MANGALAYATAN UNIVERSITY
Learn Today to Lead Tomorrow
NAAC GRADE A+ Accredited University

BACHELOR OF COMPUTER APPLICATION (BCA)

CSB-1112 (Problem Solving Using C)

Block 3: Advanced C Programming Concepts

Unit 9: Advanced Control Structures

9.0 Introduction

In this unit a program consists of a number of statements to be executed by the computer. Not many of the programs execute all their statements in sequential order from beginning to end as they appear within the program. C program may require that a logical test be carried out at some particular point within the program. One of the several possible actions will be carried out, depending on the outcome of the logical test. This is called Branching. In the Selection process, a set of statements will be selected for execution, among the several sets available. Suppose, if there is a need of a group of statements to be executed repeatedly until some logical condition is satisfied, then looping is required in the program. These can be carried out using various control statements.

These Control statements determine the “flow of control” in a program and enable us to specify the order in which the various instructions in a program are to be executed by the computer. Normally, high level procedural programming languages require three basic control statements:

- Sequence instruction
- Selection/decision instruction
- Repetition or Loop instruction

Sequence instruction means executing one instruction after another, in the order in which they occur in the source file. This is usually built into the language as a default action, as it is with C. If an instruction is not a control statement, then the next instruction to be executed will simply be the next one in sequence.

Selection means executing different sections of code depending on a specific condition or the value of a variable. This allows a program to take different courses of action depending on different conditions. C provides three selection structures.

- if
- if…else
- switch

Repetition/Looping means executing the same section of code more than once. A section of code may either be executed a fixed number of times, or while some condition is true. C provides three looping statements:

- while
- do…while
- for

This unit introduces you the decision and loop control statements that are available in C programming language along with some of the example programs.

This unit will explain to you the expressions and operators of language C.

9.1 Objectives

After completing this unit, you will be able to:

- Work with different control statements;
- Know the appropriate use of the various control statements in programming;
- Transfer the control from within the loops;
- Use the goto, break and continue statements in the programs; and
- Write programs using branching, looping statements.

9.2 Decision Control Statements

In a C program, a decision causes a one-time jump to a different part of the program, depending on the value of an expression. Decisions in C can be made in several ways. The most important is with the if...else statement, which chooses between two alternatives. This statement can be used without the else, as a simple if statement. Another decision control statement, switch, creates branches for multiple alternative sections of code, depending on the value of a single variable.

9.2.1 The if Statement

It is used to execute an instruction or sequence/block of instructions only if a condition is fulfilled. In if statements, expression is evaluated first and then, depending on whether the value of the expression (relation or condition) is “true” or “false”, it transfers the control to a particular statement or a group of statements.

Different forms of implementation if-statement are:

- Simple if statement

- If-else statement
- Nested if-else statement
- Else if statement

Simple if statement It is used to execute an instruction or block of instructions only if a condition is fulfilled.

The syntax is as follows:

if (condition)

statement;

where condition is the expression that is to be evaluated. If this condition is **true**, statement is executed. If it is **false**, statement is ignored (not executed) and the program continues on the next instruction after the conditional statement.

If we want more than one statement to be executed, then we can specify a block of statements within the curly bracets { }.

The syntax is as follows:

if (condition)

{

block of statements;

}

Example:

Write a program to calculate the net salary of an employee, if a tax of 15% is levied on his gross-salary if it exceeds Rs. 10,000/- per month.

```
/*Program to calculate the net salary of an employee */
#include<stdio.h>
main( )
{
float gross_salary, net_salary;
printf("Enter gross salary of an employee\n");
scanf("%f ",&gross_salary );
if(gross_salary<10000)
net_salary = gross_salary;
net_salary=gross_salary - 0.15*gross_salary;
printf("\nNet salary is Rs.%.2f\n", net_salary);
}
```

9.2.2 The if_else Statement

If…else statement is used when a different sequence of instructions is to be executed depending on the logical value (True / False) of the condition evaluated.

