Chapter 2: Scilab Programming

Scilab is a software which enables the users to work more freely for utilizing its available computing functionalities by developing their own programs for any method or algorithm. Scilab has its own programming syntax for various statements which are required for any programming language. Among foremost fundamental statements of such a programming language are branching statements and looping statements. In this chapter the syntax and use these statements are discussed in the context of Scilab as a programming language

2.1 Branching statements

Branching statements are used to enable a program for handling the decision making situations while they are run with practical inputs. These statements are categorized depending upon the type of decisions to be taken. Such basic statements are described below one by one.

The if statement

This statement allows executing desirable sequence of actions depending on if a certain logical condition gets satisfied. The syntax for this statement is

```
if (logical test) then
        Scilab Command 1
        Scilab Command 2 ...
end
```

The syntax of “if statement” in Scilab as given above, first evaluates the condition given through the expression “logical test”. If the output of this evaluation comes out with Boolean value “True”, then the Scilab commands written between the keywords then and end are executed sequentially.

Whereas, for the case, when the output of this evaluation is comes out with Boolean value “False”, then nothing is executed.

Let us understand this concept through a small program coded in Scilab editor “SciNotes”.

Example 2.1: The following program having an “if statement” tests whether a given real number is equal to zero, and then displays an appropriate message only for the case when output of the logical test is “True” (*i.e.*, when the number is equal to zero).

```
clear r = input("Enter a real number ") if (r == 0) then
      disp("The given number is equal to zero.") end
```

The output of the above program upon execution is demonstrated below for two different inputs.

Output:

```
-->exec('D:\SCILAB\Ch3_1_Ex1_if_Statement.sce', -1)
Enter a real number 0
 The given number is equal to zero.
-->exec('D:\SCILAB\Ch3_1_Ex1_if_Statement.sce', -1)
Enter a real number 2
-->
```

It is to be noted here that in case of first execution of the program, as the input is number zero, therefore a message is displayed due to the logical test (r == 0) having truth value “True”. Whereas, when in second execution of the program a non-zero number is input, nothing appears as a message, because there is no command to be executed if the logical test gives truth value “False”.

Let us consider a situation when it is required to execute certain sequence of commands for one outcome of the logical test whereas another

sequence of commands to be executed for another outcome. Then it is appropriate to use another branching statement, as detailed below.

The if-else statement

This statement allows executing **exactly one out of two** desirable sequences of actions depending on if a certain logical condition gets **satisfied** or **not**. The syntax for this statement is

```
if (logical test) then Scilab
        Command A1
Scilab Command A2
... else
Scilab Command
B1
        Scilab Command B2 ...
end
```

The syntax of "if-else statement" in Scilab as given above, first evaluates the condition given through the expression "logical test". If the output of this evaluation comes out with Boolean value "True", then the Scilab commands Scilab
Command A1... written between the keywords then and else are executed sequentially. Whereas, for the case, when the output of this evaluation is comes out with
Boolean value "False", then the Scilab commands Scilab Command B1... written between the keywords else and end are executed sequentially.

Let us understand this concept through another small program coded in Scilab editor "SciNotes".

Example 2.2: The following program having an "if-else statement" tests whether a given real number is equal to zero. It then displays a message for the case when output of the logical test is "True" (*i.e.*, when the number is

equal to zero), whereas, in case of output of the logical test “False”, displays another message.

```
clear r = input("Enter a real number ") if (r == 0) then
disp("The given number is equal to zero.") else disp("The
given number is not equal to zero.") end
```

The output of the above program upon execution is demonstrated below for two different inputs.

Output:

```
-->exec('D:\SCILAB\Ch3_1_Ex2_if_else_Statement.sce', -1)
Enter a real number 0.001
  The given number is not equal to zero.
-->exec('D:\SCILAB\Ch3_1_Ex2_if_else_Statement.sce', -1)
Enter a real number 0
  The given number is equal to zero.
-->
```

It is to be noted here that different messages are displayed in both the executions of the program, depending upon the truth value of the logical test (r == 0) as “True” or “False”.

A branching statement with multiple logical tests can also be used sequentially for execution of different sequences of commands for different outcomes of logical tests. Such a statement is explained below.

