Module 4:

Decision-making and forming loops

Learning objectives

1. To understand the concept of control structure
2. To know different types of control structures
3. To know control structure statements in C
4. To know different types of control structures
5. To know control structure statements in C

1. Introduction

Every c program contains a number of statements dedicated to solve some kind of problem in a systematic way. Simple calculations and expressions are only a small part of the computer program. One of the basic properties of a computer is its capability to repetitive execute a sequence of instructions. This property is possible with the help of loop statements present in C language, which helps the programmer to develop concise programs. Executing the same sequence of instructions again and again forms a program loop also known as infinite loop. In this module, the focus is on how to form a finite loop. Decision making statements used in the loop structure determine when to terminate the program loop.

The control structures define the way or flow in which the execution of **C** language statements should take place. There are number of situations, where it is necessary to change the order of execution of statements based on certain conditions (decision making) or repeat a group of statements (forming loops) until certain specified conditions are met.

When one of the many alternatives is to be selected, the programmer can use many if statements to control selection. But this may lead to complex program if number of alternatives increases. C language has a built-in multi-way decision statement known as a **switch**. In addition, C language also has many loop control statements like while, do and for statements.

It is sometimes necessary to terminate the loop or control the loop exits other than normal exit. In such situation programmer can use break or continue statements. Figure-1 shows the classification scheme for control structures.

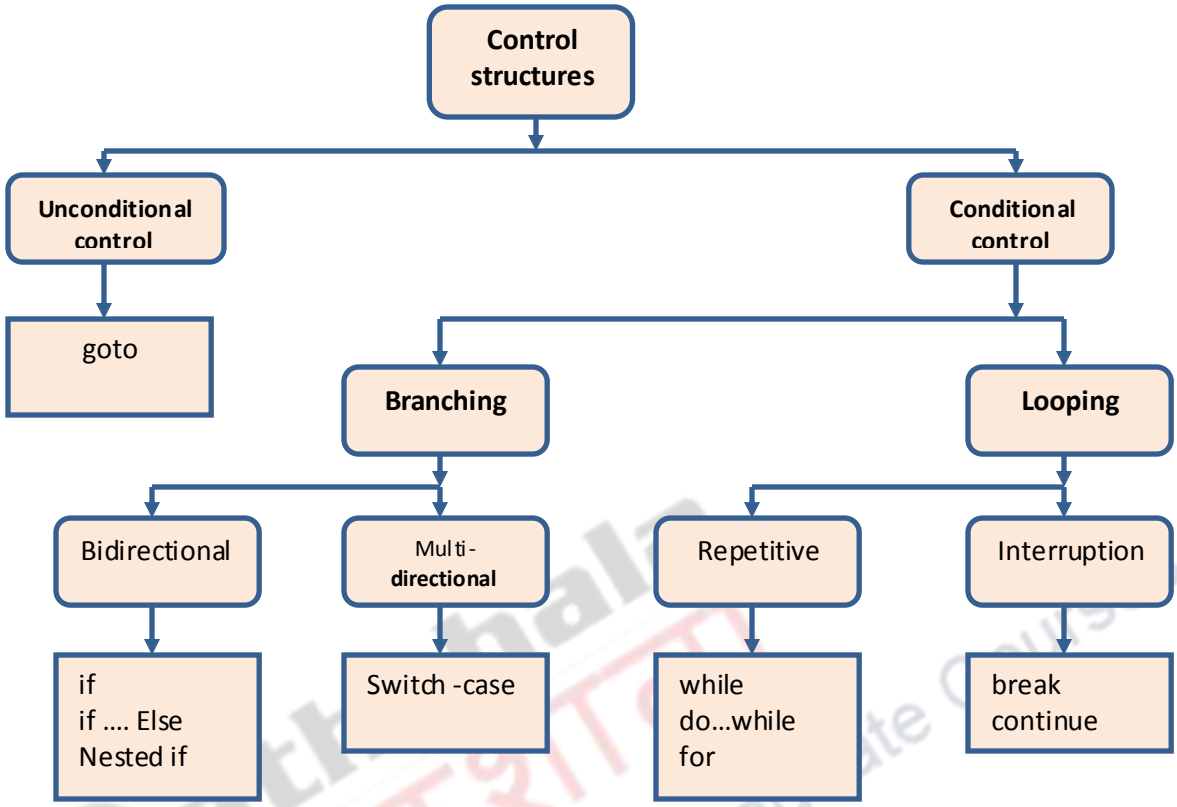


Figure-1: Classification of control structures

2. Decision making

The c language possesses decision making ability and supports following statements known as **decision making** or control statements:

