Unit : 4 Decision and Loop Control Statements

Lesson Structure

4.0 Objective

This unit will help you to

- Understand the importance of various control statements defined in programming language;
- Utilize different control statements to get the desired output;
- Transfer the control from within the loop efficiently;
- Make use of goto, break and continue statement in the programs;and
- Write the program more efficiently by making use of branching as well as looping statement.

4.1 Introduction

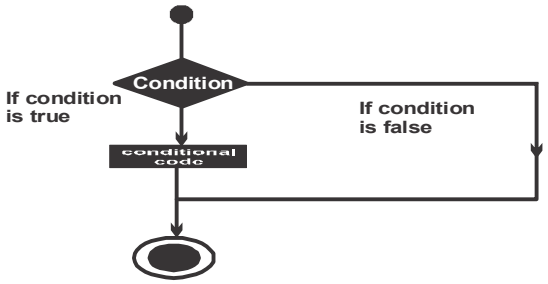
In our daily life we all need to alter our actions according to the changing situation. for example if the weather is sunny, then you will prefer to have cold drinks or juice but if the weather is cold, then you will prefer to have coffee or hot soup. As you notice that the above decision depends on the condition of weather.

A program consist of a number of statement to be executed by the computer. By default the instructions in a program are executed sequentially. Many a time, we want a set of instructions in a programs to be executed in one situation and an entirely different set of instructions to be executed in another situation.

4.2 Control Statements

Decision making structure require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Following is the general form of a typical decision making structure found in most of the programming languages :



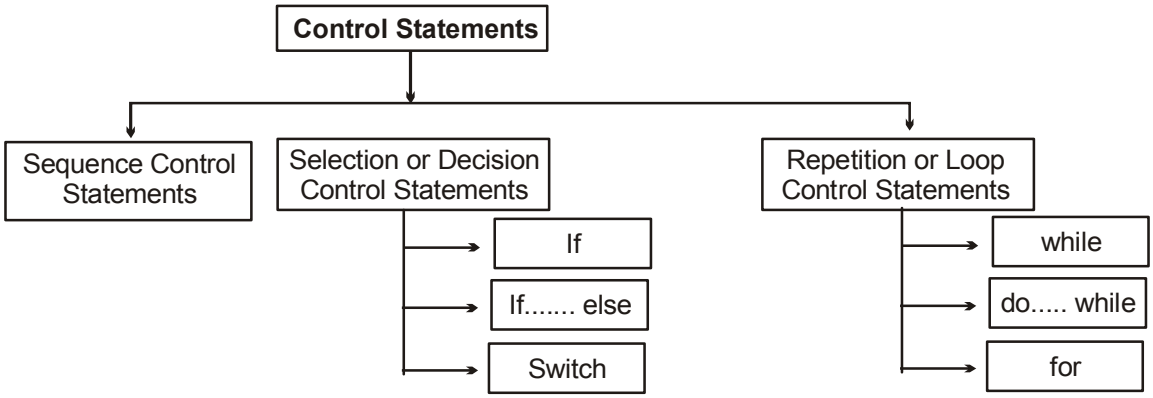
The Control Statements of a language is used to determine the "Flow Of Control" in a program. C programming language provide three types of control statements.

(i) Sequence Control Statements.

(ii) Selection or Decision Control Statements.

(iii) Repetition or Loop Control Statements.

The figures shows the categorization of Control Statements in C.



4.3 Sequence Control Starements

The sequence control statements ensures that the instruction in the program are executed in the same order in which they appear in the program. This is usually built into the languages as a default action.

4.4 Selection or Decision Control Statement

These statement allow selective processing of a statement. Selection means executing different section of code depending on a specific condition. This allows a program to take different courses of actions depending on different condition. These are also called as conditional Statements.

C programming language assumes any non-zero and non-null values as true, and if it is either zero or null,then it is assumed as false value.

C programming languages provides following types of selection or decision control statements.

Statement	Description
if statement	An it statement consists of a boolean expression followed by one or more statements.
if .... else statement	An if statement can be followed by an optional else statement, which executes when the boolen expression is false.
nested if statements	You can use one if or else if statement inside another if or else if statements.
switch statement	A switch statement allows a variable to be tested for equality against a list of values.
nested switch statement	You can use one switch statement inside another switch statement (s).

4.4.1 if statement

An if statement consists of a boolean expression followed by one or more statement.

