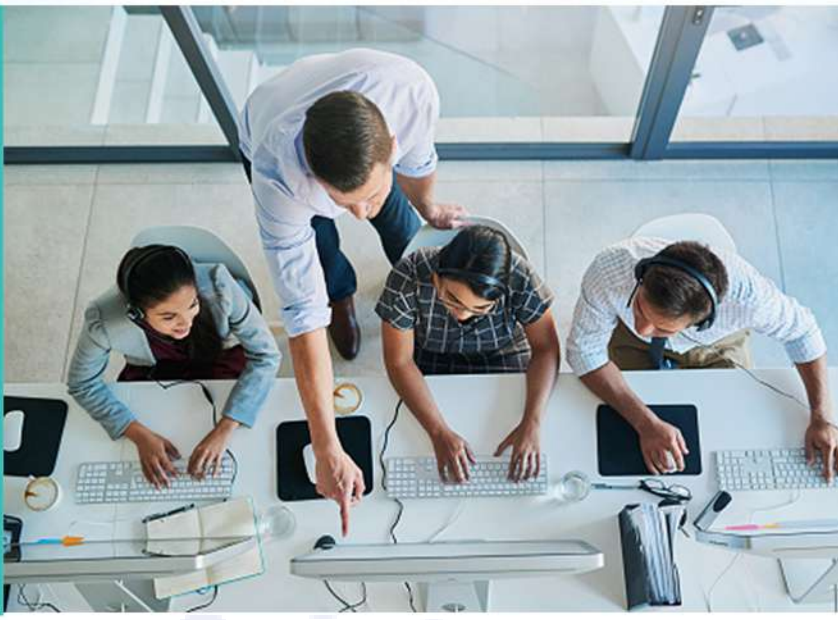
Andhra Pradesh State Skill Development Corporation



SciLab

Programming in Scilab
Conditional statements-1

Programming in Scilab

INTRODUCTION

The programming style in Scilab is similar to C programming. If you are already familiar with C programming most of the programming techniques might sound trivial. In this case, you may skim this chapter to take note of the subtle differences alone. Arithmetic, Relational and Logical operators have already been introduced, hence we will only make some additional observations. Our focus will be on variables, assignment, expressions, input & output, flow of control, and functions.

VARIABLES &VARIABLE NAMES

We have seen that like in algebra where we use variables, Scilab also permits the use of variables. Similar to our saying in algebra "Let a=5", we can have a Scilab statement too. Here a is called a variable name or identifier. Variables can be named using a combination of alphabets digits and special characters. Like all modern programming languages, Scilab is case sensitive. Variable g and G are treated as two different variables. There are certain rules to be followed when naming variables. They are:

1. It should use only alphabets, numbers, and a specified set of special characters(&, ! #, $). It should not use spaces, @, ^ ( ), and operator symbols.
2. It must have at least one alphabet. It should not begin with a number.
3. It should not be a reserved word (a word that has a special meaning in Scilab).
4. It should not be longer than 24 characters(if longer, it gets truncated).

Following are some of the reserved words in Scilab: if, else, then, select, case, for, end, and all function names (There is no need to remember all of these. Once you are familiar with Scilab, the common reserved words can be easily recognized). Unlike in C, Scilab may not always complain about variable names which are reserved words, but unexpected results might occur.

Examples of valid identifiers : average, number123, _temp, $sum, %#acc,#a1, %i

Examples of invalid identifiers, along with reasons:

Total sum space not allowed

123 number name should not begin with a number

895 no alphabets

@abc use of special character @ is not allowed

a+b=7 use of operators is not permitted

In Scilab, variables are dynamically created through the assignment operator =. Any defined variable will always have a value.

ASSIGNMENT STATEMENTS

As already discussed above, in Scilab it is not necessary to declare variable names or specify dimensions in advance. These are automatically set to the assigned value. An assignment statement in Scilab is of the form “variable =expression” where expression is a number, a variable or a formula involving these and arithmetic operators see the following examples:

```
a=20;
Interest =5.75;
Grade=’p’;
x=y+z;
area=3.14*r*r;
operator =’+’;
```

Note: character values should always be enclosed within single quotes. Only one character can be present inside the quotes.

ARITHMETIC, RELATIONAL & LOGICAL OPERATORS

Scilab uses the arithmetic operators +,-,*,/,^ which have already been introduced. Scilab also uses matrix related arithmetic operators such as .* , ./ , .\ , .\ The relational operators used in Scilab are <, <=, >, >=, == and logical operators &,| and ~. Let us now discuss the use of complex expressions and the use of brackets and operator precedence in them.