(a) The if statement

(b) The if….else statement

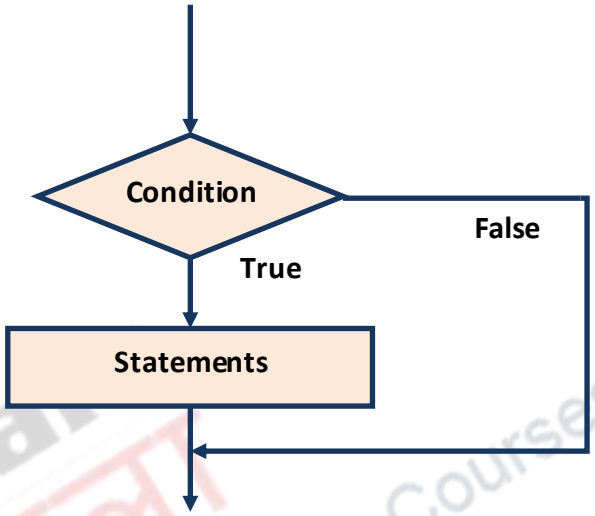
(c) Nested if statement

2.1 The if Statement

The if statement is used to make decisions. The "**if statement**" is a powerful decision making statement and is used to control the flow of execution of statements. It allows decision to be made by evaluating a given expression as true or false. Such an

expression involves the relational and logical operators. Depending on the outcome of the decision, program execution proceeds in one dereliction or another. **if** is basically a two way decision statement. The simplest form of the if statement is as follows,

```
if (expression)
        statement;


or

if (expression)
    {
      statement;
       ……….
      statement;
    }
```



The keyword **if** tells the compiler that what follows is a decision control statement. The expression which defines the conditions is always enclosed in the parenthesis. If the condition is true, then the statement is executed. If the condition is not true, then the statement is not executed. Instead program skips it.

```
/*Demonstration of if statement*/
main()          //function
   {
      int num;     //variable declaration
      printf("Enter a number less than 10");
      scanf("%d",&num);
      if(num<=10)
           printf("Number is less than 10");
   }
```

On execution of this program, if you type a number less than or equal to 10, you get a message on the screen through printf(). If you type some other number the program doesn't do anything.

2.2 The if…. else statement

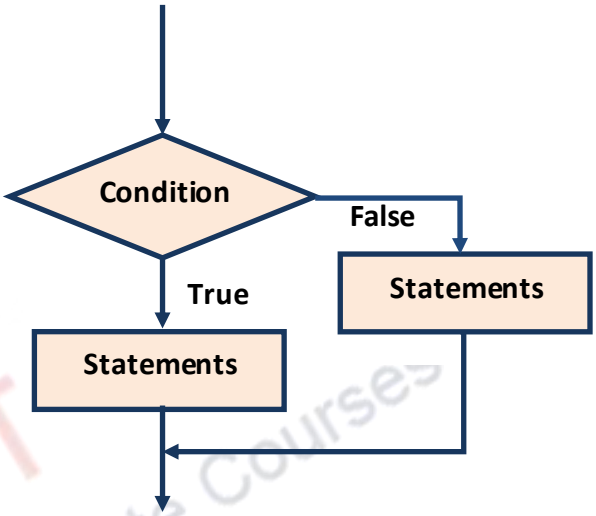
The "if statement" by itself will execute a single or a group of statements when the condition (or expression) is true. It does nothing when condition is false. However there

are many situations when two groups of statements are given and it is desired that one of them be executed if some condition is true and the other be executed if some condition is false. Such situation demands the use of if…else statement. The **if…. else** statement is an extension of simple if statement. The general form is

```
if (expression)
  statement-1;
else
 statement-2;
```

OR

```
if (expression)
  {
    ……..
   statements
    …….
  }
else
  {
    ……..
   Statements
    ……..
  }
```



When an "if statement" occurs during program execution, the value of expression is non-zero (true) then statement-1 is executed. If the value of expression is zero (false), statement-2 is executed. In either case, one of the statement -1 or statement-2 is executed but not both.

