Control Structures in C
Part - I

Module – 1

Dear friends,

In Today's discussion we will cover an interesting, very useful and most frequently used feature of C Programming named control structures. You all know that a computer program contains a sequence of instructions and ensures that the instructions are executed in the same order in which they appear in the program.

For Example consider a program segment to find the average of two numbers

```
int num1,num2,sum;
float avg;
sum=num1+num2;
avg=sum/2;
```

Now let us understand the difference between Statements and Blocks.
As you know, in C, the semicolon is a statement terminator. Expressions like **count = 1** or **++i** or **printf(...)** becomes a ***statement*** when it is followed by a semicolon, as in

```
count = 1;
++i;
printf(...);
```

But ***Blocks*** are used to group declarations and statements together into a **compound statement**. Braces **{** and **}** are used for creating blocks and they are syntactically equivalent to a single statement. Any variable can be declared only at the beginning of the block.

For example

```
{ int a, b= 10;
  a= b*b;
  printf("%d %d",b, a);
}
```

But in practice, it may be necessary to make the sequence of the execution flow to be transferred from one part of the program to another part. This can be achieved with the help of **control (flow) structures** or **control constructs.**

As the name suggests the 'control instructions' determine the 'flow of control' in a program. The control-flow of a language specifies the order in which computations are performed. C supports the control structures classified under two main categories Conditional and Unconditional control structures.

Conditional control structures

In the case of conditional execution, the flow of execution may be transferred from one part of the program to another part based on the output of the conditional test carried out.

Under conditional construct we have two categories named as selective construct and loop construct.

Selective construct

In selective constructs, the statement is selected for execution based on the output of the conditional test given by the expression. C supports 4 selective constructs

including

1. Conditional expression
2. if – else
3. switch-case

Loop construct

Sometimes the execution of certain statements need to be repeated until a given condition is satisfied or it may have to be repeated for a known number of times. Such repetitions are carried out by using a loop structure. The loop construct is also known as iterative construct or a repetitive construct. C supports

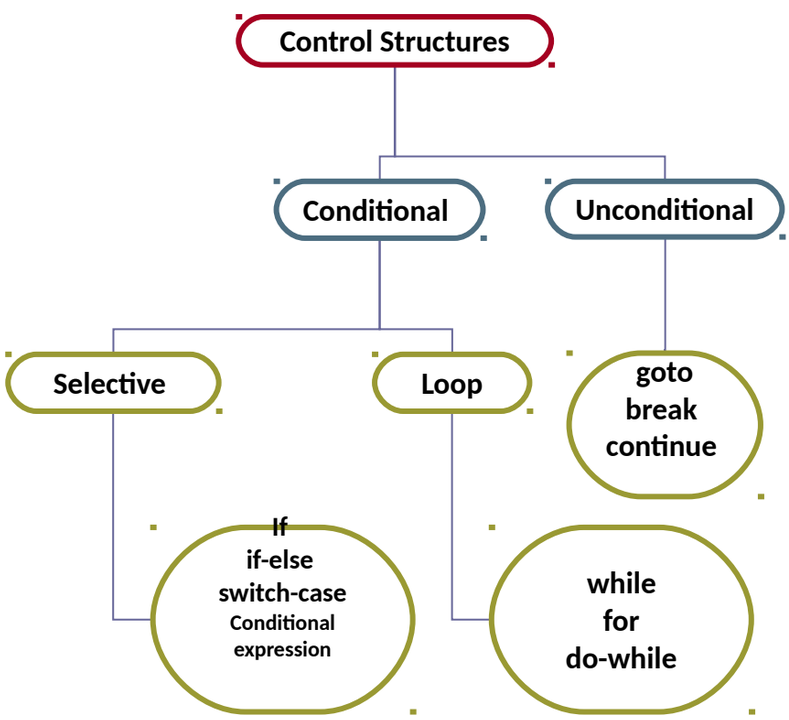
1. for
2. while
   and
3. do-while

loop constructs.

Unconditional control structures

If the flow of execution is transferred from one part of a program to another part without carrying out any conditional test, it is known as an unconditional execution. C supports break statement, continue Statement and goto unconditional control structures.

Classification of Control Structures



Now let us go through various control structures one by one in the order of simplicity with suitable examples.