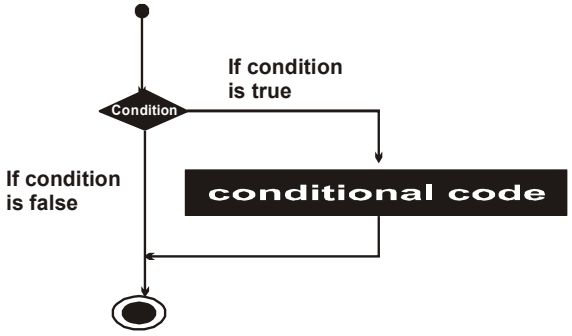
SYNTAX

The syntax of an if statement in C programming language is :

```
if ( boolen _ expression )
 {
/ * statement (s) will execute if the boolean expression is true * /
 }
```

- If the boolean expression evaluates to true, then the block of code inside the if statement will be executed.
- If boolean expression evaluates to false, then the first set of code after the end of the if statement (after the closing curly brace) will be executed.

FLOW DIAGRAM



EXAMPLE :

```
#include<stdio.h>
#include<conio.h>
int main ()
 {
int a = 10; /* local variable definition */
if (a <20) /* check the boolean condition using if statement */
 {
printf (“a is less than 20\n”);   /* if condition is true print the following */
 }
printf (“value of a is : %d\n”,a);
getch ();
return 0;
 }
```

OUTPUT

```
a is less than 20.
value of a is : 10
```

4.4.2 if ..... else statement

An if statement can be followed by an optional else statement, which executes when the boolean expression is false.

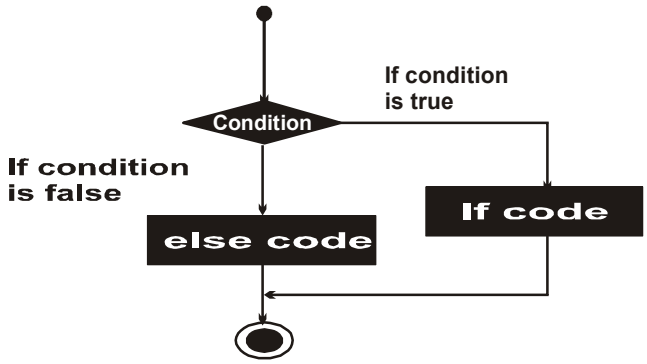
SYNTAX :

The syntax of an if .... else statement in C programming language is:

```
if (boolean _ expression)
 {
 /* statement (s) will execute if the boolean expression is true */
 }
else
{
/* statement (s) will execute if the boolean expression is false */
}
```

- If the boolean expression evaluates to true, then the if block of code will be executed, otherwise else block of code will be executed.
- C programming language assumes any non-zero and non-null values as true, and if it is either zero or null, then it is assumed as false value.

FLOW DIAGRAM



EXAMPLE :

```
#include<stdio.h>
#include<conio.h>
int main ()
 {
/ * local variable definition */
int a = 100;
 /* check the boolean condition * /
if ( a <20)
 {
 /* if condition is true then print the following */
printf ( " a is less than 20\n");
 }
else
 {
 /* if condition is false then print the following * /
printf ("a is not less than 20 \n");
```

```
}
printf ("value of a is : %d \n ", a );
getch ();
return 0;
}
```

OUTPUT

```
a is not less than 20;
value of a is : 100
```

4.4.3 else if ..... statement

An if statemenrt can be followed by an optional else if ....else statement, which is very useful to test various conditions using single if .... else if statement.

When using if, else if , else statements there are few points to keep in mind :

- An if can have zero or one else's and it must come after any else if's.
- An if can have zero to many else if's and they must come before the else.
- Once an else if succeeds, none of the remaining else if's or else will be tested.

SYNTAX :

The syntax of an else if .... else statement in C programming language is:

```
if ( boolean _ expression 1 )
{
/ * executes when the boolean expression 1 is true * /
}
else if ( boolean _ expression 2)
{
/ * executes when the boolean expression 2 is true */
}
else if ( boolean _ expression 3)
{
/ * executes when the boolean expression 3 is true */
}
```

```
else
 {
 /* executes when the none of the above condition is true */
 }
```

EXAMPLE :

```
#include<stdio.h>
int main ()
 {
 /* local variable definition */
int a = 100;
 / * check the boolean condition */
if ( a = =10)
{
/* if conditions is true print the following */
printf (“Value of a is 10\n”);
}
else if ( a = = 20 )
{
/* if else if condition is true */
printf (“Value of a is 20\n”);
}
ele if (a = = 30)
{
/* if else if condition is true */
printf (“Value of a is 30\n”);
}
else
{
/* if none of the conditions is true */
printf (“None of the values is matching\n”);
```

```
}
printf (“Exact value of a is : %d\n”,a);
return 0;
}
```

OUTPUT

```
None of the values is matching
Exact value of a is : 100
```

4.4.4 Nested if ..... else statement

It is always legal in C programming to nest if-else statements, which means you can use one if or else if statement inside another if or else if statement(s).

