

Module 2

Control Structures/Statements:

1. Statements are the instructions given to the computer to perform any kind of action. Statements are always terminated by semicolon.
2. The group of statements is called block. These statements are enclosed between the pair of braces ({ }). The opening brace ({) signifies the beginning and closing brace (}) signifies the end of the block.
3. The normal flow of execution in a high level language is sequential, i.e., each statement is executed in the order of its appearance in the program.
4. However, depending on the requirements of a problem it might be required to alter the normal sequence of execution in a program.
5. The statements which specify the order of execution of statements are called control flow statements.
6. Control structures are used to transfer the control from one statement in a program to any other statement. The control statements are classified as:
 - a. Conditional Statement
 - b. Looping Statement

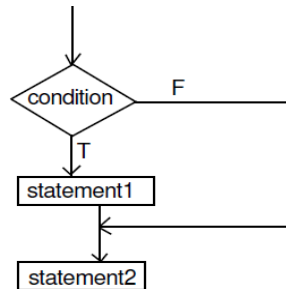
Conditional Statements:

Condition to make a decision which involve logical test that either results in true or false. Depending upon result, statement will execute and after that control transfer to statement in program and start execution from that point. This is called conditional execution.

- a. if statement consists of a Boolean expression followed by one or more statements. if statements allows branching depending upon the value or state of variables. This allows statements to be executed or skipped, depending upon decisions.

Syntax:

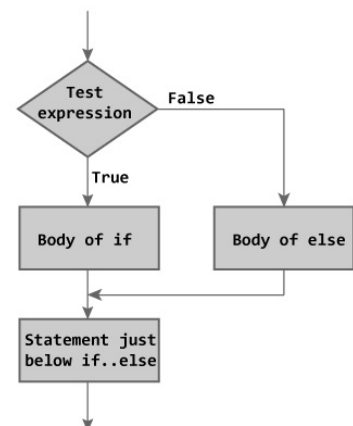
```
if(condition)
{
    statement(s);
}
```



- b. if else statement: if statement can be followed by an optional else statement, which executes when the Boolean expression is false. if...else statement is used if the programmer wants to execute some statement/s when the test expression is true and execute some other statement/s if the test expression is false.

Syntax:

```
if (condition)
{
    statements to be executed if test expression is true
}
else
{
    statements to be executed if test expression is false
}
```



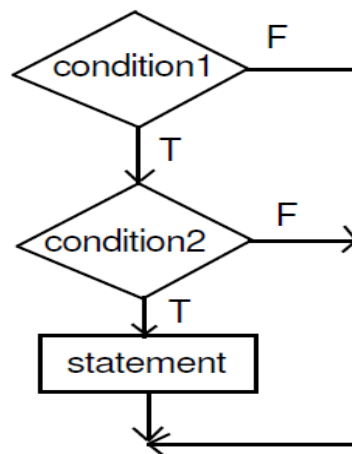
- c. Nested if Statement: If an if..else statement appears as a statement in another if or else block then it is called nested if..else.

Syntax:

```

if (condition1)
{
if (condition2)
{
Statement block-1;
}
else
{
Statement block-2;
}
}
else
{
Statement block-3;
}
Next statement

```



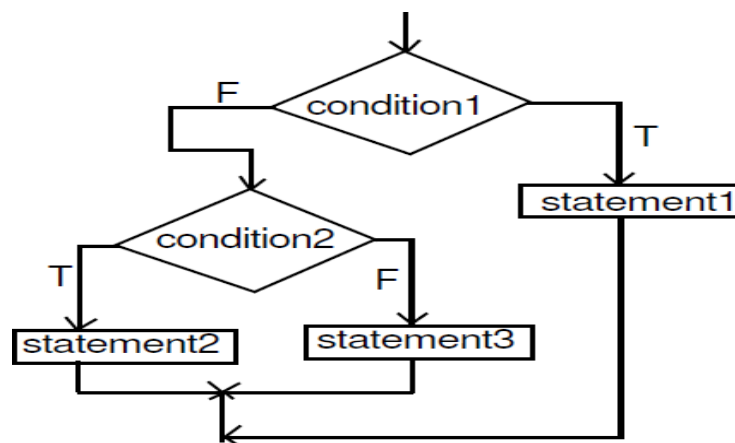
- d. Else if ladder: The else..if ladder is used for multiple decision making. In this construct each else is associated with another if statement. The default else is optional. The conditions are evaluated from top to bottom one by one.