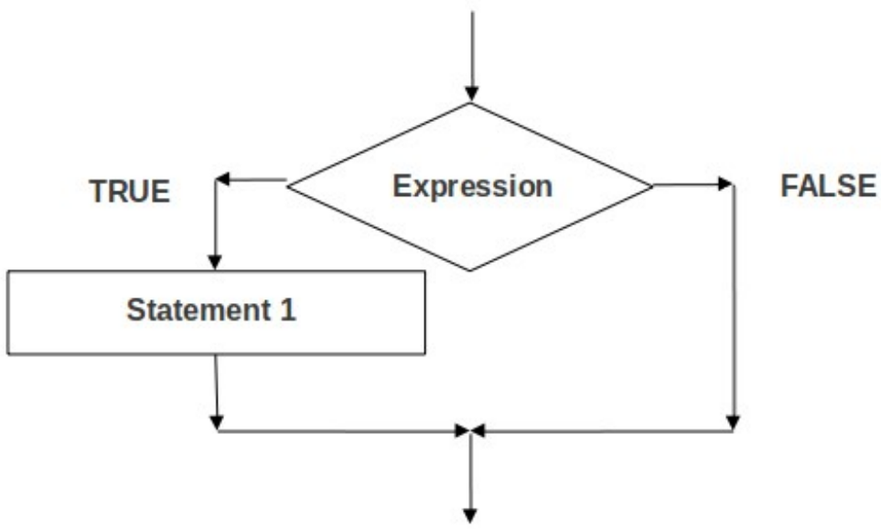
Module – 2

If CONSTRUCT

There may be situations where we have to perform different sets of actions depending on the circumstances. The if-else construct is used to select a specific statement based on the specific condition. It is a powerful decision-making statement and is used to control the flow of execution of statements. The general format is

if (test_expression)
{
True-block statements
}

it allows the computer to evaluate the expression first and then, depending on whether the value of the expression (relation or condition) is 'true' (non zero) or 'false' (zero),it transfers the control to a particular statement. This point of program has two paths to follow one for true condition and the other for false condition. You can see the flow chart representation of the flow of control in if-statement in the screen



Flow of control in *if-statement*

As a general rule, we express a condition using C's 'relational' operators. The relational operators allow us to compare two values to see whether they are equal to each other, unequal, or whether one is greater than the other. From the table, it can be observed that how the expressions look and how they are evaluated in C.

Expression	**True if**
x==y	x is equal to y
x!=y	x is not equal to y
X<y	x is less than y
x>y	x is greater than y
X<=y	x is less than or equal to y
x>=y	x is greater than or equal to y

Let me now show you a sample program to illustrate the use of if-else construct

#include <stdio.h>

```
main( )
{
        int mark ;
        printf ( "Enter the score " ) ;
        scanf ( "%d", &score) ;
        if ( num >= 50 )
        {
                printf ( "Congratulations !!" ) ;
        }

}
```

In this example, if you enter a score which is greater than or equal to 50 the statements inside the printf will be displayed.

if-else Construct

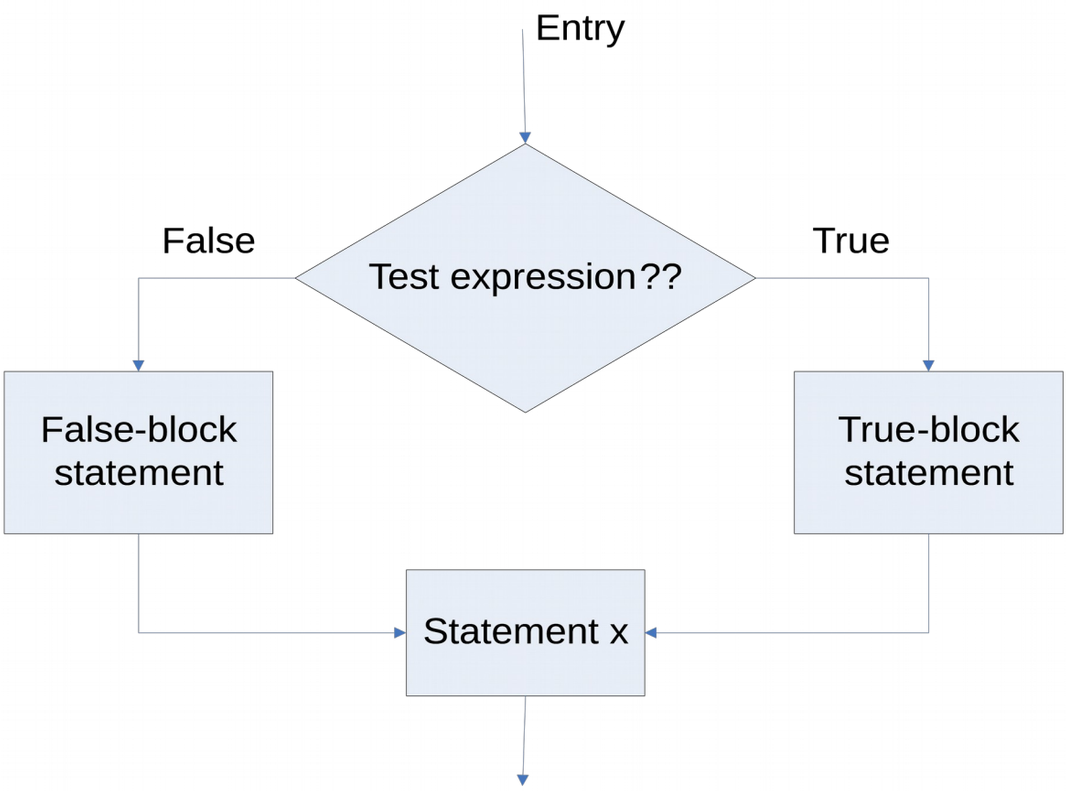
We have seen that the **if** statement by itself will execute a single statement, or a group of statements, when the expression following **if** evaluates to true. It does nothing when the expression evaluates to false. Now a question for you, can we execute one group of statements if the expression evaluates to true and another group of statements if the expression evaluates to false? Yes this is what is the purpose of the **if**-**else** statement.