In the examples, we used only a few operators in each expression. You can use any number of operators and operands in an expression. For example:

```
a+b*c-d
a*b/c
p+q*2
a+b*c-d/d*e
```

However, there are some rules to be followed on such occasions. Suppose we want to calculate a+b/c . We may naturally write a+b/c. However, this may be confused (by you not Scilab!) with an expression such as a+bc which has a different meaning.

For a=6, b=4, c=2

a+bc=6+4/2=10/2 =5 whereas a+b/c=6+4/2=6+2=8

So, in Scilab, if we write a+b/c as in the above case, what could be the result? 5 or 8? Before we answer that, we will learn how to avoid confusion. Always put parentheses to make clear to Scilab what you mean.

1. (a+b)/c can be written as (a+b)/c
2. a+bc can be written as a+(b/c)

Now that you have learned the frequently used operators in Scilab, we will take one more look at assignment statements. Recall that the general form of an assignment statement is variable= expression.

Example of valid assignment statements are:

```
x=y+2;
b=5;
sum=0;
count=n+1;
sum=sum+x;
a=5;
b=5*3;
c=a+b;
d=(a+b) *(x+y+z)/100
```

Example of invalid assignment statements are:

```
x+y=z
s=x+y
(a+b)/c=x
sum+n=sum
3=5;
3=a;
```

It may now be noticed that the interpretation of x,y, in Scilab is not like in algebra (where it could mean the value of x and y are equal). In Scilab, x=y means store the value of the right hand side (y) in the variable at the left hand side (x). As a result, x and y become equal . The meaning of the interpretation becomes clear when we look at the statement count=count+1. Mathematically it is not meaningful. But in Scilab, we have to consider it differently. Consider the following example.

```
count=5
count=count+1
```

In all assignment statements, always look at the right-hand side first. In this example, on the right-hand side, we have count+1 which is 5+1=6. Now store that value in the variable on the left-hand side. Hence count becomes 6. If a variable x appears on the left-hand side and right-hand side, you can read the right-hand side as the current value of x and the left-hand side as a new value of x. x=x+1 may be read as (new value of x)= (current value of x)+1

Let us now go back to the question of what Scilab will do with the expression a+b/c. If given without brackets, Scilab will take it as a+b.

```
a+b/c
```

Scilab will take it as the second one. This is because Scilab has a standard order in which operators are applied. This standard order of arranging operators is known as Scilab's precedence of operators.

Whatever is inside ( ) will be calculated first

*/ Among these, whichever comes first from left

+- Among these, whichever comes first from left

As an example consider the following expression and see how the evaluation is taking place

5+6*2-4

5+12-4

17-4

13.

INPUT & OUTPUT

To enable us to write interactive programs. We will introduce here three simple input and output commands in Scilab they are input ( ) disp( ) printf( ). If you want to read the values during running the program, from the keyboard you can use the input(“ “) as follows.

```
x= input( “ “);
```

Scilab would wait for input when it encounters this statement and proceed only after a value is typed. The typed value will be taken as the value of x. You can add a message within the " "in the input( ) to prompt the user properly as follows.

```
x=input("Type in a value for x:")
```

Now Scilab will wait for an input prompting the user with the message "Type in a value for x" needless to say. This is a friendly way of doing things. The quickest way of displaying values of variables is just typing the name of this variable in the command prompt of Scilab. disp( ) has also the same effect. See its use below.

```
x=5;
disp(x);
x=5
name= "Adithya";
disp(name);
disp("jai hind");
```

However, disp( )doesn't permit fine formating of the output. For this printf( ) is useful. printf( ) is very similar to the C facility of the same name. It can be used for all situations where disp( ) is used.

```
x=5;
printf("%d",x);
name= "Adithya";
printf("%s",name)
```

The values of variables can also be shown on the screen along with text in a flexible way using printf. Study the following program.

```
a=5;
printf("The value of variable =%d\n",a);
```

Inside printf brackets you find: " The value of variable=%d\n",a. \n is for effecting a new line. which we will see soon. So let us ignore that \n and have a look at the rest.

```
"The value of variable =%d", a
```

Here we have something inside double quotes followed by a comma(,) and a variable a. Inside the quote, we find a new notation: %d. You may always consider this as a blank to be filled in.

```
"The value of variable =............", a
```

%d actually tells Scilab to print an integer in its place. It is known as a format string. The format strings for different data types are different.