Syntax

```

if(condition1)
{
Statement block- 1;
}
else if(condition2)
{
Statement block-2;}
else if(condition3)
{
Statement block-3;
}
.....
else if(condition-n)
{
Statement block-n;
}
else
{
Default statement block;
}

```



- e. switch statement: The execution of switch statement begins with the evaluation of expression. If the values of expression match with the constant then the statements following this statement execute sequentially till it executes break. If the value of expression does not match with any constant, the statement with default is executed. If a programmer has to choose one block of statement among many alternatives, nested if...else can be used but, this makes programming logic complex. This type of problem can be handled in C programming using switch statement.

Syntax:

```

switch (expression)
{
case constant1: statement(s)1;
               break;
case constant2: statement(s)2;
               break;
               ....
case constantn: statement(s)n;
               break;
default: statement(s);
}

```

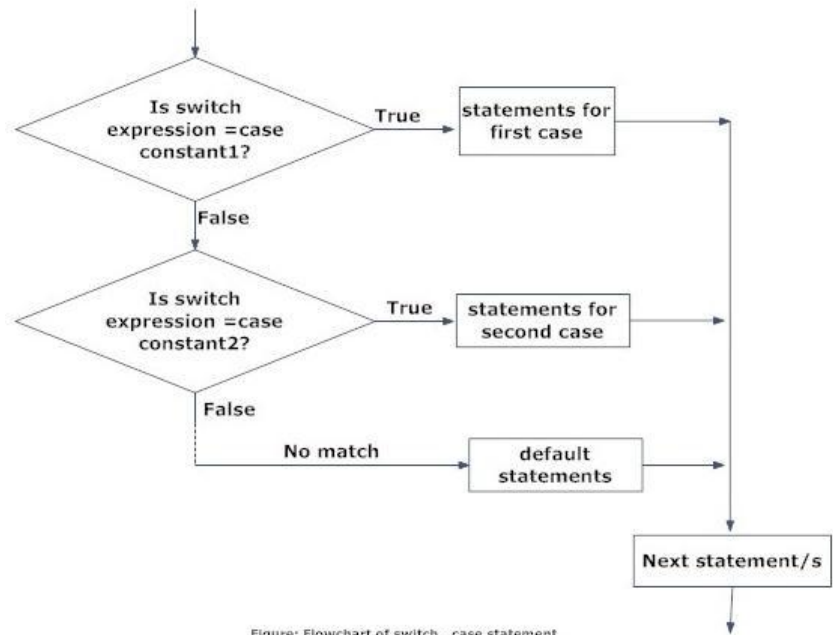


Figure: Flowchart of switch...case statement

Sr. No.	if else statement	switch statement	ladder if else statement	nested if else statement
1.	In this we can test only one condition	In this case we can test multiple conditions	In this case we can test multiple conditions	In this case we can test multiple conditions
2.	In this we cannot have different conditions	In this case we can have only one expression but various value of the same expression	In this case we can have multiple conditions independent of each other.	In this case we can have multiple conditions independent of each other.
3.	This is used when there is only one condition and one value of the same to be tested	This is used when there is only one condition but multiple values of the same to be tested	This is typically used when there are multiple conditions to be tested.	This is typically used when there are multiple conditions to be tested.
4.	It can have values based on constraints	It can have values based on user choice	It can have values based on constraints	It can have values based on constraints
5.	In this first the condition is tested and then it comes to else if the condition is not true	In this case first the case is checked and then it switches to that case	In this first the condition is tested and then it comes to else if the condition is not true and then the other conditions are tested	In this first the condition is tested and then it comes to else if the condition is not true and then the other conditions are tested
6.	if else statement is used to evaluate a condition to be true or false	A switch case statement is used to test for multiple values of the same variable or expression like 1, 2 3 etc	ladder if else is used to test multiple conditions to be true of false and accordingly perform various tasks according to the correct condition	In a nested if else, we can have multiple conditions evaluated based on the previous condition to be true to decide whether to perform a task or not

Loop Statements (Repetitive control structure):

Sometimes it is required that a set of statements to be executed a number of times by changing the value of one or more variables each time to obtain a different result. This type of program execution is called looping.