The if-elseif-else statement

This statement allows multi-stage testing for execution of **exactly one** sequence of actions, among many, depending on which combination of logical conditions gets **satisfied** or **dissatisfied**. A basic syntax for this statement is

if (logical test 1) then

```
        Scilab Command A1
        Scilab Command A2
        ...
elseif (logical test 2) then
        Scilab Command B1
        Scilab Command B2
        ...
else
        Scilab Command C1
        Scilab Command C2 ...
end
```

The syntax of “if-elseif-else statement” in Scilab as given above, first evaluates the condition given through the expression “logical test 1”. If the output of this evaluation comes out with Boolean value “True”, then the Scilab commands Scilab Command A1... written between the keywords then and elseif are executed sequentially. Whereas, for the case, when the output of this first evaluation is comes out with Boolean value “False”, then further the expression “logical test 2” is evaluated. If the output of this second evaluation comes out with Boolean value “True”, then the Scilab commands Scilab Command B1... written between the keywords elseif and else are executed sequentially. Whereas, for the case, when the output of this second evaluation is comes out with Boolean value
“False”, then the Scilab commands Scilab Command C1... written between the keywords else and end are executed sequentially.

Let us understand this concept through a small program coded in Scilab editor “SciNotes”.

Example 2.3: The following program having an “if-elseif-else statement” tests whether a given real number is zero, positive, or negative.

```
clear
r = input("Enter a real number ") if (r == 0)
then
```

```
    disp("The given number is equal to zero.") elseif (r > 0)
then
    disp("The given number is positive.") else
    disp("The given number is negative.") end
```

The output of the above program upon execution is demonstrated below for three different inputs.

Output:

```
-->exec('D:\SCILAB\ Ch3_1_Ex3_if_elseif_else_Statement.sce', 1)
Enter a real number 2
The given number is positive.
-->exec('D:\SCILAB\ Ch3_1_Ex3_if_elseif_else_Statement.sce', 1)
Enter a real number -5
The given number is negative.
-->exec('D:\SCILAB\ Ch3_1_Ex3_if_elseif_else_Statement.sce', 1)
Enter a real number 0
The given number is equal to zero.
-->
```

It is to be noted here that different messages are displayed in each of the executions of the program, depending upon the truth value of the logical test (r == 0) and further of the logical test (r > 0).

A branching statement with multiple logical tests can also be used sequentially for execution of different sequences of commands for different outcomes of logical tests. Such a statement is explained below.

The select statement

The select statement allows combining multi-stage testing several branches in a clear and simple way.

```
select n case i1
then
        Scilab Command A1
        Scilab Command A2
        ...
case i2 then
        Scilab Command B1
        Scilab Command B2
        ...
        ...
else
        Scilab Command C1
        Scilab Command C2 ...
end
```

The syntax of "select statement" in Scilab as given above, takes the value n and keeps matching with evaluations i1 , i2, … and implements the sequence of commands exactly for that case whose evaluation matches the value n. Otherwise, the Scilab commands written between the keywords else and end are executed sequentially.

Let us understand this concept through a small program coded in Scilab editor "SciNotes".

Example 2.4: The following program uses the "select statement" to check multiple test condition and executes the Scilab commands accordingly.

```
clear
r = input("Enter the Roll Number ") select r
case 1 then
        disp("Your Roll Number is 1.") case 2 then
        disp("Your Roll Number is 2.") else
        disp("Entered Roll Number is not in the list.") end
```

The output of the above program upon execution is demonstrated below with two different inputs.

Output:

-->exec('D:\SCILAB\Ch3_1_Ex4_select_Statement.sce', -1)

Enter the Roll Number 1

Your Roll Number is 1.

-->exec('D:\SCILAB\Ch3_1_Ex3_ select_Statement.sce', -1)

Enter the Roll Number 5 Entered Roll Number is

not in the list.

→

Exercise 2.1

1. Write a Scilab program to test whether a given number divides the other given number.
2. Write a Scilab program to test whether a given number is even or odd.
3. Write a Scilab program to test whether a given number is purely real number or a complex number.
4. Write a Scilab program to test whether a given number is positive, negative, or zero.
5. Write a Scilab program to test whether a given number is positive, negative, or zero, using select statement.
6. Write a Scilab program to solve a Quadratic Equation $ax^2 + bx + c = 0$. The input to the function are the values "a, b, c" and the output of the function should be in the variable names "p, q" appropriately declared.

2.2 Looping statements

Looping statements are used to enable a program to repeat a sequence of commands for a finite number of times. There are two types of such statements available in Scilab. These are appropriately used depending on the situation that whether the termination of the loop is known exactly through

the number of repetitions of commands or expressed in terms of some logical test condition.

2.2.1 The for statement

This statement allows executing desirable sequence of actions depending on if a certain logical condition gets satisfied. For any variable name i and a vector of values v, the syntax of the "for statement" is given below.

```
for i = v
    Scilab Command 1
    Scilab Command 2 ...
end
```

Syntax of the "for statement" in Scilab as given above, indicates that the sequence of Scilab commands will be executed repeatedly sequentially for each value of the vector v assigned to the variable i. Let us learn this concept through some examples.

