14

MAKING DECISIONS IN "C"

14.1 INTRODUCTION

So far we have seen that in C programs the instructions are executed in the same order in which they appear in the program. Each instruction is executed once and once only. Programs do not include any logical control structures. Most programs, however, require that a group of instructions be executed repeatedly, until some logical condition has been satisfied. This is known as **looping**. Most of the programs require that a logical test be carried out at some particular point within the program. An action will then be carried out whose exact nature depends upon the outcome of the logical test. This is known as **conditional execution**.

14.2 OBJECTIVES

After going through this lesson you would be able to

- define *'while'* statement, *for* statement and *nested loops*
- explain switch statement and goto statement
- define comma operator

14.3 WHILE STATEMENT

The while statement is used to carry out looping operations. The general form of the statement is

while (expression) statement

The loop operates in the following fashion:

The value of the test expression enclosed in parentheses is evaluated. If the result is true, then the program statement (the body of the loop) is executed. The statement may be a compound statement. Then the test expression, which may be just as complex as any of those found in if statement is evaluated again. If it is again true, the statement is executed once more. This process continues until the test expression becomes false. At that point, the loop is terminated immediately, and program execution continues with the statement (if any) following the while loop. If there are no more statements, the program terminates.

Let us consider an example, which prints the five lines. Each line is numbered by printing out the value of x each time around the loop.

```
/ * A test program for while loop */
include<stdio.h>
main()
{
      int x=1;
while(x<6)
{
print(“This is line number %d of test program\n”,x);
x++;
}
}
```

The variable x is assigned a value of 1. The while loop displays the line number of test program by incrementing the value of x by one each time until the value of x is less than 6. Thus the loop will be repeated 5 times, resulting in 5 consecutive lines of output. Thus, when the program is run, the following output will be generated.

This is line number 1 of test program.

This is line number 2 of test program.

This is line number 3 of test program.

This is line number 4 of test program.

This is line number 5 of test program.

This program can be written more concisely as

```
#include <stdio.h>
main()
        {
        int x=1;
        while (x<6)
printf("This is line number %d of test program\n";x++);
        }
```

When executed, this program will generate the same output as the first program.

All variables used in the test expression of the while statement must be initialized at some point before the while loop is reached. In addition, the body of the loop must do something to change the value of the variable used in the expression being tested. Otherwise the condition would remain true and the loop would never terminate. This situation, known as an infinite loop, is illustrated next.

```
while(x<6)
printf("Something is wrong in this loop\n");
```

The loop is infinite because the value of x is never change. If the test expression starts out being true, it remains true forever. In the previous program, the value of x determines how many times the loop executes. Therefore, the second statement in the body of the loop (x++;) serves to increment x by 1. Since x starts out with a value of 1 and is incremented by 1, at some point it must become equal to 6. Then the condition in the while loop becomes false and the loop is terminated.

If x were initialized to a value of 6, the condition in the while loop becomes false to begin with , and the body of the loop would not be executed at all. It is clear, then, that the while statement employs a loop pre-test. The loop condition is tested before each iteration, and therefore before the loop is entered at all. If the condition initially fails, the loop is skipped entirely.

If the test expression involves two variables and joined with 'and' or 'OR' operator

```
        #include <stdio.h>
```

```
main()
      {
      int x=1;
int a=4;
while (x<6 && a>3)
{
printf(" This is line number %d of test program \n",x);
x++;
   }
     }
```

The loop executes only while x is less than 6 and a is greater than 3. If at least one of these conditions becomes false, the loop terminates. If at least one of the conditions is false when the loop is first encountered, the loop is skipped entirely.

The value of a is not changed in the loop, but the value of x is incremented in a way that will make the loop terminate eventually. If the value of a is less than or equal to 3, then the initial test fails and the while loop is skipped entirely.