The general form of if-else construct is :

```
if (test expression)
{
        True-block statements
}
else
{
        False-block statements
}

statement x;
……..……
```

If the test expression is true, then the true-block statements, immediately following the if statement are executed; otherwise , the false block statements are executed. Here either true block or false block will be executed, not both. This control flow is illustrated in the flow chart representation as shown in the screen.



Flow of control in if-else statement

Now let us go through a simple program to find the biggest of two numbers which clearly shows the use of if-else statement.

```
#include<stdio.h>
void main()
{
        int x,y;
        printf("Enter value for x :");
        scanf("%d",&x);
        printf("Enter value for y :");
        scanf("%d",&y);
        if ( x > y )
            {
              printf(" The large number is %d\n", x);
        }
        else
            {
              printf("The large number is %d\n", y);
        }
}
```

In this program, if x is the biggest one then statements inside *if* block is executed otherwise the block of statements inside *else* part is executed.

It is obvious that, we cannot solve all problems related to decision making with a simple **if** construct or **if-else** construct, for this we can expand this concept in the different ways. We will discuss them one by one.

Nested if-else statements

When a series of decisions are involved, we may have to use more than one if-else statement in the nested form. When an if-else construct appears as a statement in another if-else, it is known as nested if-else construct. General form of if-else construct is shown in the screen.

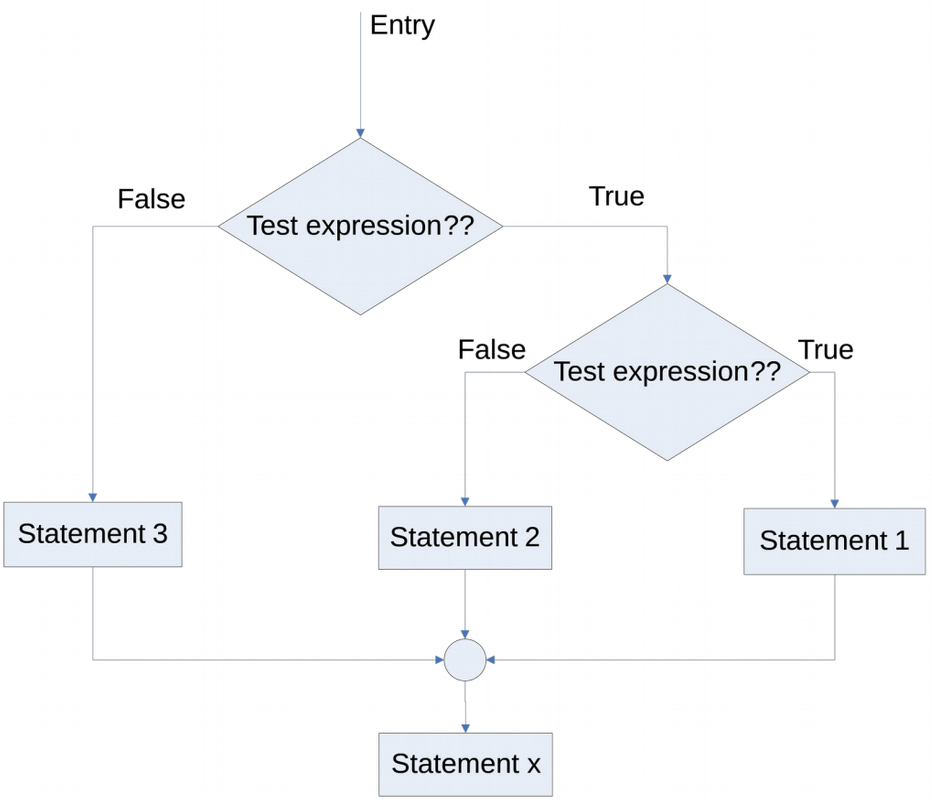
```
if(test condition a)
{
        if(test condition b)
        {
                Statement 1;
        }
        else
        {
                Statement 2;
        }
}
else
{
        Statement 3;
}

Statement x;
```