2.3 Nested if statement

As seen earlier, the if clause and the else part may contain a compound statement. Moreover, either or both may contain another if or if…….else statement. This is called as nesting of if…. else statements

Example for Nested If Else

```
#include <stdio.h>
void main()
{
  int marks;
  printf("Enter your marks : ");
```

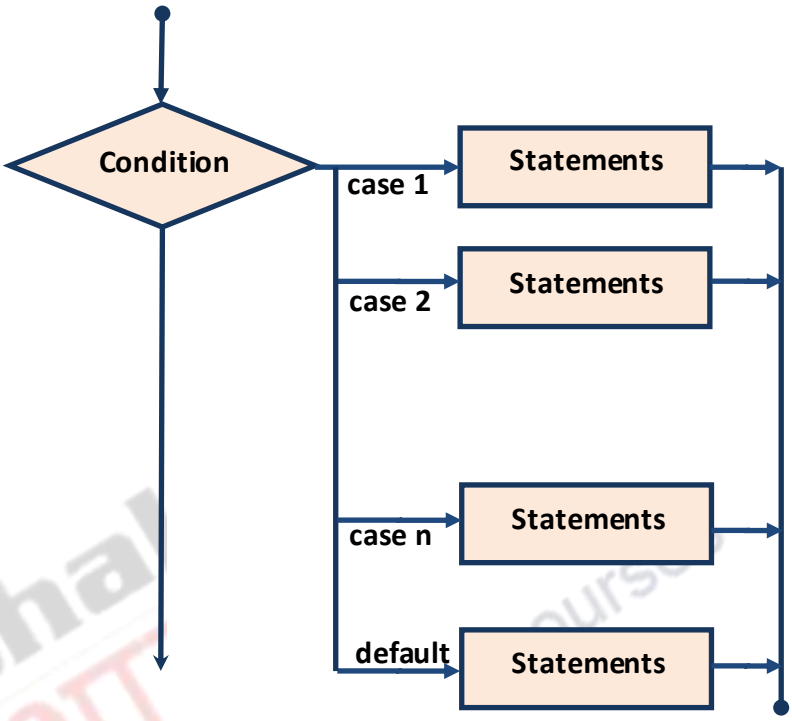
```
  scanf("%d",&marks);

  if(marks>100)       /*marks greater than 100*/
    printf("Not valid marks");
  else if(marks>=80)  /*marks between 80 & 99*/
    printf("your grade is A");
  else if(marks >=70) /*marks between 70 & 79*/
    printf("your grade is B");
  else if(marks>=50)  /*marks between 50 & 69*/
    printf("your grade is C");
 else if(marks>=35)   /*marks between 35 & 49*/
    printf("your grade is D");
  else
                        /*marks less than 35*/
    printf("your grade is E");
}
```

3. The Switch case

While writing the program with many alternative using **if** statements, the complexity of program increase tremendously. In addition to this, the program becomes difficult to read and trace. To overcome this problem, c language has provide a built-in multi way decision statement known as **Switch**.

The switch statement tests whether an expression matches with one of the constant values from a list of case values. When a match is found, a block of statements associated with that case is executed. The syntax of switch statement is as follows:

```
switch (expression)
{
  case value-1:
        statement(s);
        break;
  case value-2:
        statement(s);
        break;
   ……..
   ……..
  default:
        statement(s);
        break;
}
```

