOTO 30`
3rd time	3	6	satisfied (3 < 10)	`GOTO 30`
4th time	4	10	satisfied (4 < 10)	`GOTO 30`
5th time	5	15	satisfied (5 < 10)	`GOTO 30`
6th time	6	21	satisfied (6 < 10)	`GOTO 30`
7th time	7	28	satisfied (7 < 10)	`GOTO 30`
8th time	8	36	satisfied (8 < 10)	`GOTO 30`
9th time	9	45	satisfied (9 < 10)	`GOTO 30`
10th time	10	55	not satisfied (10 = 10)	`OUTPUT S`

16.5 The For-loop

Keypoints

A for loop carries out iterations for a number of times.

Examples 11 and 12 demonstrate iterations for a number of times. This can be replaced by a simpler method: **for-loop**. The format of the for-loop is

```
FOR {variable} = {initial value} TO {final value} STEP {step}
    {actions to be repeated}
NEXT
```

If the keyword STEP is omitted, the step is 1.

```
FOR {variable} = {initial value} TO {final value}
   {actions to be repeated}
NEXT
```

Consider the following example:

```
10  FOR X = 1 to 10
20       OUTPUT X
30  NEXT
```

As X changes from 1 to 10, the program will output 1, 2, 3, ... 10.

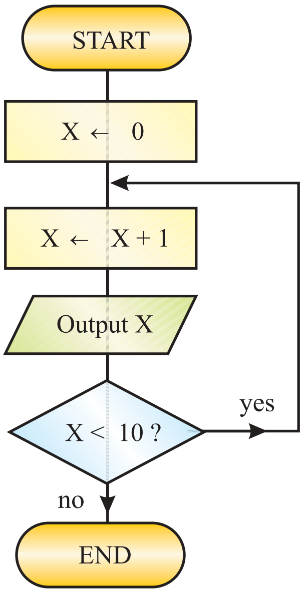


Fig.20 *A For-loop iterating for ten times*

Example 13 The following program is designed to find the sum of $1^2 + 2^2 + 3^2 .... + 10^2$.

```
10  S ← 0
20  FOR X = 1 TO 10
30        S ← S + X * X
40  NEXT
50  OUTPUT S
```

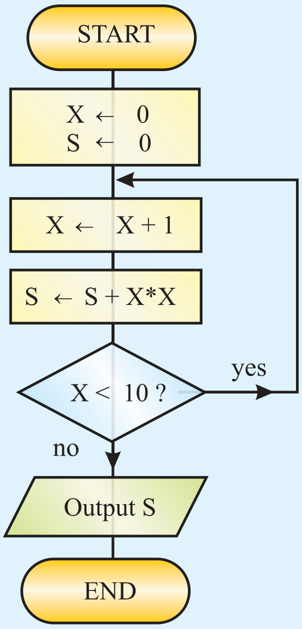


Fig.21 *Finding sum of square of integers from 1 to 10*

Example 14 The following program is designed to find the sum of

$2^1 + 2^2 + 2^3 .... + 2^{10}$.

```
10   S ← 0
20   P ← 1
30   FOR X = 1 TO 10
40        P ← P * 2
50        S ← S + P
60   NEXT
70   OUTPUT S
```

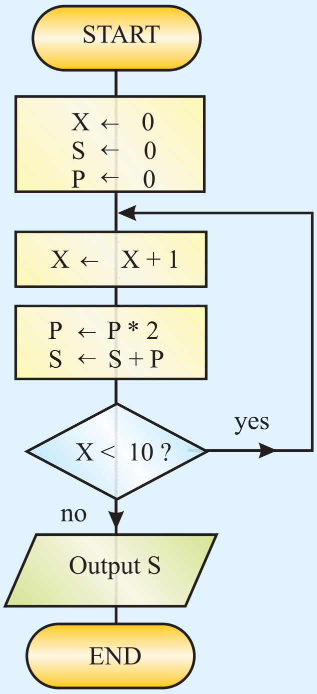


Fig.22 *Finding sum of 2 to the power from 1 to 10*

Example 15 The following program will output the square of the even numbers between 2 and 10 inclusively. i.e. 2^2, 4^2, 6^2, 8^2, 10^2.

```
10   FOR X = 2 TO 10 STEP 2
20        OUTPUT X * X
30   NEXT
```

This is equivalent to

```
10   X ← 0
20   X ← X + 2
30   OUTPUT X * X
40   IF X < 10 THEN
50        GOTO 20
30   END IF
```

The outputs from the program are 4, 16, 36, 64, 100.

