

## Unit - 3

# **Decision Control statemnet And Storage Class**

# Decision Control/Conditional Statements

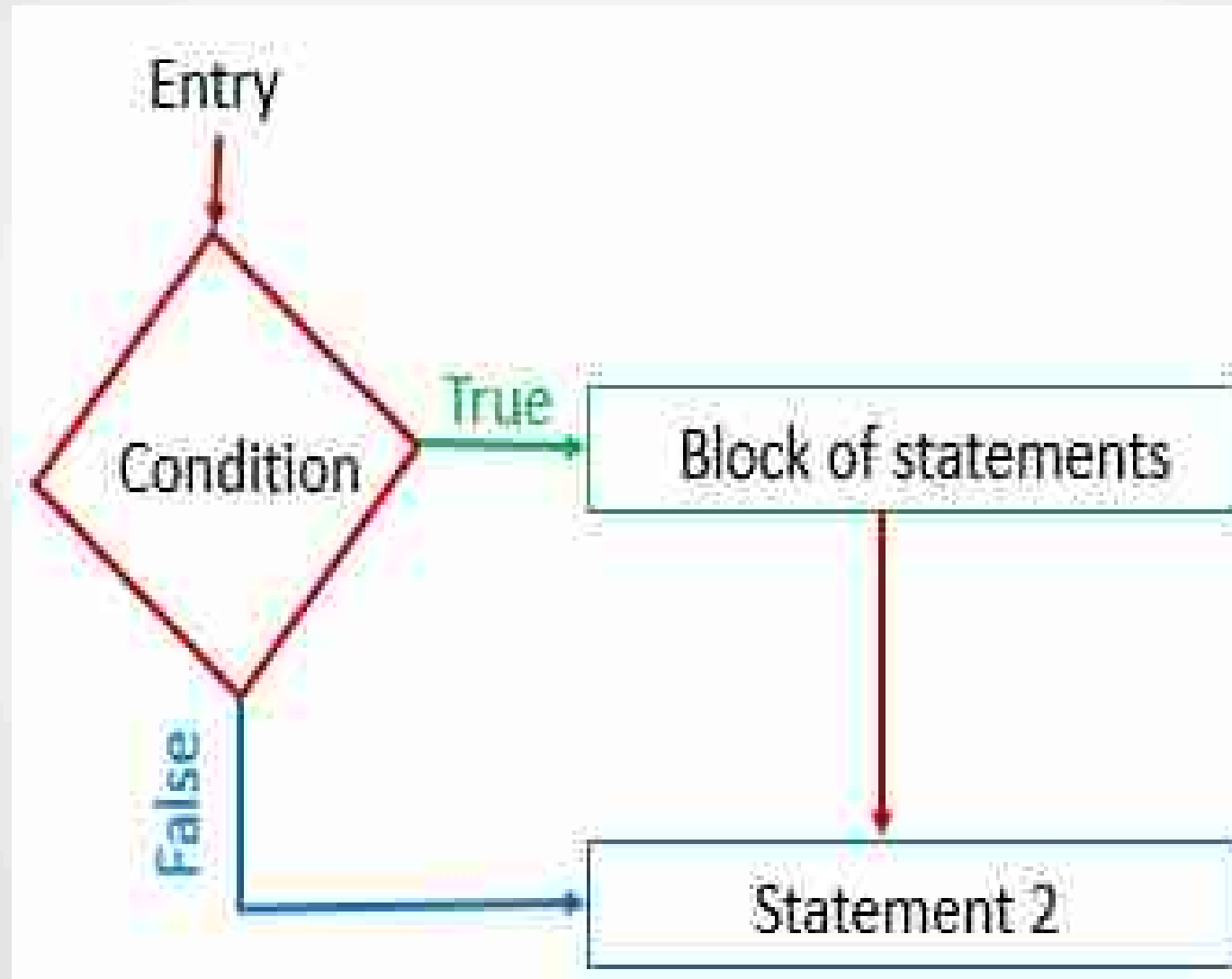
- The if-else statement in C is used to perform the operations based on some specific condition.
- In any programming language, there is a need to perform different tasks based on the condition. For example, consider an online website, when you enter wrong id or password it displays error page and when you enter correct credentials then it displays welcome page. So there must be a logic in place that checks the condition (id and password) and if the condition returns true it performs a task (displaying welcome page) else it performs a different task (displaying error page).
- There are the following types of if statement in C language.
  - If statement
  - If-else statement
  - If else-if ladder
  - Nested if