Example 2.5: The following program computes the sum of first 5 natural numbers.

```
Sum = 0 for i=1:5
    Sum = Sum + i
end disp("Final Sum")
disp(Sum)
```

In the above program coded in SciNotes, the variable i takes repeatedly and sequentially values from the vector 1:5 and corresponding to each value taken by the variable i, it is added to the variable Sum. Each time in the loop, the value of i is added to the variable Sum, the variable Sum gets value of the partial sum to finally get the desired value at the termination of the loop after variable i completing the commands for value i=5. The output of this program is as following.

-->exec('D:\SCILAB\Ch3_2_Ex5_for_loop.sce', -1)

Final Sum

15.

In the Example 3.4 discussed above, the basic colon operator is used. General colon operator with specific step length can also be used in “for loop”, as demonstrated in the next example.

Example 2.6: The following program demonstrates the sum of odd natural numbers from 1 to 10.

```
Sum = 0 for i=1:2:10
    Sum = Sum + i
end
disp("Sum of odd natural numbers between 1 and 10") disp(Sum)
```

In the above program coded in SciNotes, the variable i takes repeatedly and sequentially values from the vector 1:2:10 (*i.e.*, odd numbers between 1 and 10) and corresponding to each value taken by the variable i, it is added to the variable Sum. The output of this program in terms of the final value of variable Sum gives the desired results, which is depicted as following.

-->exec('D:\SCILAB\Ch3_2_Ex6_for_loop.sce', -1)

Sum of odd natural numbers between 1 and 10

25.

In the Examples 3.4 and 3.5 discussed above, the colon operators are used appropriately to define vectors. Even, any vector of values can in general be used in “for loop”, as demonstrated in the next example.

Example 2.7: The following program demonstrates the sum of values of a pre-defined vector.

```
Sum = 0 v = [1, 9, %e,
5] for i=v
```

```
    Sum = Sum + i end
disp("Sum of values in the vector v:") disp(Sum)
```

In the above program coded in SciNotes, the variable i takes repeatedly and sequentially values from the vector v and corresponding to each value taken by the variable i, it is added to the variable Sum. The output of this program in terms of the final value of variable Sum gives the desired results, which is depicted as following.

```
-->exec('D:\SCILAB\Ch3_2_Ex7_for_loop.sce', -1)
Sum of values in the vector v:

    17.718282
```

Exercise 2.2

1. Write a Scilab program to compute sum of first ‘n’ natural numbers.
2. Write a Scilab program to compute factorial of a natural number ‘n’.
3. Write a Scilab program to obtain the Fibonacci sequence with ‘n’ members, and Fibonacci series with ‘n’ terms.
4. Write a Scilab program to test whether a given number is prime number or not.
5. Write a Scilab program using for loop to compute the sum of two given matrices, if they are of comparable order.
6. Write a Scilab program using for loop to compute the matrix multiplication of two given matrices, if they are of comparable order. Verify the obtained matrix by using Scilab matrix multiplication operator ‘*’.
7. Write a Scilab program to sorting (arrange) a set of numbers in ascending and descending order.
8. Write a Scilab program to compute the number of permutations & number of combinations for given values of ‘n’ and ‘r’.

The while statement

Some situations appropriate to looping statements are encountered some times during the programming where it the termination point for the loop cannot be identified in advance as an exact number, unlike the for loop. Rather, in such cases, the termination is identified through some criterion which is expressible as a logical test condition. Such situations are appropriate to be programmed as a "while statement".

The general format of while statement is

```
while (logical test)
        Scilab Command 1
        Scilab Command 2
        ...
end
```

Let us learn use of while statement through the following example.

Example 2.8: The following program computes the sum of digits of a natural number.

```
n=input("Enter a natural number: ") t=n Sum=0
while (t~=0)
        digit = pmodulo(t,10)
Sum=Sum+digit     t=int(t/10) end
disp("The sum of digits of the number " + string(n) + " is "+ string(Sum)+".")
```

The output upon the execution of this program is as following.

```
-->exec('D:\SCILAB\Ch3_2_Ex8_while_loop.sce', -1)

Enter a natural number: 145

 The sum of digits of the number 145 is 10.
```

Exercise 2.3

1. Write a Scilab program to find the number of digits of a natural number '*n*'.
2. Write a Scilab program to obtain a number with digits as the reverse of a given natural number '*n*'.
3. Write a Scilab program to test whether a given number is Palindrome.
4. Write a Scilab program to test whether a given number is Armstrong number.
5. Write a Scilab program to obtain the binary equivalent of a given decimal number.
6. Write a Scilab program to obtain the decimal equivalent of a given binary number.
7. Write a Scilab program to compute sum of first '*n*' prime numbers.

_____********_____