There is another way to use while loop, instead of incrementing the value of variables inside the loop, you can also decrement it. For example, if you want to print the line number of test program in reverse order then test expression changes and the program is as follows:

```
#include <stdio.h>
main()
{
            int x = 0
            printf("This is line number of test program in reverse
            order");
            while(x>=0)
      {
            printf("%d\n",x);
            x - - ;
      }
}
```

There is an option for test expression in while loop, instead of giving some constant value to test variable you can even given it some variable also. The value of this variable can be input from the user interactively e.g. if the user wants to calculate the average of numbers, but 'how many numbers?' this number can be input from the user itself and then this variable can be used in test expression.

Let us consider an example.

```
#include <stdio.h>
main()
{
int number, n=1;
float x, average, sum=/0;
printf("How many  number ?");
scanf("%d", & number);
while(n<=number)
{
scanf("%f", &x);
sum=sum+x;
++n;
}
average=sum/number;
printf("\n The average is %f  \n", average );
}
```

The output of this program is as follows:

How may number? 6

1.

2.

3.

4.

5.

6.

The average is 3.500000

do……while loop

The do… while loop differs from its counter part, the while loop in that it makes what is called a loop **post-test**. That is the condition is not tested until the body of the loop has been executed once. In the while loop, by contrast, the test is made on entry to the loop rather than at the end. The effect is that even if the condition is false when the do-while loop is first encountered, the body of the loop is executed at least once. If the condition is false after the first iteration, the loop terminates. If the first iteration has made the condition true, however the loop continues.

The general form of the do….while loop is as follows:

do

Statement;

while (test expression);

The fact that the while clause is located after the statement reflects the fact that the test is made after the statement is executed.

If the body of the loop is a single statement, it must be terminated with a semicolon. For example:

do

a=a+10;

while (a<b);

This semicolon marks the end of the inner statement only not of the entire loop construct. In every situation that requires a loop, either one of these two loops can be used. Let us consider an example:

```
#include <stdio.h>
main()
{
int numbers, n=1;
float x,average ,sum=0;
printf("How many number ?");
scanf("%d", &numbers);
do
{
scanf("%f; &x);
sum=sum+x;
```

```
++n;
}
while (n<=numbers);
average=sum/numbers;
printf("\n The average is %f\n", average);
}
```

The output of this program is same as that with while loop.

INTEXT QUESTIONS

1. What happens if the condition in a while loop is initially false?
2. What is the minimum number of times the body of a do... while loop is executed?
3. What is the essential difference between a while and a do.... While loop?
4. Define syntax for do.... While loop?

14.4 FOR STATEMENT

The for statement is the most commonly used looping statement in 'C'. This statement includes an expression that specifies an initial value for an index, another expression that determines whether or not the loop is continued and the third expression that allows the index to be modified at the end of each pass.

The general form of the for statement is

for (expression1; expression2; expression3) statement

Expression1 is the initialization expression, usually an assignment, which is performed once before the loop actually begins execution. Expression 2 is the test expression, exactly like the one used in the while loop, which is evaluated before each iteration of the loop and which, determines whether the loop should continue or be terminated. Finally, expression 3 is the modifier statement, which changes the value of the variable used in the test. This expression is executed at the end of each iteration, after the body of the loop is executed. Statement is the body of the loop, which may as usual be compound. The three loops expressions are separated by two semicolons. No semicolon should be placed after expression 3.

The first expression of the for loop can be omitted if the variable is initialized outside the loop. If one or more expressions are omitted from the for loop, the two semicolons still must appear, even if they are not preceded or followed by anything. Let us understand the concept of for loop with the help of an example:

```
#include <stdio.h>
main()
{
int x;
for(x=1; x<=10; x ++)
printf("This is line number %d of test program\n",x);
}
```

The output for this program is as follows:

```
This is line number 1 of test program
This is line number 2 of test program
This is line number 3 of test program
This is line number 4 of test program
This is line number 5 of test program
This is line number 6 of test program
This is line number 7 of test program
This is line number 8 of test program
This is line number 9 of test program
This is line number 10 of test program
```

In order to print the tenth line, the test expression must use the <= relational operators. If < alone were used, the loop would print only nine lines.