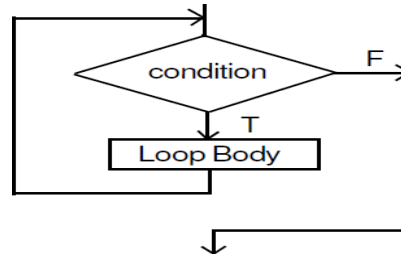
A loop consists of control statement and its body. Control statements perform logical test whose result is either true or false. If result is true, then the statements contained in the body of loop are executed otherwise the loop is terminated. Control statement placed before the body of loop is entry control loop whereas control statement written after the body of loop is called exit control loop.

a. while loop:

While loop checks whether the test expression is true or not. If it is true, code/s inside the body of while loop is executed, that is, code/s inside the braces { } are executed. Then again the test expression is checked whether test expression is true or not. This process continues until the test expression becomes false.

Syntax:

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



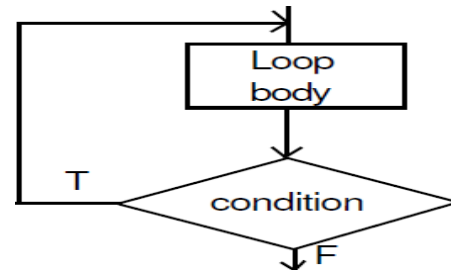
b. do while loop:

In C, do...while loop is very similar to while loop. Only difference between these two loops is that, in while loops, test expression is checked at first but, in do...while loop code is executed at first then the condition is checked. So, the body of loop is executed at least once in do...while loops.

Syntax

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

Note: There is semicolon in the end of while (); in do...while loop

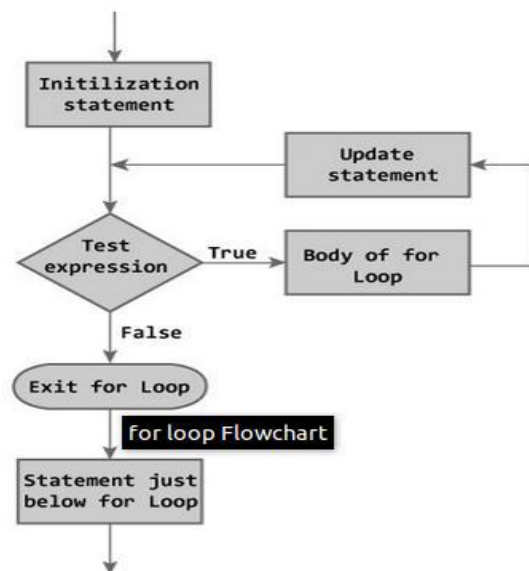


c. For loop:

It is a count controlled loop. The initialization statement is executed only once at the beginning of the for loop. Then the condition is checked by the program. If it is false, the loop is terminated and if it is true then the code inside body is executed and finally control variable is updated. This process repeats until test expression is false.

Syntax:

```
for (initialization; condition; updation)
{
statement(s);
}
```



For loop	While loop	Do while loop
Syntax: For(initialization; condition; updation) { Statements; }	Syntax: While(condition) { Statements; }	Syntax: Do { Statements; } While(condition);
It is known as entry controlled loop	It is known as entry controlled loop.	It is known as exit controlled loop.
If the condition is not true first time then control will never enter in a loop	If the condition is not true first time then control will never enter in a loop.	Even if the condition is not true for the first time the control will enter in a loop.
There is no semicolon; after the condition in the syntax.	There is no semicolon; after the condition in the syntax.	There is semicolon; after the condition in the syntax.
Initialization and updating is the part of the syntax.	Initialization and updating is not the part of the syntax.	Initialization and updating is not the part of the syntax

Jump Statements

The jump statements unconditionally transfer program control within a function. Loops perform a set of repetitive task until text expression becomes false but it is sometimes desirable to skip some statement/s inside loop or terminate the loop immediately without checking the test expression. In such cases, break and continue statements are used.