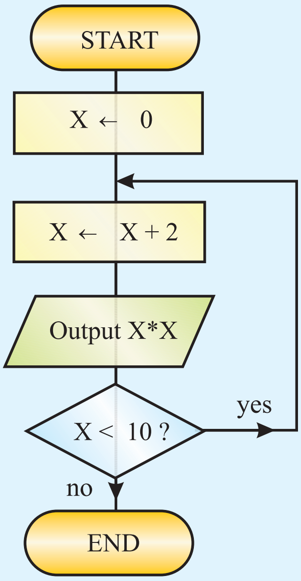


Fig.23 *The squares of even numbers from 2 to 10*

Example 16 The following program will input 10 positive numbers and determine the largest one.

```
10  MAX ← -1
20  FOR I = 1 TO 10
30        INPUT X
40        IF X > MAX THEN
50              MAX ← X
60        ENDIF
70  NEXT
80  OUTPUT MAX
```

Line 10 assigns -1 to the variable MAX. Line 40 compares the input data with MAX. If the input data is larger, then it would replace the contents of MAX.

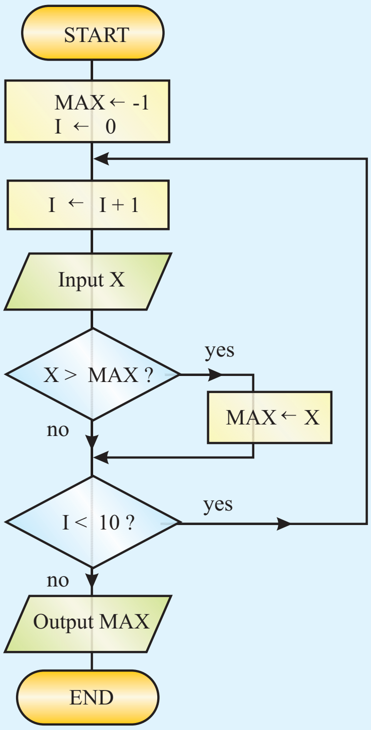


Fig.24 *Finding maximum*

16.6 The Repeat ... Until-loop

Keypoints

A Repeat...until loop will take place at least once.

The format of the `Repeat .. until` loop is

```
REPEAT
   {actions to repeat}
UNTIL {condition is satisfied}
```

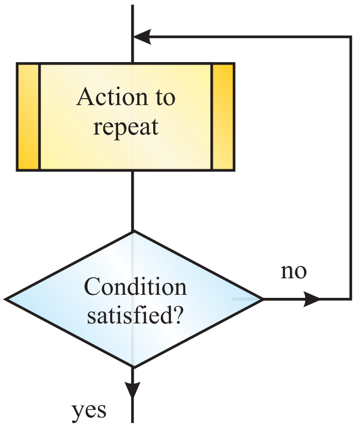


Fig.25 *Repeat...until loop*

The actions inside the loop will take place at least once. Consider the following example:

```
10  REPEAT
20        INPUT X
30  UNTIL X > 0
```

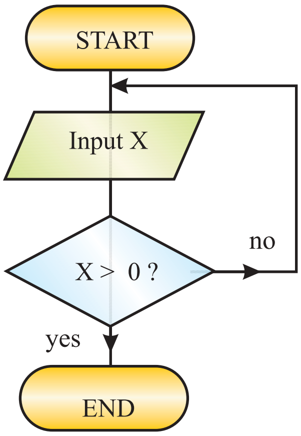


Fig.26 *Ensuring non-negative input*

This program segment will input a data. If the input is not greater than 0, the computer will ask for input again. This ensures that the input is positive. It is equivalent to the following:

```
10  INPUT X
20  IF NOT (X > 0) THEN
30        GOTO 10
40  END IF
```

Example 17 The following program will find out how many consecutive integers starting from 1 are needed to make up a sum of at least 10.

```
10  S ← 0
20  C ← 0
30  REPEAT
40      C ← C + 1
50      S ← S + C
60  UNTIL S >= 10
80  OUTPUT C
```

After executing line 60	C	S	Repeat or not
1st time	1	1	Yes
2nd time	2	3	Yes
3rd time	3	6	Yes
4th time	4	10	No

It is equivalent to the following:

```
10  S ← 0
20  C ← 0
30  C ← C + 1
40  S ← S + C
50  IF NOT (S >= 10) THEN
60      GOTO 30
70  END IF
80  OUTPUT C
```