Let us consider one more program to print first five even numbers through use of for loop.

```
#include <stdio.h>
main()
{
    int i;
    for(i=2; i<=10; i=i+2)
```

```
printf("%d\n", i);
}
```

The output of this program is as follows:

2

4

6

8

10

In the for loop, if the second expression is omitted, however it will be assumed to have a permanent value of 1 (true) thus the loop will continue indefinitely unless it is terminated by some other means, such as a break or a return statement.

INTEXT QUESTIONS

5. Which is the better loop to use, the for loop or the while loop?
6. What is a special advantage of the for loop?
7. What separates the three expressions of a for statement?
8. How does the for loop operate?

14.5 NESTED LOOPS

Loops can be nested or embedded one within another. The inner and outer loops need not be generated by the same type of control structure. It is essential, however that one loop be completely embedded within the other there can be no overlap. Each loop must be controlled by a different index. Let us consider an example of nested for loops:

```
#include <stdio.h>
main()
{
int i,j,n,sum;
for(i=1;i<=5; i++)
{
printf("\nEnter a positive number:");
scanf("%d",&n);
```

```
            sum=0;
            for(j=i; j<n; j++)
            {
            sum=sum+j;
            printf("\n The sum of the integers from 1 to %d
is:%d\n",          n,sum);
            }
}

  }
```

The output is as follows: -

Enter a positive number: 5
The sum of the integers from 1 to 5 is: 15

Enter a positive number: 10
The sum of the integers from 1 to 10 is: 55

Enter a positive number: 15
The sum of the integers from 1 to 15 is: 120

Enter a positive number: 20
The sum of the integers from 1 to 20 is: 210

Enter a positive number: 5
The sum of the integers from 1 to 25 is: 325

In this program firstly, outer for loop will be executed, if the expression (test) is true, sum variable is initialized to zero every time and then inner loop will be executed. Inner loop will be executed till the test expression of inner loop satisfies. Then if the expression is false, the pointer again goes to outer loop and then it will reexecute. For nested loops any other loop structures could also have been selected.

14.6 IF...ELSE STATEMENT

C allows decisions to be made by evaluating a given expression as true or false. Depending upon the result of the decision, program execution proceeds in one direction or another. This is carried out in 'C' by 'if' statement. The simplest form of the if statement is as follows:

if (expression)

Statement;

'if' statement will execute if the given expression is true. If the user wants to add more than one statement then there must be pair of curly braces after 'if'.

```
if  (expression)
{
statement;
statement;
}
Let us consider an example of if statement.
#include <stdio.h>
main()
{
char grade;
printf("Enter a character value for grade:");
scanf("%c", & grade);
if(grade = = 'A')
printf("The grade is excellent \n");
printf("Thanks for using this program\n");
}
```

The output is as follows:

Enter a character value for grade: A

The grade is excellent

thanks for using this program

If we re-run the program with different value for grade than the output is

Enter a character value for grade: C

Thanks for using this program

In the second case, the value for grade is 'C' other than 'A' which is in test expression of if statement, thus the first printf statement will not execute.

In the previous program only the general message is printed if the

user selects a grade other than 'A'. If the user wants to display other message when enter a grade other than 'A', you have to add the else clause of if statement. The general form of if..else is as follows.

```
if (statement)
Statement 1;
else
Statement 1;
```

Both if and else clause are terminated by semicolons. Let us consider an example of if…else statement.

```
#include <stdio.h>
main()
{
        char grade;
        printf("Enter a character value for grade:");
        scanf("%c", &grade);
        if(grade= ='A')
        printf("grade is excellent \n");
        else
        printf("grade is other than excellent\n");
}
```

The user can use compound statements both in if and else statements. The first printf is executed if and only if grade is equal to 'A', if grade is not equal to 'A', the first printf is ignored, and the second printf, the one following the word else, is executed.