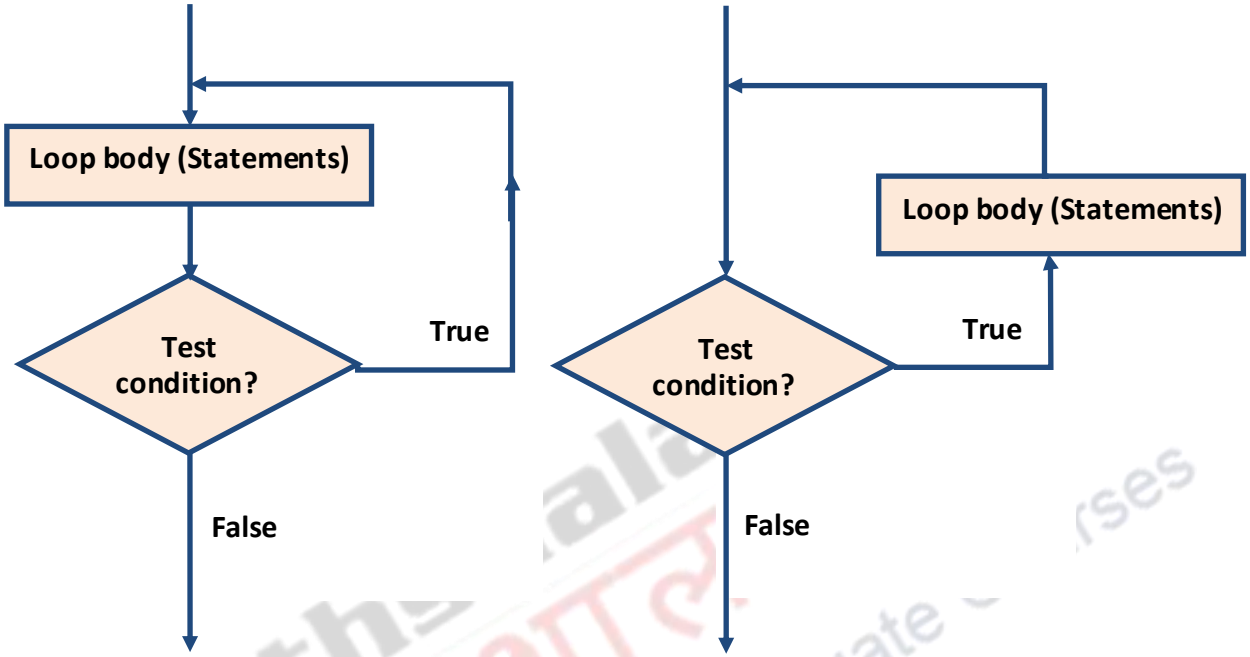


The expression in switch statement is an integer expression or character type. Each case statement must be followed by the value to be compare to. Value-1, value-2,…… are constant or constant expressions. Each of these values must be unique within the switch statement. When a variable is equal to one of the many cases then the statements given in the block will be executed until a break statement is reached. The break statement at the end of each block indicates the end of the particular case and causes an exit from the switch statement. A switch statement can have an optional default case, which must appear at the end.

4. Forming loops

The loop statements allow a sequence of program statement to be executed repeatedly until the test condition is false. In looping structure, a sequence of instructions is executed until some conditions for termination of loop are satisfied. A program loop consists of two parts, body of the loop and control statement. The control statement tests certain condition and then directs the repeated execution of statements contained in the body of the loop. Depending upon the position of the test condition in the loop,

loop structure may be classified either as pre-test (entry controlled) loop and post-test (exit controlled) loop as shown below.



A looping process, usually includes four steps:

1. Initialization of a counter
2. Execution of statements in the loop
3. Test condition for the execution of the loop
4. Increment the counter

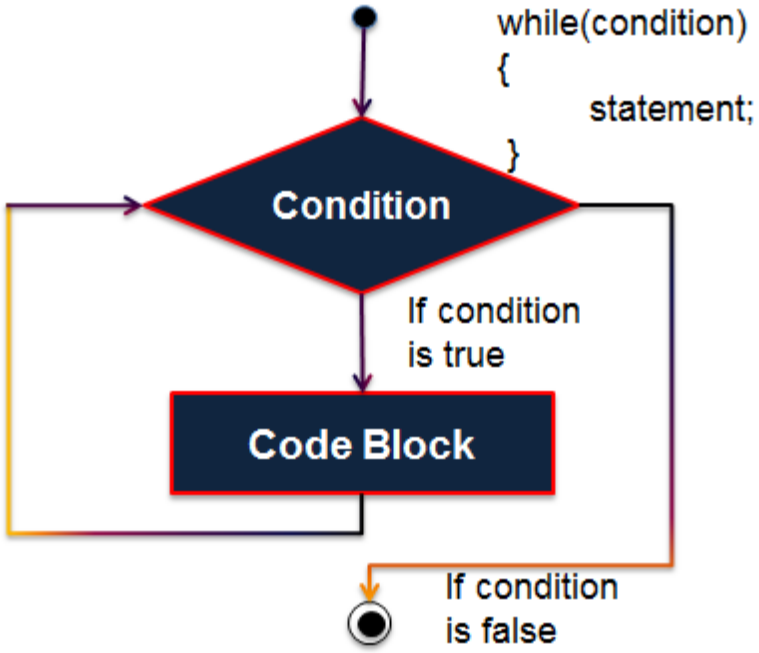
The test may be either to determine whether the loop has been executed (repeated) the specified number of times or to determine whether a particular condition is met.