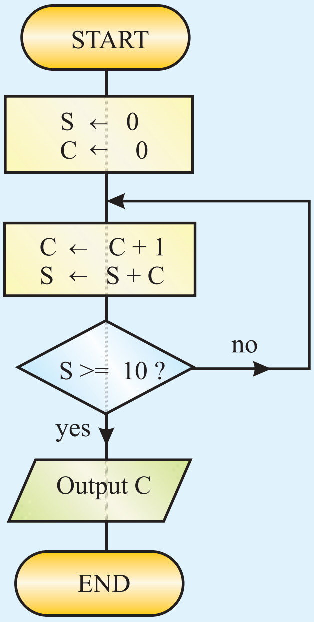


Fig.27 *Application of Repeat ... until*

16.7 The While-loop

Keypoints

A while-loop may or may not take place.

The format of the **while-loop** is

```
WHILE {condition is satisfied}
      {actions to repeat}
END WHILE
```

The actions inside the loop may or may not take place.

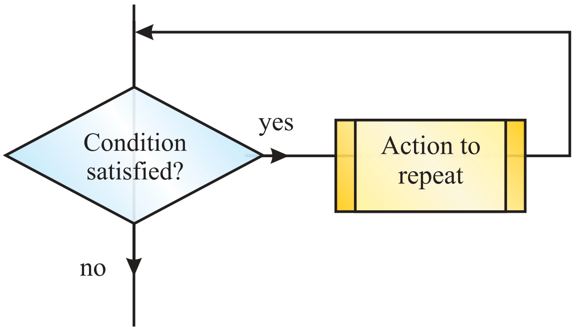


Fig.28 *Iteration control structure: The While loop*

Consider the following example:

```
10  INPUT X
20  WHILE X < 0
30        INPUT X
40  END WHILE
```

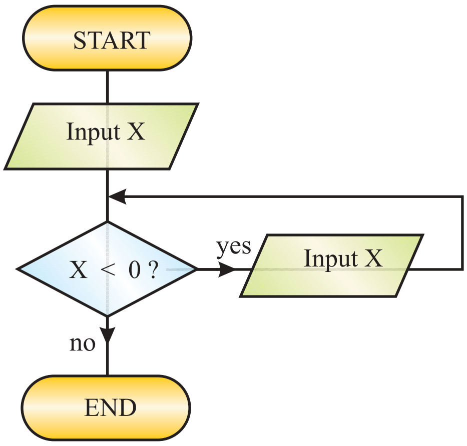


Fig.29 *Ensuring non-negative input*

This program segment will input a data. If the input is not greater than 0, the computer will ask for input again. This ensures that the input is non-negative. It is equivalent to the following:

```
10  INPUT X
20  IF X < 0 THEN
30        INPUT X
40        GOTO 20
40  END IF
```

Example 18 The following program will find out the remainder when a positive number is divided by another.

```
10  INPUT N
20  INPUT DIVISOR
30  WHILE N >= DIVISOR
40        N ← N - DIVISOR
50  END WHILE
60  OUTPUT N
```

It is equivalent to the following:

```
10  INPUT N
20  INPUT DIVISOR
30  IF N >= DIVISION THEN
40        N ← N - DIVISOR
50        GOTO 30
60  END IF
70  OUTPUT N
```

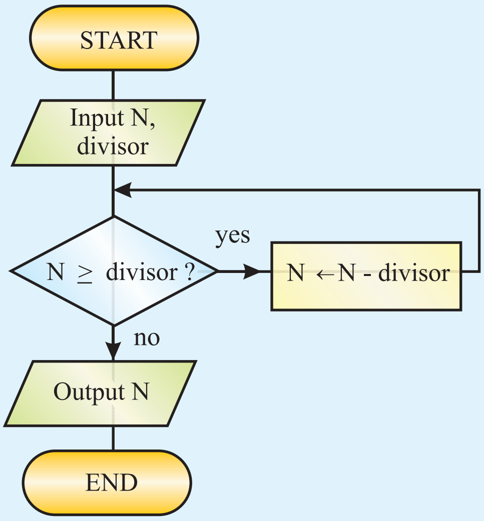


Fig.30 *Using a while-loop to find the remainder*

Continued Example 18

If the number (N) is 15 and the divisor is 4, the program will subtract the number by 4 for 3 times i.e. perform 15 - 4 - 4 -4 = 3. If the number is 3 and the divisor is 4, the program will not perform any subtraction and the remainder is 3.