SYNTAX :

The syntax for a nested if statement is as follows :

```
if (boolean_expression 1)
{
(* Executes when the boolean expression 1 is true */
if (boolen_expression 2)
{
/* Executes when the boolean expression 2 is also true after expression 1 is true */
}
}
```

You can nest else if....else in the similar way as you have nested is statement.

EXAMPLE :

```
#include<stdio.h>
int main ()
{
/* local variable definition */
int a = 100;
int b = 200;
/* check the boolean condition */
```

```
if (a = = 100)
{
/* condition is true then check the following */
if (b = = 200)
{
/* if condition is true then print the following */
printf ("Value of a is 100 and b is 200\n");
}
}
printf ("Exact value of a is : %d\n", a);
printf ("Exact value of b is : %d\n", b);
return 0;
}
```

OUTPUT

```
Value of a is 100 and b is 200
Exact value of a is : 100
Exact value of b is : 200
```

4.4.5 Switch Statement

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each switch case.

SYNTAX :

The syntax for a switch statement in C programming language is as follows :

```
switch (expression)
{
case constant-expression1 :
                    statement(s);
                    break;/* optional */
case constant-expression2 :
                    statement(s);
                    break;/* optional */
```

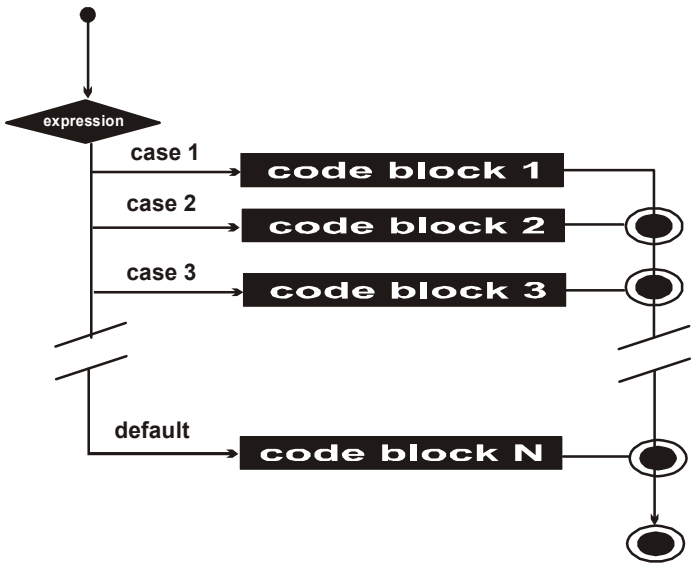
```
/* you can have any number of case statements */
default :                    /* Optional */
                             statement(s);
}
```

The following rules apply to a switch statement :

- The expression used in a switch statement must have an integral or enumerated type.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The constant-expression for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.
- When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a **break**. If no **break** appears, the flow of control will fall through to subsequent cases until a break is reached.
- A **switch** statement can have an optional **default** case, which must appear at the end of the switch.

The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

FLOW DIAGRAM :



EXAMPLE :

```
#include<stdio.h>
int main ()
{
/* local variable definition */
char grade = 'B';
switch (grade)
{
case 'A' :
        printf ("Excellent!\n");
        break;
case 'B' :
case 'C' :
        printf("Well done\n");
        break;
case 'D' :
        printf ("You passed\n");
        break;
case 'F' :
        printf ("Better try again\n");
        break;
default :
        printf("invalid grade\n");
}
printf("Your grade is %c\n", grade);
return 0;
}
```

OUTPUT

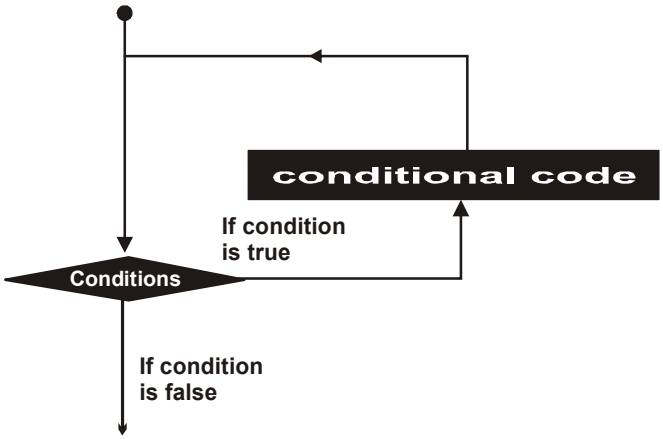
```
Well done
Your grade is B
```

4.5 Repetition or Loop Control Statement

There may be a situation, when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages.