There are three types of loop statements in C language

1. The while statement
2. The do statement
3. The for statement

4.1 The While statement

The simplest loop structure in C language is the while statement. It is a pre-test loop structure. The while statement is preferred when one is not sure about whether the loop will be executed or not. After is checking the condition is true or false, the loop may be executed. If condition is true then the program enters into the loop operation. The while has the form:



The statement is only executed if the expression is non-zero. After every execution of the statement, the expression is evaluated again and the process repeats if it is non-zero. The only point to watch out for is that the statement may never be executed, and that if nothing in the statement affects the value of the expression then the while will either do nothing or loop forever, depending on the initial value of the expression.

The simple meaning is (a) evaluate the expression (b) if its value is true (i.e., not zero) do the statement, and go back to (a) Because the expression is tested before the

statement is executed, the statement part can be executed zero times, which is often desirable.

```
#include <stdio.h>
int main ()
  {
   int a = 5;

    while( a < 15 )   //  while loop execution
       {
        printf("value of a = %d\n", a);
        a++;
       }
   return 0;
  }
```

4.2 The do ….. while statement

The do statement is used whenever the programmer is sure about the test condition. It is occasionally desirable to guarantee at least one execution of the statement before testing the condition. As the test condition is evaluated after the execution of the loop, hence it is a post-test loop structure. The general form of do …. While loop is:

Syntax:

```
do
  {
     Statement(s)
  }
while(expression);
```

Semicolon—it is not optional. The effect is that the statement part is executed before the controlling expression is evaluated, so this guarantees at least one trip around the loop. It is undoubtedly essential in certain cases, but experience has shown that the use of do statements is often associated with poorly constructed code. Not every time, obviously, but as a general rule you should stop and ask yourself if you have made the

right choice. Their use often indicates a hangover of thinking methods learnt with other languages, or just sloppy design.

Consider the following example

```
# include<stdio.h>
int x=3;
main()
{
  do
   {
    printf("x=%d \n",x--);
    }
   while (x>0);
 }
```

outputs:-

x=3

x=2

x=1

NOTE: The postfix x- operator which uses the current value of x while printing and then decrements x.

You can use a do….while loop whenever you want the loop body to be executed at least once, otherwise use a while loop.

4.3 The for Statement

The for loop is very flexible, powerful and most commonly used statement in C. A special statement that constructs an iterative loop (a loop that is repeated a specified number of times). The for statement is a somewhat generalized while that lets us put the initialization and increment parts of a loop into a single statement along with the test.

Syntax

```
for( expression-1; expression-2; expression-3 )
  {
    Statement(s);
  }
Or
for (i=initial;i<=limit;i+=step)
{
    Statement(s);
  }
```

The for loop consist of three expressions. The first expression is for initialization of the loop index variable such as i=1, count=10 etc. The variable I, count are the loop control variables. The second expression is used to check whether or not the loop be continued or not. It sets the limit on the control variable such as i<5, count <max etc. The third expression is used to change the value of loop index variable for the next iteration.

General execution process for a for statement is described below.

1. Steps in Executing a for Loop
2. Control variable assigned initial value.
3. Check if loop repetition condition is true. If it is not, do NOT execute the loop, and jump to the statement following the body of the loop.
4. Execute the body of loop.
5. Change the value of the Control variable.
6. Repeat steps 2, 3, and 4 until the loop condition checked in step 2 is false.