Its form used in conjunction with if and the syntax is as follows:

```
if (condition)
Statement _1;
```

```
else
Statement_ 2;
statement_3;
Or
if (condition)
{
Statements_1_Block;
}
else
{
Statements_2_Block;
}
Statements _3_Block;
```

If the condition is **true**, then the sequence of statements (Statements_1_Block) executes; otherwise the Statements_2_Block following the else part of if-else statement will get executed. In both the cases, the control is then transferred to Statements_3 to follow sequential execution of the program.

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int no;
clrscr();
printf("\n Enter Number :");
scanf("%d",&no);
if(no>0)
{
printf("\n\n Number is greater than 0 !");
}
else
{
if(no==0)
{
printf("\n\n It is 0 !"); }
else
{
printf("Number is less than 0 !");
}
}
getch();
}
```

Output:

```
Enter Number: 0
        It is 0!
```

9.2.3 The switch Statement

This is a multiple or multiway branching decision making statement.

When we use nested if-else statement to check more than 1 conditions then the complexity of a program increases in case of a lot of conditions. Thus, the program is difficult to read and maintain. So to overcome this problem, C provides 'switch case'.

Switch case checks the value of an expression against a case values, if condition matches the case values then the control is transferred to that point.

Syntax:

```
switch(expression)
{
case expr1:
statements;
break;
case expr2:
statements;
break;
----------------------------
----------------------------
----------------------------
case exprn:
statements;
break;
default:
statements;
}
```

Switch case, break are keywords.

expr1, expr2 are known as 'case labels.'

Statements inside case expression need not to be closed in braces.

break statement causes an exit from switch statement.

default case is optional case. When neither any match found, it executes.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int no;
clrscr();
printf("\n Enter any number from 1 to 3 :");
scanf("%d",&no);
switch(no)
{
case 1:
printf("\n\n It is 1 !");
break;
case 2:
printf("\n\n It is 2 !");
break;
case 3:
printf("\n\n It is 3 !");
```

```
break;
default:
printf("\n\n Invalid number !");
}
getch();
}
```

Output:

```
Enter any number from 1 to 3 : 3
It is 3 !
```

a) Rules for declaring switch case:

- The case label should be integer or character constant.
- Each compound statement of a switch case should contain break statement to exit from case.
- Case labels must end with (:) colon.

b) Advantages of switch case:

- Easy to use.
- Easy to find out errors.
- Debugging is made easy in switch case.
- Complexity of a program is minimized.

9.3 Loop Control Statements

Loop control statements are used when a section of code may either be executed a fixed number of times, or while some condition is true. C gives you a choice of three types of loop statements, while, do- while and for.

- The while loop keeps repeating an action until an associated condition returns false. This is useful where the programmer does not know in advance how many times the loop will be traversed.
- The do while loop is similar, but the condition is checked after the loop body is executed. This ensures that the loop body is run at least once.
- The for loop is frequently used, usually where the loop will be traversed a fixed number of times.

9.3.1 The while Loop

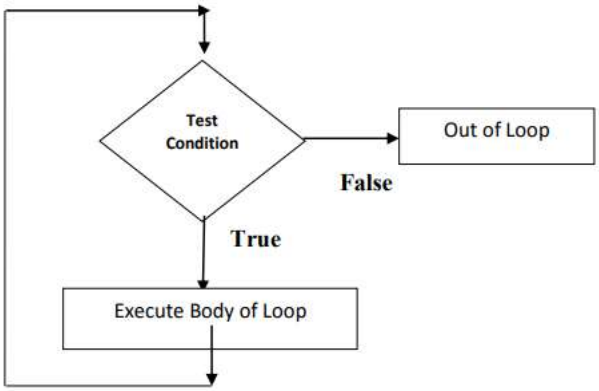
When in a program a single statement or a certain group of statements are to be executed repeatedly depending upon certain test condition, then while statement is used.