Since a repeat...until-loop will perform the task at least once. If the while-loop is replaced by a repeat...until-loop as follows, it will lead to a logical error in case the divisor is greater than the number.

```
10  INPUT N
20  INPUT DIVISOR
30  REPEAT
40         N ← N - DIVISOR
50  UNTIL N < DIVISOR
60  OUTPUT N
```

For instance, if N is 3 and the divisor is 4, line 40 will be executed. As a result, it will give -1 as the remainder.

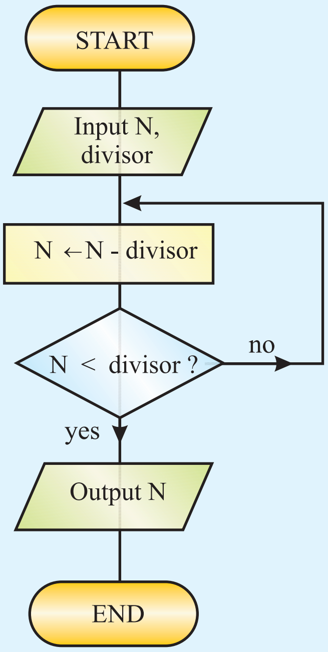


Fig.31 *Logical error may result if a Repeat...until-loop is used to find the remainder*

Summary

1. A variable is a memory location that holds data. An assignment statement assigns value to a variable, overwriting its contents.
2. A computer always follows the proper order of precedence as we do mathematics.
3. An input statement allows user to change the value of a variable interactively; an output statement puts the value on the screen.
4. A conditional statement will carry out an action if a specific condition is met. The condition is tested using a conditional expression that includes relational operators, such as =, <>, >, <, >= and <=. The result of a conditional expression is a Boolean value: either true or false. The Boolean operators that work on two Boolean values are AND, OR and NOT.
5. A branching statement will change the sequence of execution using GOTO command.
6. Iterations mean repeating a sequence of instruction for a specified number of times or under certain conditions. Constructs include: For-loop, Repeat ... until-loop and While-loop.
7. A Repeat ... until-loop will take place at least once. A While-loop may or may not take place.

Review Exercise

Multiple Choice Questions

1. Which of the following statements consist of syntax errors?
   (1) x + 1 ← 4
   (2) x ← 5 + 8 * 2
   (3) x ← "5" + 5

   A. (1) only
   B. (3) only
   C. (1) and (3) only
   D. (2) and (3) only

2. A conditional statement consists of
   A. IF ... THEN
   B. assignment statement
   C. GOTO
   D. WHILE

3. An iteration statement may consist of
   (1) REPEAT .. UNTIL
   (2) WHILE .. END WHILE
   (3) FOR .. NEXT

   A. (1) only
   B. (3) only
   C. (1) and (2) only
   D. (1), (2) and (3) only

4. An input statement
   (1) must consist of at least one variable
   (2) is used with an output statement
   (3) must not be used in a FOR loop

   A. (1) only
   B. (3) only
   C. (1) and (2) only
   D. (2) and (3) only

5. Which of the following will give a true result?
   A. (X > X + 5) AND (1 > 2)
   B. (3 > -7) OR (-7 > 3)
   C. (5 >= 5) AND (4 >= 5)
   D. (7+3 < 8) OR (-7 > -6)

1. C
2. A
3. D
4. A
5. B
6. D
7. D
8. C
9. B
10. C

6. Consider the following program:

```
10    X ← 3
20    Y ← 2
30    Z ← X
40    Y ← Z
50    X ← Z
```

After execution, the contents of the variables are

	X	Y	Z
A.	2	2	2
B.	2	3	2
C.	3	2	3
D.	3	3	3

7. Consider the following program:

```
10    INPUT X
20    IF X > 5 THEN
30          X ← 5
40    ENDIF
```

The purpose of the program is to

A. allow 5 to be entered only

B. allow numbers less than or equal to 5 to be entered only

C. ensure that the value of X is greater than 5

D. ensure that the value of X is less than or equal to 5

8. How many times will the following For-loop iterate?

```
100    FOR I = 6 TO 10
 ..
900    NEXT
```

A. 0

B. 4

C. 5

D. 6

9. How many times will the following While-loop iterate?

```
110    I ← 6
120    WHILE I < 10
 ..
900          I ← I + 1
910    END WHILE
```

A. 0

B. 4

C. 5

D. 6

10. How many times will the following Repeat..Until-loop iterate?

```
110    I ← 6
120    REPEAT
       ..
900        I ← I + 1
910    UNTIL I > 10
```