The rules for the for Loop

1. INITIAL, LIMIT, and STEP can be: Positive OR Negative; Integer OR Double; Constants OR Variables OR Expressions
2. Do not change the value of i, the control variable, inside the loop, unless there is no step function in the for statement.

3. Changing the value of INITIAL inside the loop has no effect on the number of times the loop is executed.
4. After the for loop is completed, the control variable, i, contains the last value that exceeded the LIMIT.
5. You can have a for loop without an initialization statement and without a change of the loop control variable: for (;i<6;) The variable must then be initialized elsewhere, and updated WITHIN the for loop.

Example 1: Consider the following program which prints the values from 1 to 5

```
# include<stdio.h>
main()
  {
   int i;
   for (i=1; i<=5, i++)
      {
      printf("%d\t", i);
      }
   }
```

The output is displayed as 1 2 3 4 5

Example2: Add n numbers

```
#include <stdio.h>

int main()
{
  int n, sum = 0, c, value;

  printf("Enter the number of integers you want to add\n");
  scanf("%d", &n);

  printf("Enter %d integers\n",n);
```

```
for (c = 1; c <= n; c++)
   {
    scanf("%d",&value);
    sum = sum + value;
    /*adding each no in sum*/
    }

printf("Sum of entered integers = %d\n",sum);

return 0;
}
```

Nested for loop

Nesting is a programming concept where one control structure is placed inside another. Nested loops can be extremely useful, but also confusing to deal with. Remember for each iteration of the outer loop, the inner loop is executed and repeated until done. Two for loops must use different control variables.

Example3: Print Pattern pyramid

```
#include <stdio.h>

int main()
{
  int row, c, n, temp;

printf("Enter the number of rows in pyramid of stars you wish to see
");
  scanf("%d",&n);

  temp = n;

  for ( row = 1 ; row <= n ; row++ )
  {
  for ( c = 1 ; c < temp ; c++ )
    printf(" ");
    /*Leaving spaces before printing * */
```

```
    temp--;

    for ( c = 1 ; c <= 2*row - 1 ; c++ )
      printf("*");
      /*Printing stars by calculating rows */

    printf("\n");
      /*moving to new line after printing one */

  }

    return 0;
  }
```

5. Loop interruption

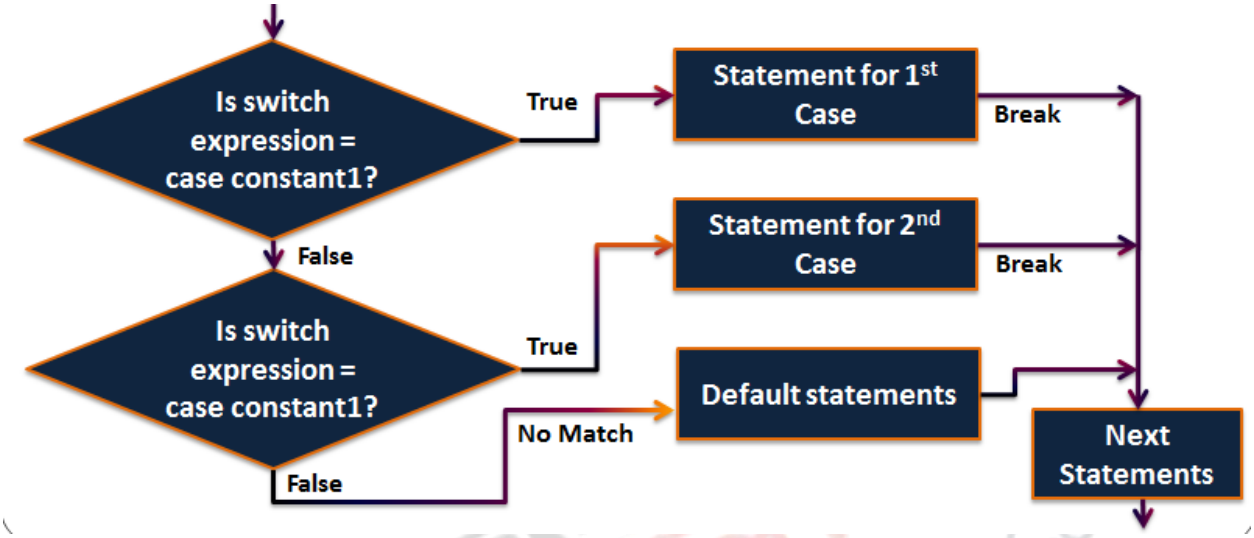
In many situations, it is desirable to control the loop exits other than the loop termination condition. It is sometimes desirable that a particular iteration be interrupted without exiting the loop. There are three statements available in C for such purpose.