Now let us understand the execution of this nested if-else structure:

If the ***condition a*** is false then the statement 3 will be executed; otherwise it continues to perform the second test. If the ***condition b*** is true the statement 1 will be executed; otherwise statement 2 will be evaluated and then the control is transferred to the statement x.

To get more clarity let us now see the use of nested if-else construct as a flow chart as shown in the screen.



Flow chart for nested if-else statement

Now let me show you a quick demo of nested if-else statement with the help of a simple program to find the largest of three numbers.

```
#include<stdio.h>
Main()
{
        int x,y,z;
        printf("Enter three numbers :");
        scanf("%d %d %d", &x , &y , &z);
        if(x>y)
        {
                if(x>z)
                {
                        printf("x is the largest number");
                }
                else
                {
                        printf ("z is the largest number");
                }
        }
        else
        {
                if(z>y)
                {
                        printf("z is the largest number");
                }
                else
                {
                        printf ("y is the largest number");
                }
        }
}
```

Next we will see the else if ladder statement (OR if-else if construct) and its usage.

This method is used when multipath decisions are involved in a problem. A multipath decision is a chain of **if** statements in which the statements associated with each **else** is an **if.** The general form is :

```
        if(condition 1)
                statement-1;
        else if(condition 2)
                statement-2;
              else if(condition 3)
                      statement 3;
                    else if(condition 4)
                            statement 4;
                          ……….
                          ……….
                           else
                                default statement;

        statement x;
```

In this case every **else** is associated with its previous **if**. The last **else** goes to work only if all the conditions fail. Even in **else if** ladder the last **else** is optional. If there is no else part with the respective if, the compiler associates the else part with the closest inner if construct which doesn't have an else part. Care must be taken to use braces appropriately for association of else part with its if.