FLOW DIAGRAM



C programming language provides the following types of loop to handle looping requirements.

Loop Type	Description
while loop	Repeats a statement or group of statement while a given condition is true. It tests the condition before executing the loop body.
for loop	Execute a sequence of statement multiple times and abbreviates the code that manges the loop variable.
do.... while loop	Like a while statement, except that it tests the condition at the end of the loop body.
nested loops	You can use one or more loop inside any another while, for or do... while loop.

4.5.1 While Loop

A while loop statement in C programming language repeatedly executes a statement as long as a given condition is true.

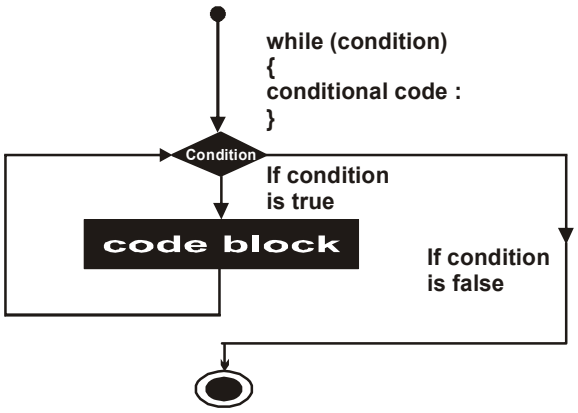
SYNTAX :

The syntax of a while loop in C programming language is :

```
while ( condition )
{
        statement (s);
}
```

- Here, statement (s) may be a single statement or a block of statements.
- The condition may be any expression, and true is any nonzero value.
- The loop iterates while the condition is true.
- When the condition becomes false, program control passes to the line immediately following the loop.

FLOW DIAGRAM



Here, key point of the while loop is that the loop might not ever run.

When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

EXAMPLE :

```
#include<stdio.h>
int main ()
{
/ * local variable definition * /
int a = 10;
/ * while loop execution * /
```

```
while ( a <20)
 {
        printf ("value of a: %d\n", a);
        a++;
}
return 0;
 }
```

OUTPUT

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

4.5.2 DO....WHILE LOOP

Unlike while loop which test the loop condition at the top of the loop, the do... while loop in C programming language checks its condition at the bottom of the loop.

A do... while loop is similar to a while loop, except that a do ..... while loop is guaranted to execute at least one time.

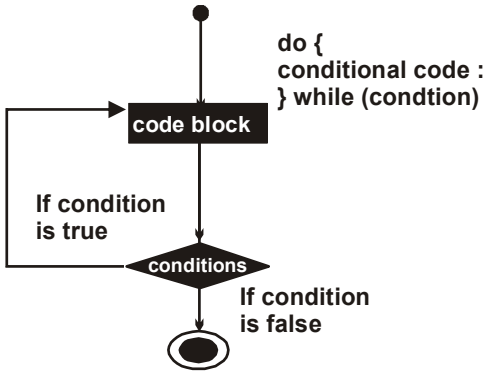
SYNTAX :

The syntax of a do.... while loop in C programming language is :

```
do
 {
statement (s);
 }
while ( condition );
```

- The conditional expression appears at the end of the loop, so the statement (s) in the loop execute once before the condition is tested.
- If the condition is true, the flow of control jumps back up to do, and the statement (s) in the loop execute again.
- This process repeats until the given condition becomes false.

FLOW DIAGRAM



EXAMPLE :

```
#include<stdio.h>
#include<conio.h>
int main ()
{
/ * local variable definition */
int a = 10;
/ * do loop execution */
do
{
        printf (“value of a: %d \n”, a);
        a = a + 1;
}while ( a <20);
getch ();
return 0;
}
```

OUTPUT

```
value of a : 10
value of a : 11
value of a : 12
value of a : 13
value of a : 14
value of a : 15
value of a : 16
value of a : 17
value of a : 18
value of a : 19
```

4.5.3 For Loop

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

SYNTAX :

The syntax of a for loop in C programming language is:

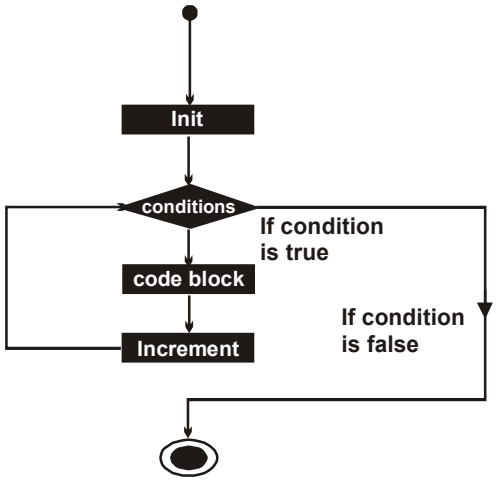
```
for ( init; condition; increment )
 {
statement(s);
 }
```

Here is the flow of control in a for loop :

1. The init step is executed first, and only once. This step allows you to declare and initialize any loop control variables.
2. Next, the condition is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the for loop.
3. After the body of the for loop executes, the flow of control jumps back up to the increment statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.

4. The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the for loop terminates.

FLOW DIAGRAM



Various forms of loop statement can be :

(a) for (; condition ; increment/decrement)

body;

A blank first statement will mean no initialization.

(b) for (initialization ; condition ;)

body ;

A blank llast statement will mean no running increment / decrement.

(c) for (initialization ;; increment / decrement)

body;

A blank second conditional statement means no test condiiotn to control the exit from the loop. so, in the absence of second statement, it is required to test the condition inside the loop otherwise it results in an infinite loop where the control never exist from the loop.

(d) for (;; increment / decrement)

body;

initialization is required to be done before the loop and test condition is checked inside the loop.

(e) for (initialization ;;)

body;

Test condition and control variable increment / decrement is to be done inside the body of the loop.

(f) for (;condition;)

body;

Initialization is required to be done before the loop and control variable increment / decrement is to be done inside the body of the loop.

(g) for (;;;)

body;

Initialization is required to be done before the loop, test condition and control variable. increment / decrement is to be done inside the body of the loop.

EXAMPLE :

```
#include<stdio.h>
int main ()
 {
 / * for loop execution*/
 for ( int a = 10; a <20; a = a +1)
{
printf (“value of a : %d \n”, a);
 }
return 0;
```

OUTPUT

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

EXAMPLE : FIBONACCI SERIES PROGRAM USING FOR LOOP

```
#include<stdio.h>
#include<conio.h>
int main ()
 {
int n, first = 0, second = 1, next, c;
printf (“Enter the number of terms \n”);
scanf (“%d”, &n);
printf (“ First %d terms of Fibonacci series are :-\n”,n);
for ( c = 0 ; c < n ; c++)
 {
        if ( c <=1)
        next = c;
        else
 {
next = first + second;
first = second;
second = next;
}
printf (“%d\n”, next);
 }
getch ();
return 0;
 }
```

OUTPUT

```
Enter the number of terms
5
First 5 terms of Fibonacci series are
0
1
1
2
```

PROGRAM ILLUSTRATION

Using the code above you can print as many numbers of terms of series as desired. Numbers of Fibonacci sequence are known as Fibonacci numbers. First few numbers of series are 0, 1, 1, 2, 3, 5, 8 etc, Except firsr two terms in sequence every other terms is the sum of two previous terms, for example 8 = 3 + 5 (addition of 3, 5).

4.6 Nested Loop

C programming language allows to use one loop inside another loop.

Following section shows few example to illustrate the concept.

Nested Switch Statement

It is possible to have a switch as part of the statement sequence of an outer switch. Even if the case constants of the inner and outer switch contain common values, no conflicts will arise.

SYNTAX :

The syntax for a nested switch statement is as follows:

```
switch (ch1)
{
        case 'A'
        printf ("This A is part of outer switch");
switch (ch2) /* switch (ch2) inside case 'A' of switch (ch1) */
{
        case 'A'
        printf ("This A is part of inner switch");
        break;
case 'B': / * case code */
}
break; /* case 'A' break of switch (ch1) */
case 'B': / * case code for switch (ch1) */
}
```

EXAMPLE :

```
#include<stdio.h>
int main ()
 {
        /* local variable definition */
        int a = 100;
        int b = 200;
        switch(a)
{
        case 100:
        printf ("This is part of outer switch \n", a );
        switch (b)
{
        case 200 :
        printf ("This is part of inner switch \n", a );
 }
}
        printf ("Exact value of a is : %d\n", a);
        printf ("Exact value of b is : %d\n", b);
        return 0;
}
```

OUTPUT

```
This is part of outer switch
This is part of inner switch
Exact value of a is : 100
Exact value of b is : 200
```