A. 0
B. 4
C. 5
D. 6

Conventional Questions

1. (a) P = 4, Q = 4, R = 4 (b) P = 1, Q = 11, R = 8 (c) P = 4 , Q = -15, R = 8
2. (a) P = 2, Q = -6 (b) P = 96, Q = 0.25 (c) P = 42 , Q = 36
3. (a) 110 (b) 1000 (c) 10

1. Consider the following program segments:

```
(a)                    (b)                    (c)
10  P ← 1              10  P ← 3              10  P ← -2
20  R ← 8              20  Q ← 4              20  Q ← 3
30  Q ← P + 3          30  R ← P + 5          30  R ← P * (-4)
40  P ← Q              40  P ← P - 2          40  P ← R / 2
50  R ← P              50  Q ← R + 3          50  Q ← Q * (-5)
```

For each case, determine the values of P, Q and R after execution.

2. Consider the following program segments:

```
(a)                    (b)                    (c)
10  P ← 6              10  P ← 4              10  P ← 12
20  Q ← 2              20  Q ← 6              20  Q ← 6
30  P ← P + Q          30  P ← P * Q          30  P ← P - Q
40  Q ← Q - P          40  Q ← Q / P          40  Q ← Q * P
50  P ← P + Q          50  P ← P / Q          50  P ← P + Q
```

For each case, determine the values of P and Q after execution.

3. Consider the following program segments:

```
(a)                      (b)                      (c)
10  X ← 10               10  X ← 10               10  S ← 10
20  S ← 100              20  S ← 100              20  S ← S / 2
30  IF X  < 2 THEN       30  IF 2*X  < 15 THEN    30  IF S/2 > 2.5 THEN
40      S ← S - 10       40      S ← S / 10       40      S ← S / 2
50  ELSE                 50  ELSE                 50  ELSE
60      S ← S + 10       60      S ← S * 10       60      S ← S * 2
70  ENDIF                70  ENDIF                70  ENDIF
```

For each case, determine the values of S after execution.

4. Fill in the following program to make it reasonable:

```
10    INPUT X
20    IF _______ THEN
30        OUTPUT "X is positive"
40    ELSE
50        IF _______ THEN
60          OUTPUT "X is zero"
70        ELSE
80          OUTPUT "X is negative"
90        END IF
100   END IF
```

4. 20 IF X > 0 THEN

50 IF X = 0 THEN

5. (a) 200 (b) 1750 (c) 5550(d) 15000

6. (a) 6 (b) 4 (c) 4

5. The following program calculates the salary tax for a given net chargeable income X. The tax to be paid is put in T.

```
120   INPUT X
130   IF X < 35000 THEN
140       T ← X * 0.02
150   ELSE
160       IF X < 70000 THEN
170         T ← 700 + (X - 35000) * 0.07
180       ELSE
190         IF X < 105000 THEN
200                T ← 3150 + (X - 70000) * 0.12
210         ELSE
220                T ← 7350 + (X - 105000) * 0.17
230         ENDIF
240       ENDIF
250   END IF
```

Determine the salary tax if the net chargeable income is

(a) 10,000
(b) 50,000
(c) 90,000
(d) 150,000

6. Determine the number of 1's printed in each of the following programs:

(a)

```
10   X ← 0
20   OUTPUT 1
30   X ← X + 1
40     IF X <= 5 THEN
50        GOTO 20
60   ENDIF
```

(b)

```
10   X ← 1
20   OUTPUT 1
30   X ← X + 1
40     IF X < 5 THEN
50        GOTO 20
60   ENDIF
```

(c)

```
10   X ← 1
20     IF X < 5 THEN
30        X ← X + 1
40        OUTPUT 1
50        GOTO 20
60   ENDIF
```

7. Determine the output of the following program:

```
10      S ← 100
20      X ← 10
30      S ← S - X
40      X ← X * 2
50      IF S > 0 THEN
60              GOTO 30
70      ELSE
80              OUTPUT S, X
90      END IF
```