1. **goto statement:** After the execution of goto statement, the control transfers to the line of specified label. A label is any name given to a particular part in the program. The label is to be followed with a colon (:).

syntax
goto pgr;

.....

.....

pgr:

Note: It is not a good programming to use goto statement in a program.

2. **break statement:**

The break statement neglects the statements in the loop and transfers the control outside the loop. That is, it breaks the inner loop.

syntax of break statement

```
while (condition)
```

```
{
```

```
statement 1;
```

```
if (condition)
```

```
    break ;
```

```
statement2;
```

```
}
```

```
statement3;
```

3. **continue statement:**

The continue statement causes a program to skip the rest of the body of the loop and transfers the control back to the starting of the loop for next iteration.

```
while (condition)
```

```
{
```

```
statement 1;
```

```
if (condition)
```

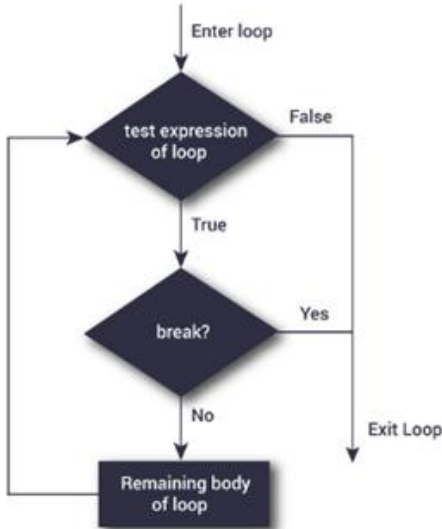
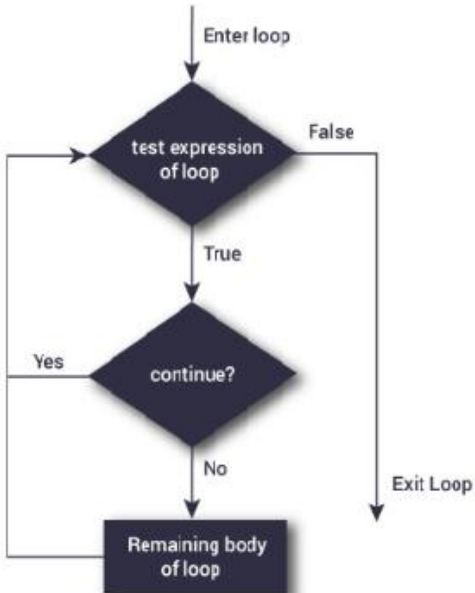
```
    continue;
```

```
statement2;
```

```

}
statement3;

```

Break	continue
A break can appear in both switch and loop (for, while, do) statements.	A continue can appear only in loop (for, while, do) statements. It will give an error if it appears in switch statement.
A break causes the switch or loop statements to terminate abruptly the moment break is encountered.	A continue doesn't terminate the loop, it skip statements in the current round and causes the loop to go to the next iteration.
When a break statement is encountered, it terminates the block and gets the control out of the switch or loop.	When a continue statement is encountered, it gets the control to the next iteration of the loop.
A break causes the innermost enclosing loop or switch to be exited immediately.	A continue inside a loop nested within a switch causes the next loop iteration.
Syntax	Syntax
 <pre> graph TD Start([Enter loop]) --> Test{test expression of loop} Test -- False --> Exit([Exit Loop]) Test -- True --> Break{break?} Break -- Yes --> Exit Break -- No --> Body[Remaining body of loop] Body --> Test </pre>	 <pre> graph TD Start([Enter loop]) --> Test{test expression of loop} Test -- False --> Exit([Exit Loop]) Test -- True --> Continue{continue?} Continue -- Yes --> Test Continue -- No --> Body[Remaining body of loop] Body --> Test </pre>

break	continue	goto
<pre> for(i=1;i<=5;i++) { if(i==2 i==4) { break; } printf("%d",i); } Output: 1 </pre>	<pre> for(i=1;i<=5;i++) { if(i==2 i==4) { continue; } printf("%d",i); } Output: 135 </pre>	<pre> int i=1; lbl: if(i<=5) { printf("%d",i); i++; goto lbl; } Output: 12345 </pre>