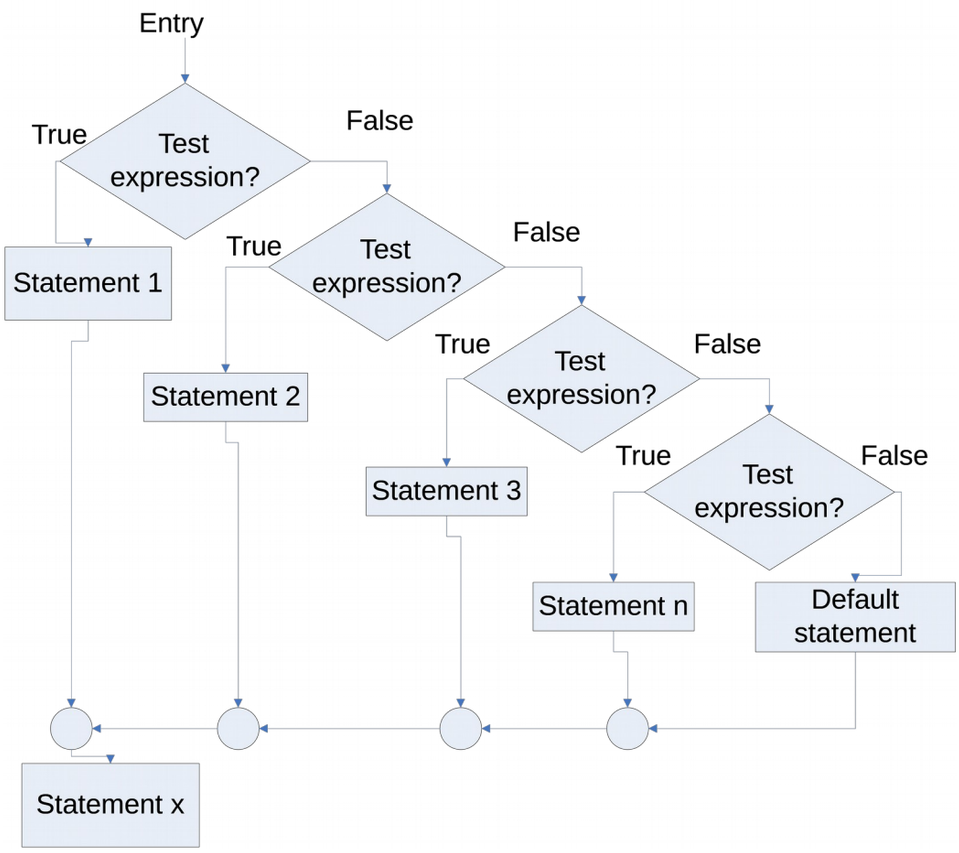
Note that the **else if** clause is nothing different. It is just a way of rearranging the **else** with the **if** that follows it. This would be evident if you look at the code given in the screen:

```
if ( i == 1 )
        printf ( "red" ) ;
else
{
        if ( j == 2 )
        printf ( "apple" ) ;
}
```

```
if ( i == 1 )
        printf ( "red" ) ;

else if(j==2)
        printf("apple");
```

To get more clarity let us now see the use of if-else ladder statement with the help of flow chart as shown in the screen.



Flow chart for else-if ladder

Now let me show you a quick demo of else-if ladder statement with the help of a simple C program to check the class obtained by the students from their mark obtained from five subjects.

```
#include<stdio.h>
main( )
{
        int m1, m2, m3, m4, m5, per ;
        printf("enter the mark of five papers");
        scanf("%d%d%d%d%d",&m1,&m2,&m3,&m4,&m5);
        per = ( m1+ m2 + m3 + m4+ m5 ) / 5 ;

        if ( per >= 60 )
                printf ( "First class" ) ;
        else if ( per >= 50 )
                printf ( "Second class" ) ;
        else if ( per >= 40 )
```

```
                printf ( "Third class" ) ;
        else
                printf ( "fail" ) ;
}
```

Module – 3

Switch-case construct

In the previous methods when the program needs more and more alternatives the complexity of the program increases and also it is difficult to read and follow those codes. To overcome this, C has a built in multi-way decision statement known as **switch-case construct**. The **switch** statement testes the value of a given expression against a list of case values and the block of statements associated with the matching case value is executed. Three keywords **switch, case,** and **default,** go together to make up the control statement.

The general format is:

```
switch (expression )
{
case 1 :
        statements;
        break;
case 2 :
        statements;
        break;

case 3 :
        statements;
        break;

default :
        statements ;
        break;
}
```

Each **case** is labeled by an integer or character expression yielding an integer value known as case labels. The expression is evaluated first and its value is then matched against the case labels. If a case matches the expression value, execution starts at that case. All case expressions must be different. The case labeled default is executed if none of the other cases are satisfied. A **default** is optional; if it isn't there and if none of the cases match, no action at all takes place. Cases and the default clause can occur in any order.
If a break is not included in each case statement, whenever a match is found, the program executes the statements following the case and also all the subsequent case statements and default statements. Even though a break statement is not necessary in the default part, it is a good programming practice to add it always.
The colon (:) must be placed at the end of each case label and default. The braces following the labels are optional.

Now we can see the use of switch-case construct with the help of a program segment written to find the grade of students.

```
-------------------------------------------------
-------------------------------------------------

point =mark/total*10;
switch(point)
{
        case 10 :
        case 9 :
        case 8 :
```

```
                    grade = ”A+”;
                    break;
        case 7 :
        case 6 :
                    grade = “A”;
                    break;
        case 5 :
                    grade = “B”;
                    break;
        case 4 :
                    grade = “C”;
                    break;
        default :
                    grade = “Fail”;
                    break;
}

Printf(“Grade is %s, grade”);
……
……
```

Now let us discuss another conditional control structure named conditional expression.