Nested For Loop

SYNTAX :

The syntax for a nested for loop statement in C is as follows :

```
for (init; condition; increment )
 {
for (init; condition; increment )
 {
        statement(s);
 }
        statement(s);
}
```

EXAMPLE : Write A C Program to Display the given Pattern

```
#include<stdio.h>
#include<conio.h>
int main ()
 {
int row , c, n, temp;
printf (Enter the number of rows in pyramid of stars “);
 scanf (“%d”, &n);
temp = n;
for (row = 1 ; row <= n ; row++)
{
for ( c =1 ; c < temp ; c++)
        printf (“  “);
temp--;
for ( c = 1 ; c <=2*row - 1 ; c++)
        printf (“*”)
printf (“\n”);
}
return 0;
 }
```

OUTPUT

```
*
***
*****
*******
*********
```

Nested while loop

The syntax for a nested while loop statement in C programming language is as follows:

```
while ( condition)
{
while ( condition)
{
statement (s);
 }
statement(s);
 }
```

EXAMPLE :

```
#include<stdio.h>
int main ()
 {
int j = 0
int j = 0;
while (i < 10)
 {
printf("i:%d\n", i);
while (j < 10)
 {
printf("j: %d\n",j);
j ++;
```

```
 }
i++;
 }
}
```

OUTPUT

```
i:0
j:0
j:1
j:2
j:3
j:4
j:5
j:6
j:7
j:8
j:9
i:1
i:2
i:3
i:4
i:5
i:6
i:7
i:8
i:9
```

Nested do..... while loop

The syntax for a **nested do ..... while loop** statement in C programming language is as follows :

```
do
 {
```

```
statement(s);
do
{
statement(s);
 } while ( condition );
 } while ( condition );
```

A final note on loop nesting is that you can put any type of loop inside of any other type of loop. For example, a for loop can be inside a while loop or vice versa.

EXAMPLE :

The following program uses a nested for loop to find the prime numbers from 2 to 100:

```
#include<stdio.h>
int main ()
 {
 /* local variable definition */
 int i, j;
for (i = 2; i<100; i++) /*outer loop */
{
for(j =2; j <=(i/j); j++)
 /* inner loop, always terminate before outer loop */
if (!(i%j))
break; // if factor found, not prime
if(i>(i/j))
printf (“%d is prime \n”, i);
 }
return 0;
 }
```

OUTPUT

```
2 is prime
3 is prime
```

5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
29 is prime
31 is prime
37 is prime
41 is prime
43 is prime
47 is prime
53 is prime
59 is prime
61 is prime
67 is prime
71 is prime
73 is prime
79 is prime
83 is prime
89 is prime
97 is prime

4.7 Loop Control Statements

Loop control statement change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

C supports the following control statements.

Control statement	Description
break statement	Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch.

Control statement	Description
Continue statement	Cause the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
goto statement	Transfer control to the labeled statement. Though it is not advised to use goto statement in your program.

4.7.1 BREAK STATEMENT

The break statement in C programming language has the following two usages.

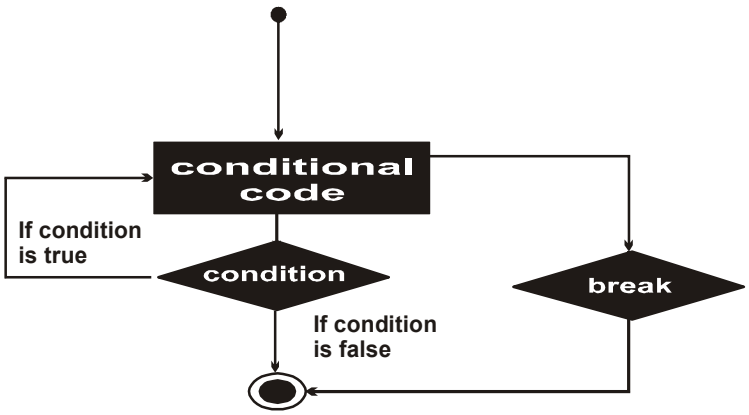
1. When the break statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.
2. It can be used to terminate a case in the switch statement.
3. If you are using nested loops (i.e., one loop inside another loop), the break statement will stop the execution of the innermost loop and start executing the next line of code after the block.