The syntax is as follows:

```
while (test condition)
{
body_of_the_loop;
}
```

Here, test condition is an expression that controls how long the loop keeps running. Body of the loop is a statement or group of statements enclosed in braces and are repeatedly executed till the value of test condition evaluates to true. As soon as the condition evaluates to false, the control jumps to the first statement following the while statement. If condition initially itself is false, the body of the loop will never

be executed. While loop is sometimes called as entry-control loop, as it controls the execution of the body of the loop depending upon the value of the test condition.



Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a;
clrscr();
a=1;
while(a<=5)
{
printf("MCMT \t");
a+=1                    // i.e. a = a + 1
}
getch();
}
```

Output:

MCMT MCMT MCMT MCMT MCMT

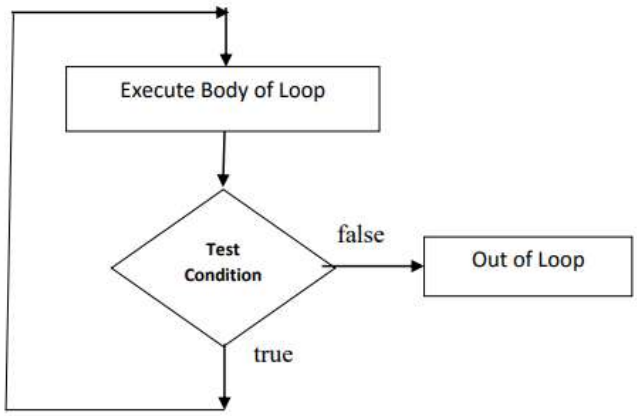
9.3.2 The do-while Statement

There is another loop control structure which is very similar to the while statement – called as the **do.. while** statement. The only difference is that the expression which determines whether to carry on looping is evaluated at the end of each loop.

The syntax is:

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

In do-while loop, the body of loop is executed at least once before the condition is evaluated. Then the loop repeats body as long as condition is true. However, in while loop, the statement doesn't execute the body of the loop even once, if condition is false. That is why do-while loop is also called exit-control loop.



Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a;
clrscr();
a=1;
do
{
printf("MCMT\t");      // 5 times
a+=1;                  // i.e. a = a + 1
}
while(a<=5);
getch();
}
```

Output:

MCMT MCMT MCMT MCMT MCMT

Infinite loop:

A looping process, in general, includes the following four steps:

- Setting of a counter.
- Execution of the statements in the loop.
- Testing of a condition for loop execution.
- Incrementing the counter.

The test condition eventually transfers the control out of the loop. In case, due1 to some reasons, if-it does not do so, the control sets up an infinite loop and the loop body is executed over and over again. Such infinite loops should be avoided. Ctrl + C or Ctrl + Break are used to terminate the program caught in an infinite loop.

Example:

```
#include<stdio.h>
void main()
{
```

```
int i=1;
while(i<=10)
{
Printf(“ i= %d\n”,i);
}
}
```

This program will never terminate as variable i will always be less than 10. To get the loop terminated, an increment operation (i + +) will be required in the loop.

9.3.3 The for Loop

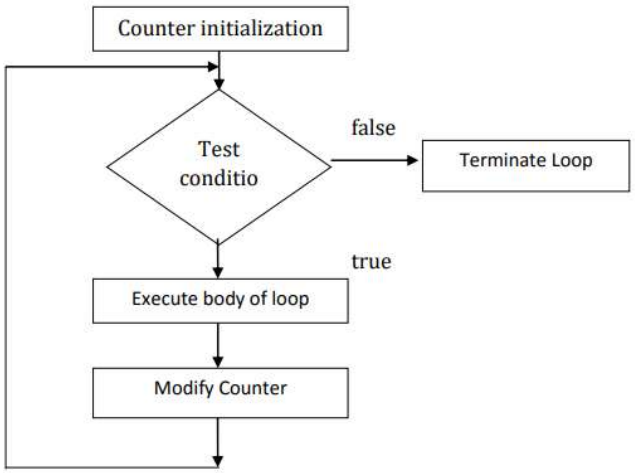
for statement makes it more convenient to count iterations of a loop and works well where the number of iterations of the loop is known before the loop is entered. **The syntax is:**

```
for (initialization; test condition; increment or decrement)
{
Statement(s);
}
```

The main purpose is to repeat statement while condition remains true, like the while loop. But in addition, for provides places to specify an initialization instruction and an increment or decrement of the control variable instruction. So this loop is specially designed to perform a repetitive action with a counter.