Data type	format string
int	%d
float	%f
char	%c
string	%s

The format strings %d, %f, %c, %s all indicates blanks to be filled in. In addition they indicate the blanks have to be filled in by integer, float, and character values respectively.

Some examples are a=5;

printf(“ The value of a=%d \n”,a);

p=27.3;

printf(“The value of p=%f\n”,p);

If instead of %f, %d is used, the value of p will be truncated to 27.

x= ‘m’;

printf(“The value of x= %c\n”,x);

Here if %d is used, then Scilab shows an error. We can print out the values of more than one variable using printf as in the examples given below.

a=5;

b=10.56;

x= ‘m’;

printf(“The value of a is %d, p is %f, x is %c\n”, a, p, x);

You may note that the blanks inside the “ “ are filled in with the variable values in the order in which they are given. You must ensure the appropriate format string has been used for each variable type. Otherwise unexpected results may occur.

Let us close this section with a mention of escape characters

We have already learned about \n (back -slash-en) to print a new line on the screen. “/” is called an escape character and there are more:

\a	alert
\b	backspace
\n	new line
\t	tab

Remember that escape characters always appear inside the double quotes " ". See an example for \t escape sequence. The effect is that between two variables, there will be 7 spaces between them.

```
a=2;
b=3;
c=5;
printf("a=%d,t b=%d, \tc=%d", a, b, c);
```

FLOW CONTROL/BRANCHING/ CONDITIONAL STATEMENTS

In Scilab programs we have written so far, all statements in the program were executed one after the other, in the order in which they appeared in the program. Scilab permits this order flow of control to be changed if desired. The statements used for this are known as branching statements or conditionals. There are two branching statements in Scilab if/if-else statement, select-case statement.

if/if-else statement

The form of an if statement is as follows:

```
If (condition)
Statement 1:
statement  2:
:
:
Statement n:
end
```

The if statement starts with the keyword 'if' followed by a condition inside ( ), followed by any number of statements ending with "end". The condition is a relational or logical expression like (a-5),(b=-6),(a>b)&&(a>c) etc. If the condition is true, then the statements until "end" are executed, and if false, the statements are ignored.

```
Example
min=40
if(mark<min){
printf("failed\n")
```

The example shown above checks if the mark scored by a student is less than the required pass. If so then displays the message that the student has failed. Here are some examples of how if statements begin:

```
if((a+b)==x)
if((a>b)&&(b>c))
if(!(mark>75))
if((a>b)||(c==d))
```

Whenever you face a situation where you need to say : do these only if such-and such a condition is true, you will find an if statement appropriate.

Let us write a program to read the marks of a student and declare “passed” if marks is greater than 40.

```
printf(“please give your mark\n”);
m-input(“marks”)
if(m>40)(printf(“passed\n”);
end
```

Here is a program to read two numbers and print out the smallest.

```
printf(“Please give two integers\n”)
scanf(“%d%d”, &a,&b);
if(a<b)printf(“%d is the smallest\n”,a); end
if(b<a)printf(“%d is the smallest\n”,b); end
```

Now we look at the if-else statement. The form of an if-else statement is:

```
if(condition)
Do this set of statements if the condition inside( ) is true
else
Do this set of statements if the condition inside( ) is false
```

This simply says that if the expression inside the parentheses is true, the first group of statements is to be executed and if false, the second group of statements following the ‘else’ is to be executed. Thus one of the two sets of statements will always be executed, whereas in the simple if statement, the given set of statements was either executed or skipped.

To demonstrate if else statement, let us write a program to read two numbers and print out the smallest.(Note the difference between this and the example given above for simple if)

```
printf("Please give two integers\n");
a=input("Type a");
b=input("Type b");
if (a<b) printf("%d is the smallest\n",a);end
else print("%d is the smallest \n",b);end
```

Note: predict the result if (i) a=15,b=11 (ii) a=25,b=20 (iii) a=30, b=30

Here is a program to ask the year of birth of a person and calculate the age in the year 2075 and if the result (age) is more than 90 then give a message that It is likely that you may not be alive then

```
printf("I will calculate your age in year 2075\n");
year=input ("year of birth");
age= 2075-year;
printf("Your age again in the year 2075 will be %d\n", age);
if(age<90) printf("I don't think you will be a live then\n");
end
else
printf("I think you will still be alive\n");
end
```