SYNTAX :

The syntax for a **break** statement in C is as follows:

```
break;
```

FLOW DIAGRAM



EXAMPLE :

```
#include<stdio.h>
int main ()
{
/* local variable definition */
```

```
int a = 10;
/ * while loop execution */
while ( a <20)
{
        printf ("value of a: %d\n ", a);
         a++;
if (a > 15)
{
/* terminate the loop using break statement */
break;
}
}
return 0;
}
```

OUTPUT

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
```

EXAMPLE : Write A Program to Calculate the First Smallest Divisor of A Number

```
/ * program to calculate smallest divisor of a number */
#include<stdio.h>
int main ()
{
int div, num, i;
printf ("Enter any number:/n");
scanf ("%d", &num);
```

```
for (i = 2;i<=num;++i)
 {
if (num % i) == 0)
 {
printf (“Smallest divisor for number %d is  %d “, num,i);
break;
 }
 }
 }
```

OUTPUT

```
Enter any number :
 9
Smallest divisor for number 9 is 3
```

PROGRAM ILLUSTRATION

In the above program. we divide the input number with the integer starting from 2 onwards, and print the smallest divisior as soon as remainder comes out to be zero. Since we need the first smallest divisior and not all divisior of a given number, so we jump out of the for loop using break statement without further going for the next iteration of for loop.

4.7.2 CONTINUE STATEMENT

The continue statement in C programming language works somewhat like the break statement . Instead of forcing termination, however continue forces the next iteration of the loop to take place, skipping any code in between.

For the for loop, **continue** statement causes the conditional test and increment portions of the loop to execute.

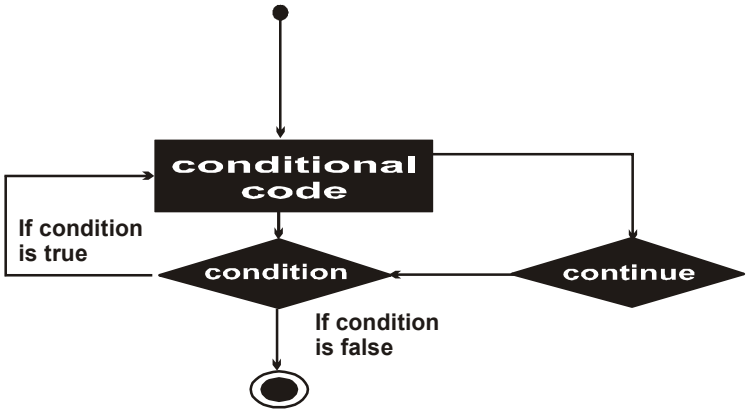
For the while and do.... while loops, continue statement causes the program control passes to the conditional tests.

SYNTAX :

The syntax for a continue statement in C is as follows :

```
continue;
```

FLOW DIAGRAM



EXAMPLE :

```
#include<stdio.h>
int main ()
 {
 /* local variable definition */
int a = 10;
/ * do loop execution */
do
{
if( a = = 15)
{
 / *skip the iteration */
a = a + 1;
continue;
 }
printf("value of a; %d\n", a);
 a++;
 } while( a < 20 );
return 0;
 }
```

OUTPUT

value of a : 10
value of a : 11
value of a : 12
value of a : 13
value of a : 14
value of a : 16
value of a : 17
value of a : 18
value of a : 19

4.7.3 GO TO STATEMENT

A **goto** statement in C programming language provides an unconditional jump from the goto to a labeled statement in the same function.

NOTE : Use of goto statement is highly discouraged in any programming language because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify. Any program that uses a goto can be rewritten so that it doesn't need the goto.

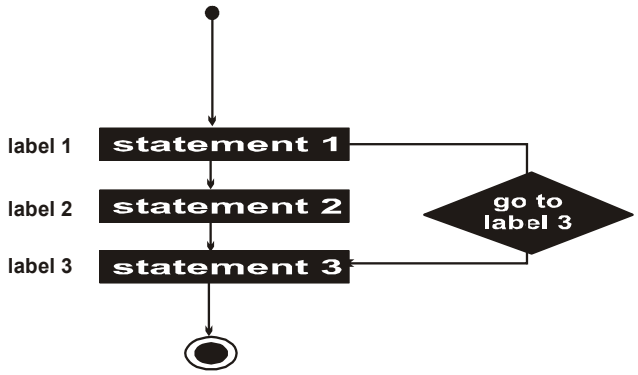
SYNTAX :

The syntax for a goto statement in C is as follows:

```
goto label;
....
label : statement;
```

Here **label** can be any palin text except C keyword and it can be set anywhere in the C program above or below to goto statement.

FLOW DIAGRAM



EXAMPLE :