8. Determine the output of the following programs:

(a)

```
10      X ← 2
20      FOR I = 2 TO 8
30              X ← X + 3
40      NEXT
50      OUTPUT X
```

(b)

```
10      X ← 2
20      REPEAT
30              X ← X + 3
40      UNTIL X > 20
50      OUTPUT X
```

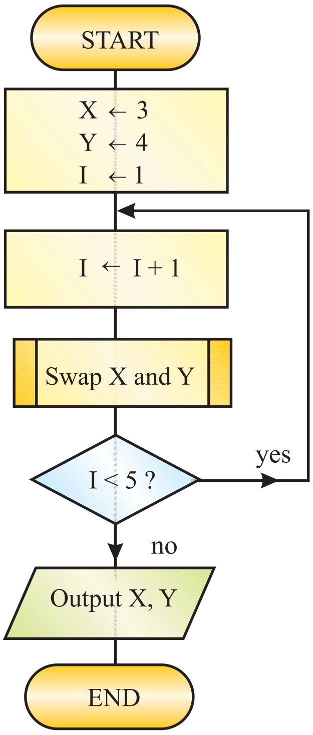
(c)

```
10      X ← 2
20      WHILE X < 20
30              X ← X + 3
40      END WHILE
50      OUTPUT X
```

9. Consider the program flowchart on the right hand side:

(a) Write a program segment for the process "Swap X and Y".

(b) Determine the output of the flowchart.



10. Write a program to determine the minimum value of ten numbers inputted from the keyboard. You may assume that the numbers are all less than 100.

7. -50, 160

8. (a) 23 (b) 23 (c) 23

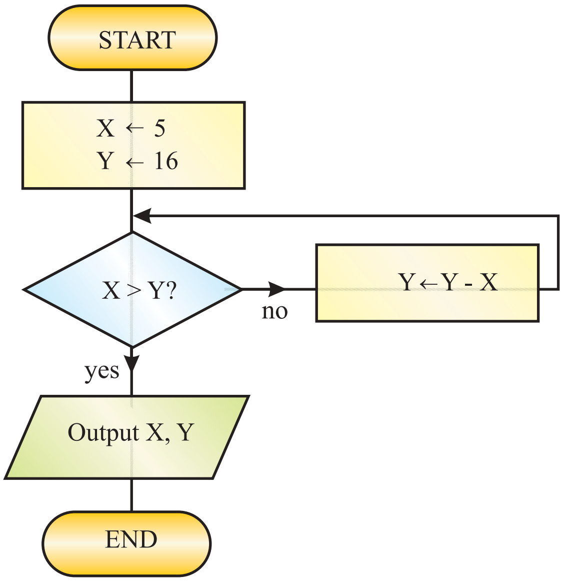
9. (a) Z ← X; X ← Y; Y ← Z
   (b) 3, 4

10.

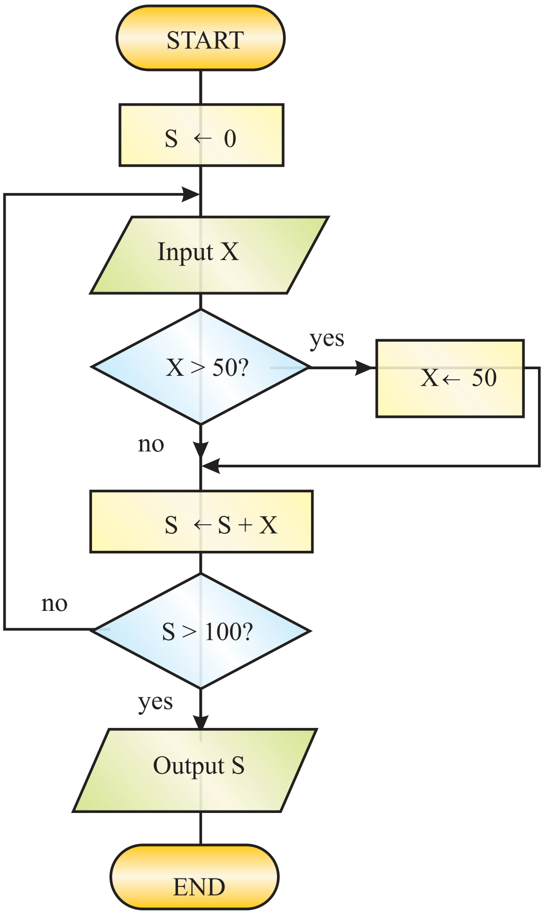
```
10  MIN ← 100
20  FOR I = 1 TO 10
30          INPUT X
40          IF X < MIN THEN
50                  MIN ← X
60          ENDIF
70  NEXT
80  OUTPUT MIN
```

11. X: 5 Y: 1
12. (a) 110
(b) 148

11. What are the outputs from the following program flowchart?



12. Consider the following program flowchart:



State the output from the program if the inputs are

(a) 10, 20, 60, 10, 20

(b) 120, 130, -2, 200