1. The break statement
2. Continue statement
3. Goto statement

5.1 The break statement

The controlled statement is executed until the loop is broken by some means. The simplest and commonest way of doing this is by using a break statement. A break statement consists simply of the word break followed by a semi-colon. Its effect is to cause immediate exit from the enclosing while statement. (It has the same effect when used with do and for statements, these will be described in due course.) It provides a means for writing a simpler and tidier version of the interactive program that appeared earlier in these notes. This is a simple statement. It only makes sense if it occurs in the body of a switch, do, while or for statement. When it is executed the control of flow jumps to the statement immediately following the body of the statement containing the

break. Its use is widespread in switch statements, where it is more or less essential to get the control that most people want.



The following example illustrates how to use the `break` statement:

```
#include <stdio.h>

int main()
{
   char key;

   printf("Press any key or E to exit:\n");
   while(1) {
      scanf("%c", &key);
      // if E or e, exit
      if (key == 'E' ||  key == 'e')
          break;
   }
   printf("Goodbye!\n");
}
```

The program asked users to enter any character. If the user enters E or e, The break statement terminates the loop and control is passed to the statement after the loop that displays the Goodbye message.

Besides using the break statement to terminate a loop, we also use the break statement to terminate the processing of a case branch in the statement.

5.2 The continue statement

This statement has only a limited number of uses. The rules for its use are the same as for break, with the exception that it doesn't apply to switch statements. Executing a continue starts the next iteration of the smallest enclosing do, while or for statement immediately. The use of continue is largely restricted to the top of loops, where a decision has to be made whether or not to execute the rest of the body of the loop. In this example it ensures that division by zero (which gives undefined behaviour) doesn't happen.

Write a C program to find the product of 4 integers entered by a user. If user enters 0 skip it.

```
//program to demonstrate the working of continue statement in C programming
# include <stdio.h>
int main(){
inti,num,product;
for(i=1,product=1;i<=4;++i){
printf("Enter num%d:",i);
scanf("%d",&num);
if(num==0)
continue;/ *In this program, when num equals to zero, it skips
the statement product*=num and continue the loop. */
product*=num;
}
printf("product=%d",product);
return0;
}
```

5.3 The goto statement

The goto statement is used to alter the program execution sequence by transfer of control to some other part of the program. The syntax of goto statement is:

```
goto  label;
```

The goto statement can be used in two different ways: unconditional goto where the control is unconditionally transferred and conditional transfer where contro is transferred after testing the condition.

Example:

```
if (a> b)
 {
  large =a;
  goto output;
 }
else
 {
 large=b;
 goto output;
 }

Output: printf("largest=%f", large);
```

6. Summary

The control structures defines the way in which the execution of C language program should take place. if .. else statements are used to select between two alternative statements based on the value of expression. It is also possible to have nested it statements for multiple alternatives. To avoid complexity of program, switch case statements are preferred. Repetitive statements create program loops. The while statement checks its expression before executing other statements in the loop. The do statement checks its expression at the end of the loop. This ensures that the body of the loop is executed at least once. The for loop is extremely used when the fixed iterations are to be made. There are loop interruption statements like break, continue and goto.

Development Team

Principal Investigator:	Dr. Arvind D Shaligram, Savitribai Phule Pune University, Pune
Paper Coordinator:	Dr. Vijay P Labade, Fergusson College, Pune
Content Writer:	Dr. Vijay P Labade, Fergusson College, Pune
Content Reviewer:	Dr. Ajay Kumar (Director), Jayawant Institute of Business Studies Tathwade, Pune