```
#include<stdio.h>
int main ()
 {
 /* local variable definition */
 int a = 10;
 / * do loop execution */
LOOP:
do
{
if( a = = 15)
{
 / *skip the iteration */
a = a + 1;
goto LOOP;
 }
printf("value of a; %d\n", a);
a++;
} while( a < 20 );
return 0;
 }
```

OUTPUT

```
value of a : 10
value of a : 11
value of a : 12
value of a : 13
value of a : 14
value of a : 16
value of a : 17
value of a : 18
value of a : 19
```

EXAMPLE :

Write a program to print first 10 even numbers

```
/ * program to print 10 even numbers */
#include<stdio.h>
main ()
{
int i = 2;
while (1)
{
printf("%d ",i);
i = i + 2;
if (i>20)
goto outside;
}
outside : printf("FINISHED")
}
```

OUTPUT

2 4 6 8 10 12 14 16 18 20 FINISHED

PROGRAM ILLUSTRATION

This program initialize the value of i with 2. The printf statement prints the value of i once it enters the while loop. After display of i value, its value get increment by 2. A condition checking is done. If its value less than 20 it continue inside while loop with displaying and incrementing its value by 2 but once it value becomes 20 it goto label outside.

4.8 The Infinite Loop

A loop becomes infinite loop if a condition never becomes false. The for loop is traditionally used for this purpose. Since none of the three expression that the for loop are required, you can make an endless loop by leaving the conditional expression empty.

```
#include<stdoi.h>
int main ()
{
for ( ; ; )
{  printf ("This loop will run forever. \n"); }
return 0;
```

```
}
```

When the conditional expression is absent, it is assumed to be true. you may have an initialization and increment expression, but C programming more commonly use the for(; ;) construct to signify an infinite loop.

NOTE : You can terminate an infinite loop by pressing Ctrl + C keys.

4.9 Summary

During execution of process it may require to repeat execution of a part of code more than once depending upon the requirements of upon the decisions. C provides control and looping statements. In this unit we had learnt about different looping statements provided by C language namely if, if .... else, while do .... while and for.

Using break statement, we can get out of a loop even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end. The continue statement causes the program to skip the rest of the loop in the present iteration as if the end of the statement block would have reached, causing it to jump to the following iteration.

Using the goto statement, we can make an absolute jump to another point in the program. The destination point is identified by a label, which is then used as argument for the goto instruction. A label is made of a valid identifier followed by a colon (:)

4.10 Question for Exercise

1. What are decision control statement? Explain in detail.
2. Compare the use of if-else construct with that of ternary operator.
3. In what situation will you prefer to use for, while and do-while loop?
4. Write a program which demonstrates the use of goto, break and continue statements.
5. Write a program that displays all the numbers from 1-100 that are divisible by 7.
6. Write a program using switch case to dispaly a menu that offers five options: read three numbers, calculate total, calculate avarage, dispaly the smallest and the largest value.
7. Write a program to enter a decimal number. Calculate and dislay the binary equivalent.
8. Write a program to enter a number and then calculate the sum of its digit.
9. Write a program to print the reverse of number and check whether the entered number is palindrome or not.
10. Write a program to generaate the following pattern given below:

1

1 2

1 2 3

1 2 3 4

11. Compare while and do while loop using example.
12. Write a menu driven program using switch case that offers following option.
    1. Sum of first 10 natural numbers.
    2. Sum of square of first 10 even numbers.
    3. Sum of factorial of first five number (1! + 2! + 3! + 4! + 5!)
14. Write a program to accept marks of five paper and calculate it percentage. On the basic of percentage the following grade should be displayed.
    i) if per >=75, Grade A.
    ii) if per >=60 and <75, Grade B
    iii) if per>=40 and < 60, Grade C
    iv) if per < 40, FAIL
15. Write a program to find the greatest among three numbers.
16. Write a program to check whether the entered year is leap year or not.

4.11 Suggested Readings

1. Computer Science: A Structed programming Approach Using C, Second Edition, Behrouz A. Forouzan, Richard F. Gilberg, Brooks/Cole Thomas Learning, 2001.
2. The C programming language, Brain W. Kernighan, Dennis M. Ritchie, PHI
3. C, The complete Reference, Fourth Edition, Herbdert Schildt, Tata McGraw Hill,
4. Programming with C, Second Edition, Byron Gottfried, Tata Mc Graw Hill, 2003.
5. The C Primer, Leslie Hancock, Morris Krieger, Mc Graw Hill, 1983.

Reference Links

- www.tutorialspoint.com / cprogramming
- https: / / tecnoesis. wordpress.com
- www.worldbestlearningcenter.com
- www.notes99.com

●●●