The for loop as works in the following manner:

1. Initialization is executed. Generally it is an initial value setting for a counter variable. This is executed only once.
2. Condition is checked, if it is true the loop continues, otherwise the loop finishes and statement is skipped.
3. Statement(s) is/are executed. As usual, it can be either a single instruction or a block of instructions enclosed within curly brackets { }.
4. Finally, whatever is specified in the increment or decrement of the control variable field is executed and the loop gets back to step 2.



Features:

- More concise
- Easy to use
- Highly flexible
- More than one variable can be initialized.
- More than one increments can be applied.
- More than two conditions can be used.

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a;
clrscr();
for(i=0; i<5; i++);
{
Printf("MCMT!\t");              // 5 times
}
getch();
}
```

Output:

MCMT MCMT MCMT MCMT MCMT

More About for Loop:

The for loop in C has several capabilities that are not found in other loop constructs. More than one variable can be initialized at a time in the for statement.

The Statement

p = 1;

for (n = 0; n < 17; + + n)

can be rewritten as for

(p = 1, n = 0; n < 17; ++ n)

The increment section may also have more than one part as given in the following

Example:

```
for (n = 1, m = 50; n < = m; n = n+1, m = m-j)
{
p = m/n;
printf ("%d %d %d\n", n, m, p):
}
```

9.3.4 The Nested Loop

Loops within loops are called nested loops. An overview of nested while, for and do .. while loops is given below:

Nested while:

It is required when multiple conditions are to be tested.

The syntax is:

```
        while (condition 1)
{
        ----------
        while (condition 2)
{
        ----------
        while (condition n)
{
        ----------
}
}
}
```

Example:

Write a program to generate the following pattern:

```
1
1   2
1   2   3
1   2   3   4
```

```
/* Program to print the pattern */
#include<stdio.h>
main( )
{
int i,j;
for (i=1;i<=4;++i);
{
printf("%d\n",i);
for(j=1;j<=i;++j);
printf("%d\t",j);
}
}
```

Here, an inner for loop is written inside the outer for loop. For every value of i, j takes the value from 1 to i and then value of i is incremented and next iteration of outer loop starts ranging j value from 1 to i.

Example:

```
#include<stdio.h>
#include<conio.h>
main( )
{
Int i = 1, N;
Clrscr();
```

```
While(i<=5)
{
N=1;
While(N<=5)
Printf("%d", N);
}
printf("\n");
}
}
```

OUPUT:

```
1  2  3  4  5
1  2  3  4  5
1  2  3  4  5
1  2  3  4  5
1  2  3  4  5
```

Nested for:

It is used when multiple set of iterations are required.

The syntax is:

```
for ( ; ; )
{
    ----------
    ----------
    ----------
for ( ; ; )
{
    ----------
    ----------
for ( ; ; )
{
    ----------
    ----------
    ----------
}
}
}
```

Example:

```
#include
#include
main()
{
```

```
int i , N;
clrscr();
for( i=1 ; i<= 5 ; i++ );
{
for( N=1 ;N<= 5 ; N++ );
{
printf(“ %d ” , N);
}
printf( “ \n”);
}
}
```

Ouput:

```
1  2  3  4  5
1  2  3  4  5
1  2  3  4  5
1  2  3  4  5
1  2  3  4  5
```

9.4 The Goto Statement

C supports the goto statement to branch unconditionally from one point to another in the program. A goto statement breaks the normal sequential execution of the program. The goto requires a label in order to identify the place where the branch is to be made. A label is any valid variable name, and must be followed by a colon. The label is placed immediately before the statement where the control is to be transferred.

It is a well known as “**jumping statement**”. It is useful to provide branching within a loop. When the loops are deeply nested at that if an error occurs then it is difficult to get exited from such loops. Simple break statement cannot work here properly. In this case, goto statement is used.

The general forms of goto and label statements are:

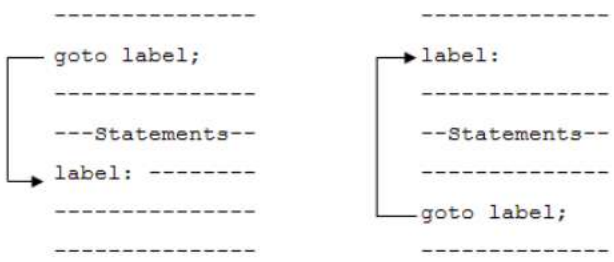
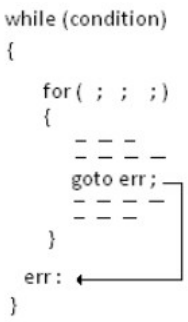


Figure:



Example:

```
#include<stdio.h>
#include<conio.h>
main()
{
int i=1, j;
clrscr();
while(i<=3)
{
for(j=1;
j<=3; j++)
{
printf(" I=%d \t J=%d \n", i , j);
if(j==2) goto stop;
}
i = i + 1;
}
getch();
}
```

Output:

```
I=1    J=1
I=1    J=2
```

9.5 The Break Statement

Sometimes, it is required to jump out of a loop irrespective of the conditional test value. **Break** statement is used inside any loop to allow the control jump to the immediate statement following the loop.

The syntax is: **break;**

When nested loops are used, then break jumps the control from the loop where it has been used. Break statement can be used inside any loop i.e., while, do-while, for and also in switch statement. Let us consider a program to illustrate break statement.

Example:

```
#include<stdio.h>
#include<conio.h>
main()
```

```
{
int i;
clrscr();
for(i=1; i<=20 ; i++);
{
if(i>5)
break;
printf("%d",i); // 5 times only
}
printf(" \nOut of loop");
getch();
}
```

Output:

1 2 3 4 5
Out of loop

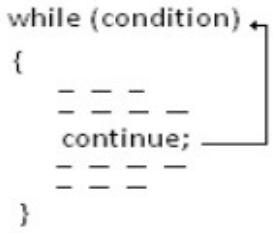
9.6 The Continue Statement

Sometimes, it is required to skip a part of a body of loop under specific conditions. So, C supports 'continue' statement to overcome this anomaly.

The working structure of 'continue' is similar as that of that break statement but difference is that it cannot terminate the loop. It causes the loop to be continued with next iteration after skipping statements in between. Continue statement simply skips statements and continues next iteration.

The syntax is: **continue;**

Figure:



Example:

```
#include<stdio.h>
#include<conio.h>
main()
{
int i;
clrscr();
for(i=1; i<=10; i++)
{
if( i>=6 && i <=8 )
continue;
```

```
printf("\t%d",i); // 6 to 8 is omitted
}
getch();
}
```

Output: 1 2 3 4 5 9 10

9.7 The Exit( ) Function

The exit( ) function is used to terminate the execution of 'C' program. It is a standard library function and uses header file stdlib.h.

The general form of exit( ) function is

exit (int status);

The difference between break and exit( ) is that former terminates the execution of loop in with it is written while exit( ) terminates the execution of program itself.

The status (in the general form of exit) is a value returned to the operation system after the termination. of the program.

The value zero "0" indicates that the termination is normal while value one "1" (Non-Zero) indicates different types of errors.

Example:

```
#include<stdio.h>
#include<conio.h>
main()
{
int i;
clrscr();
for(i=1; ; i++)
{
if(i>5)
exit(0);
printf("%d",i); // 5 times only
}
printf(" \nOut of loop"); // control will not reached here
getch();
}
```

Output:

1 2 3 4 5

9.8 Conclusion

A program is usually not limited to a linear sequence of instructions. During its process it may require to repeat execution of a part of code more than once depending upon the requirements or take decisions. For that purpose, C provides control and looping statements. In this unit, we had seen the different looping statements provided by C language namely *while, do…while and for*.

Using **break** statement, we can leave 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 before its natural 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. You should use this feature carefully since its execution ignores any type of nesting limitation. 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 (:).

9.9 Questions /Answers

1. **What is the purpose of the goto statement in C? Provide an example of its usage.**

Ans.1 The goto statement in C provides a way to jump to another part of the program. It is generally discouraged because it can make the code difficult to read and maintain, but it can be useful in certain scenarios where other control structures are cumbersome. Here's an example of its usage:

```
#include <stdio.h>
int main() {
   int num = 10;
   if (num == 10) {
      goto label;
   }
   printf("This will be skipped.\n");
   label:
   printf("This will be printed because of goto.\n");
   return 0;
}
```

2. **Explain the concept of a switch statement in C. How is it different from an if-else statement? Provide an example.**

Ans.2 The **switch** statement in C is a control structure used to execute one block of code from multiple options based on the value of an expression. It differs from **if-else** in that it is typically used when you have a single variable or expression that you want to compare against several constants. Here's an example:

```
#include <stdio.h>
int main() {
   int choice = 2;
   switch (choice) {
      case 1:
         printf("Choice is 1.\n");
         break;
      case 2:
         printf("Choice is 2.\n");
         break;
      case 3:
         printf("Choice is 3.\n");
```

```
            break;
        default:
            printf("Invalid choice.\n");
    }
    return 0;
}
```

3. Describe the do-while loop in C. How does it differ from a while loop? Provide an example.

Ans.3 The **do-while** loop is similar to the **while** loop but guarantees that the loop body will be executed at least once before the condition is tested. This is because the condition is checked after the loop body is executed. Here's an example:

```
#include <stdio.h>
int main() {
    int num = 1;
    do {
        printf("Number: %d\n", num);
        num++;
    } while (num <= 5);
    return 0;
}
```

4. What is a break statement and how is it used in loops and switch cases? Provide an example.

Ans.4 The **break** statement is used to exit from a loop or a **switch** case before it has completed its normal execution. In loops, it immediately terminates the loop and transfers control to the statement following the loop. In a **switch** statement, it ends the case and prevents the execution from falling through to the next case. Here's an example:

```
#include <stdio.h>
int main() {
    int i;
    for (i = 1; i <= 10; i++) {
        if (i == 5) {
            break;
        }
        printf("%d ", i);
    }
    printf("\n");
    switch (i) {
        case 1:
            printf("i is 1\n");
            break;
        case 5:
            printf("i is 5\n");
            break;
        default:
```

```
            printf("i is not 1 or 5\n");
        }
        return 0;
    }
```

5. **Explain the continue statement in C. How does it differ from the break statement? Provide an example.**

Ans.5 The **continue** statement skips the remaining code inside the loop for the current iteration and moves to the next iteration of the loop. Unlike **break**, which exits the loop, **continue** does not terminate the loop but just skips to the next iteration. Here's an example:

```
#include <stdio.h>
int main() {
    int i;
    for (i = 1; i <= 10; i++) {
        if (i % 2 == 0) {
            continue;
        }
        printf("%d ", i);
    }
    return 0;
}
```

9.10 References

- Kernighan, B. W., & Ritchie, D. M. (1988). The C Programming Language (2nd ed.). Prentice Hall.
- King, K. N. (2008). C Programming: A Modern Approach (2nd ed.). W. W. Norton & Company.
- Prata, S. (2014). C Primer Plus (6th ed.). Addison-Wesley Professional.
- Deitel, P., & Deitel, H. (2016). C How to Program (8th ed.). Pearson.

***