A clause of the if statement may itself contain another if statement, this construct known as nesting of if statements. Let us consider an example nested if..else statement:

```
#include <stdio.h>
main()
{
int year;
printf("Enter a year to check for leap year");
scanf("%d", & year);
if (year %4= = O),
```

```
if(year %100 != 0)
printf("%d is a leap year:\n", year);
else
if(year %400 = =)
printf("%d is a leap year \n",year);
else
printf("%d is not a leap year \n", year);
else
printf ("%d is not a leap year\n", year);
}
```

It is very important to be sure which else clause goes with which if clause. The rule is that each else matches the nearest if preceding it which has not already been matched by an else. Addition of braces prevents any association between the if statement within the braces and the else clause outside them. Even where braces are not necessary, they may still be used to promote clarity.

INTEXT QUESTIONS

9. In an if statement, if two separate statements are to be executed when the comparison is true, what must be done with them?

10. What is the function of the else clause in an if statement?

14.7 SWITCH STATEMENT

The switch statement causes a particular group of statements to be chosen from several available groups. The selection is based upon the current value of on expression that is included within the switch statement. The general form of the switch statement is

switch (expression) statement

Where expression results in an integer value. Expression may also be of type char, since individual characters have equivalent integer values. The embedded statement is generally a compound statement that specifies alternate courses of action. Each alternative is expressed as a group of one or more individual statements within the overall embedded statement. For each alternative, the first statement within the group must be preceded by one or more case labels. The case labels identify the different groups of statements and

distinguish them from one another. The case labels must therefore be unique within a given switch statement.

Thus, the switch statement is in effect an extension of the familiar if...else statement. Rather than permitting maximum of only two branches, the switch statement permits virtually any number of branches.

In general terms, each group of statements is written as

```
case expression1;
case expression2;
      :
      :
case expression m:
Statement 1
Statement 2
      :
Statement n
```

Where expression 1, expression 2.... Expression n represent constant, integer valued expressions. Each individual statement following the case labels may be either simple or complex. When the switch statement is executed, the expression is evaluated and control is transferred directly to the group of statements whose case-label value matches the value of the expression. If none of the case-label value matches the value of the expression, then none of the groups within the switch statement will be selected. In this case control is transferred directly to the statement that follows the switch statement.

Let us consider an example of switch statement:

```
      #include <stdio.h>
      main()
{
      char vowel;
      printf("Enter a character");
      scanf("%c",&vowel);
      switch(vowel)
{
      case'a'
```

```
        case'A':
                printf("vowel");
                break;
        case 'e' :
        case 'E':
                printf("vowel");
                break;
        case 'i' :
        case 'I' :  printf("vowel"); break;
        case' o' :
        case 'O' :
                printf("vowel");
                break;
        case 'u' :
        case 'U' :
                printf("vowel");
}
     }
```

One of the labeled groups of statements within the switch statement may be labeled default. This group will be selected if none of the case labels matches the value of the expression. The default group may appear anywhere within the switch statement. If none of the case labels matches the value of the expression and default group is not present, then the switch statement will take no action.

```
#include <stdio.h>
main()
{
        char vowel;
        printf("Enter a character");
        scanf("%c", & vowel);
        switch(vowel)
{
        case 'a':
        case 'A':
              printf("vowel");
```

```
            break;
        case'e':
        case 'E':
            printf("vowel");
            break;
        case 'i' :
        case 'I':
            printf("vowel");
            break
        case'o':
        case 'O':
            printf("vowel");
            break;
        case'u'
        'u':
            printf("vowel");
            break;
        default:
            printf("Not a vowel");
        }
}
```

The default label can be placed anywhere within the body of the switch statement; it need not be the last label. In fact, the labels of a switch statement can appear in any order, depending on the logic of the program. It is desirable for several different values of the switch variable to cause execution of the same set of statements. This can be accomplished by including several labels in succession with no intervening statements.

14.8 THE BREAK STATEMENT

The break statement is used to force fully terminate loops or to exit from a switch. It can be used within a while, a do-while, for or a switch statement. The format is simple as

```
break;
```

without any embedded expression or statements. The break statement causes a transfer of control out of the entire switch statement, to the first statement following the switch statement.