Module – 4

Conditional expression

The conditional operators **?** and **:** named **ternary operator** since they take three arguments. In fact, they form a kind of foreshortened if-then-else. Their general form is,

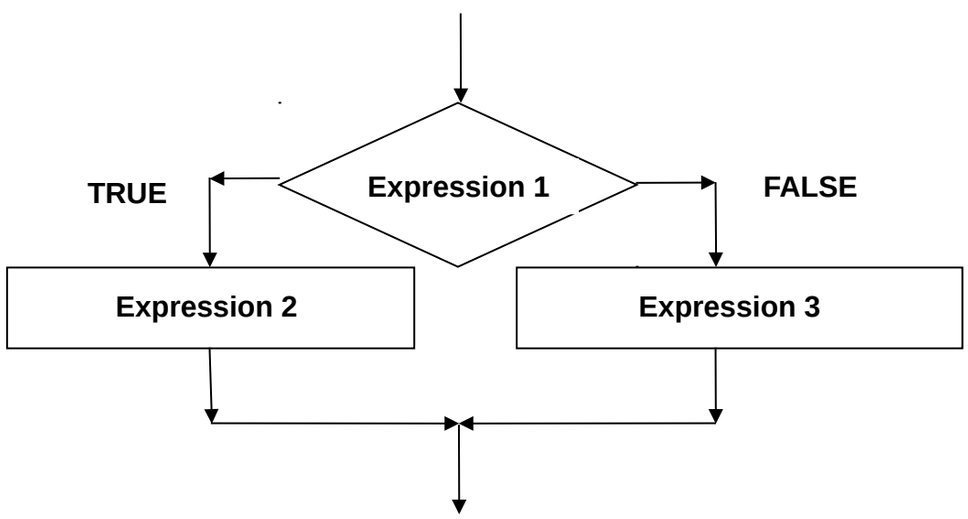
expression 1 ? expression 2 : expression 3

What this expression says is: “if **expression 1** is true (that is, if its value is non-zero), then the value returned will be **expression 2**, otherwise the value returned will be **expression 3**”.

That means a conditional expression will be evaluated as,

1. The expression 1 is evaluated to find the logical value true(non-zero) or false (zero)
2. If the expression 1 is true, then the expression 2 is evaluated and that will be the resulting value of the conditional expression.
3. If the expression 1 is false, then the expression 3 is evaluated and that will be the resulting value of the conditional expression.

For more clarity let us see the flow chart representation



For example the segment

```
if(x==1)
        flag=1;
else
        flag=0;
```

can be written as

```
flag = (x==1) ? 1 : 0;
```

also consider an example of finding largest of three numbers using conditional operator :

```
……..
 ….        ……..

big = ( a > b ? ( a > c ? 1: 3 ) : ( b > c ? 2: 3 ) ) ;
……..
……..
```

Now we have covered conditional selective control structures including if-else (nested if and if-else ladder), switch-case and Conditional expression. Let us now move to Loop conditional control structures.

There are many more other features in C programming language are remaining to explore. So let us wait for the coming sessions. Till then bye.

Summary

C is a structured programming language. A simple statement is terminated by a semicolon and compound statements are written with the braces known as blocks. C supports many control structures broadly classified under conditional execution and unconditional execution.

Control instructions' determine the 'flow of control' in a program. The control-flow of a language specifies the order in which computations are performed. C supports the control structures classified under two main categories Conditional and Unconditional control structures. In the case of conditional execution, the flow of execution may be transferred from one part of the program to another part based on the output of the conditional test carried out. In selective constructs, the statement is selected for execution based on the output of the conditional test given by the expression. C supports selective constructs such as Conditional expression , if – else , switch-case etc.

switch-case construct is a multi-way decision statement. It is used to transfer the control to more than two alternatives.

Assignments

- Explain the classification of control structures in C.
- Explain in detail the different forms of if construct in C.
- Illustrate the use of ternary operator in C.
- Compare the switch-case structure with the if-else structure. Which is more convenient? Give a suitable example.
- Write a program to check whether the entered year is leap year or not using switch-case construct.

Reference

- *B. W. Kernighan and D. M. Ritchie*, The C Programming Language, Prentice-Hall, Englewood Cliffs, New Jersey, 1978.
- *Greg Perry,* Absolute Beginners' guide to C 2nd Edition, SAMS publishing, A division of Prentice Hall Computer Publishing, 201 West 103rd Street , Indianapolis, Indiana 46290. April 1994.
- Yashavant Kanetkar; Let us C, BPB Publications, New Delhi.