If a break statement is included in a while, in do while or in for loop, then control will immediately be transferred out of the loop when the break statement is encountered. Thus provides a convenient way to terminate the loop if an error or other irregular condition is detected. Let us consider a program segment of break statement in while loop.

```
#include <stdio.h>
main()
{
        int x, sum=O;
        printf("Enter any number");
        scanf("%d" &x);
        while(x<=50)
        {
                if (x<zero)
        {
        printf("error value because of negative value");
        break;
        }
                scanf("%d", &x);
        }
```

The continue statement is used to bypass the remainder of the current pass through a loop. The loop does not terminate when a continue statement is encountered, instead the remaining loop statements are skipped and the computation proceeds directly to the next pass through the loop. It can also be included within a while, do while or a for statement as like break statement. It is also written simply as

continue;

without any embedded statement or expression.

14.9 THE COMMA OPERATOR

Comma operator is used primarily in conjunction with the for statement. This operator permits two different expressions to appear in situation where only one expression would ordinarily be used.

For example:

for (expression 1a, expression 1b; expression 2; expression 3) statement.

where expression 1a and expression 1b are the two expressions, separated by comma operator, where only one expression would normally appear.

Let us consider an example:

```
#include<stdio.h>
main()
{
        int i,j,n;
      printf("Enter number for a table");
        scanf("%d", &n);
        printf("\n");
        for(i=0, j=n; i<n; i++, j– – )
        printf("%3d+%3d= %3d\n", i,j,n);
}
```

The output is as follows:

Enter number for a table 20

```
 0+20=20
 1+19=20
 2+18=20
 3+17=20
 4+16=20
 5+15=20
 6+14=20
 7+13=20
 8+12=20
 7+13=20
 8+12=20
 9+11=20
```

10+10=20

11+9=20

12+8=20

13+7=20

14+6=20

15+5=20

16+4=20

17+3=20

18+2=20

19+1=20

20+0=20

The comma operator has the lowest precedence. Thus, the comma operator falls within its own unique precedence group, beneath the precedence group containing the various assignment operators. Its associatively is left-to-right

14.10 THE GOTO STATEMENT

The goto statement is used to alter the normal sequence of program execution by transferring control to some other part of the program. It is written as

goto label;

Where label is an identifier used to label the target statement to which control will be transferred. Control may be transferred to any other statement within the program. The target statement must be labeled, and the label must be followed by a colon. Thus the target statement will appear as

Label : statement

No two statements cannot have the same label

Goto statements has many advantages like branching around statements or groups of statements under certain conditions, jumping to the end of a loop under certain conditions, thus bypassing the remainder of the loop during the current pass, jumping completely out of a loop under certain conditions, thus terminating the execution of a loop.

14.11 WHAT YOU HAVE LEARNT

In this lesson, you have learnt about different types of loops like for, do..while, while. You are now familiar with if-else statement and switch statement. You are now also familiar with comma operators, break and continue statement and goto statement. You can very well use all of them in a 'C' program to make a program interactive and user friendly.

14.12 TERMINAL QUESTIONS

1. What is meant by looping? Describe two different forms of looping.
2. What is the purpose of while statement?
3. How is the execution of a while loop terminated?
4. How many times will a for loop be executed, what is the purpose of the index in a for statement?
5. What is the purpose of the switch statement?
6. Compare the use of the switch statement with the use of nested if else statement?
7. What is the use of break and continue?
8. What is the purpose of goto statement. Explain their usage in the program.

14.13 KEY TO INTEXT QUESTIONS

1. The while loop is skipped over
2. One
3. A while loop performs its test before the body of the loop is executed, whereas a do..loop makes the test after the body is executed.
4. do

   Statement;

   while (test expression);
5. It depends on the nature of the problem to be solved, and upon the preference of the programmer

6. The initialization, testing and modifier expressions of the loop all are specified within a single set of parentheses.
7. Semicolons
8. The first expression is executed once. Then the second expression is tested, if it is true, the body of the loop is executed. Then the third expression is executed. Again, second expression is evaluated, till the second expression is true, the body of the loop is executed, followed by the third expression.
9. They must be combined into a compound statement, using the curly braces { and }.
10. If present, it is followed by a statement that is executed only if the condition being tested proves to be false.