# Decision Control/Conditional Statements

- Decision making structures require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements 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.
- **If Statement:** The statements inside if body executes only when the condition defined by if statement is true.
- **Syntax:**

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

# Simple If

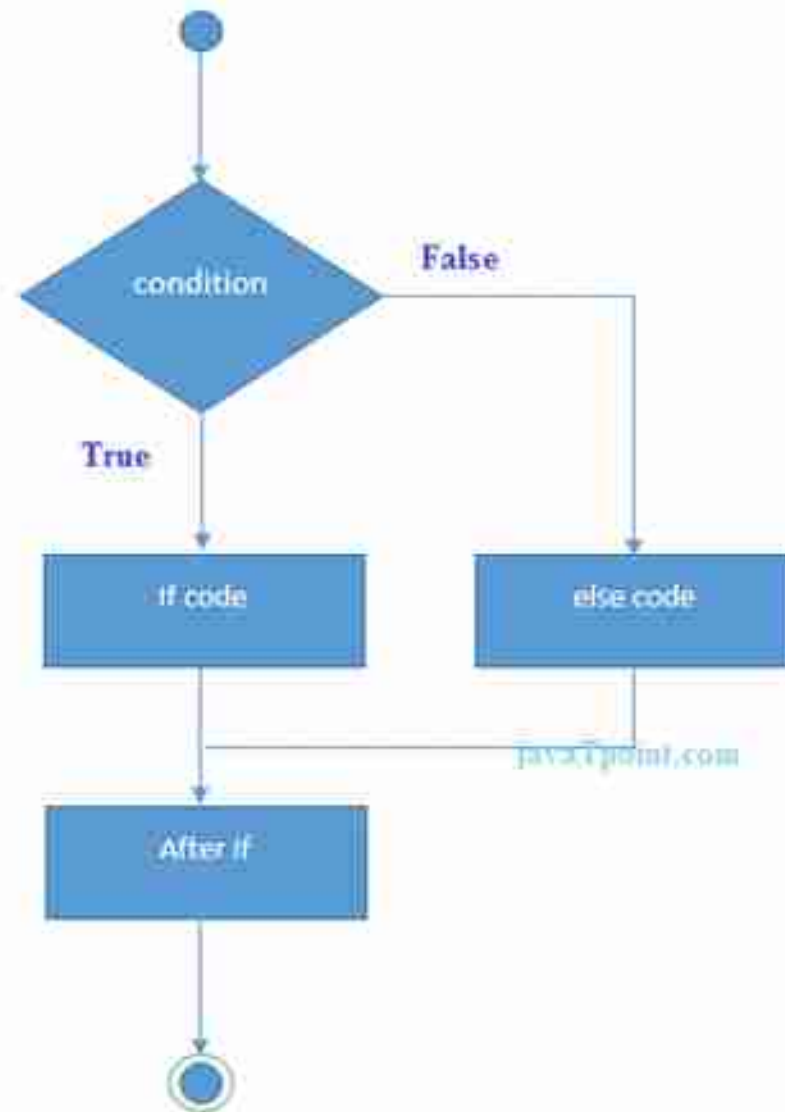


# If - Else

- **If-else Statement:** The if-else statement is used to perform two operations for a single condition.
- We can perform two different operations, i.e., one is for the correctness(true) of that condition, and the other is for the incorrectness(false) of the condition.
- **Imp Note:** if and else block cannot be executed simultaneously.
- **Syntax:**

```
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 - Else



# If else-if ladder Statement

- **If-else-if ladder Statement:** The if-else-if ladder statement is an extension to the if-else statement.
- It is used in the scenario where there are multiple cases to be performed for different conditions.
- In if-else-if ladder statement, if a condition is true then the statements defined in the if block will be executed, otherwise if some other condition is true then the statements defined in the else-if block will be executed, at the last if none of the condition is true then the statements defined in the else block will be executed.
- There are multiple else-if blocks possible.
- It is similar to the switch case statement where the default is executed instead of else block if none of the cases is matched.

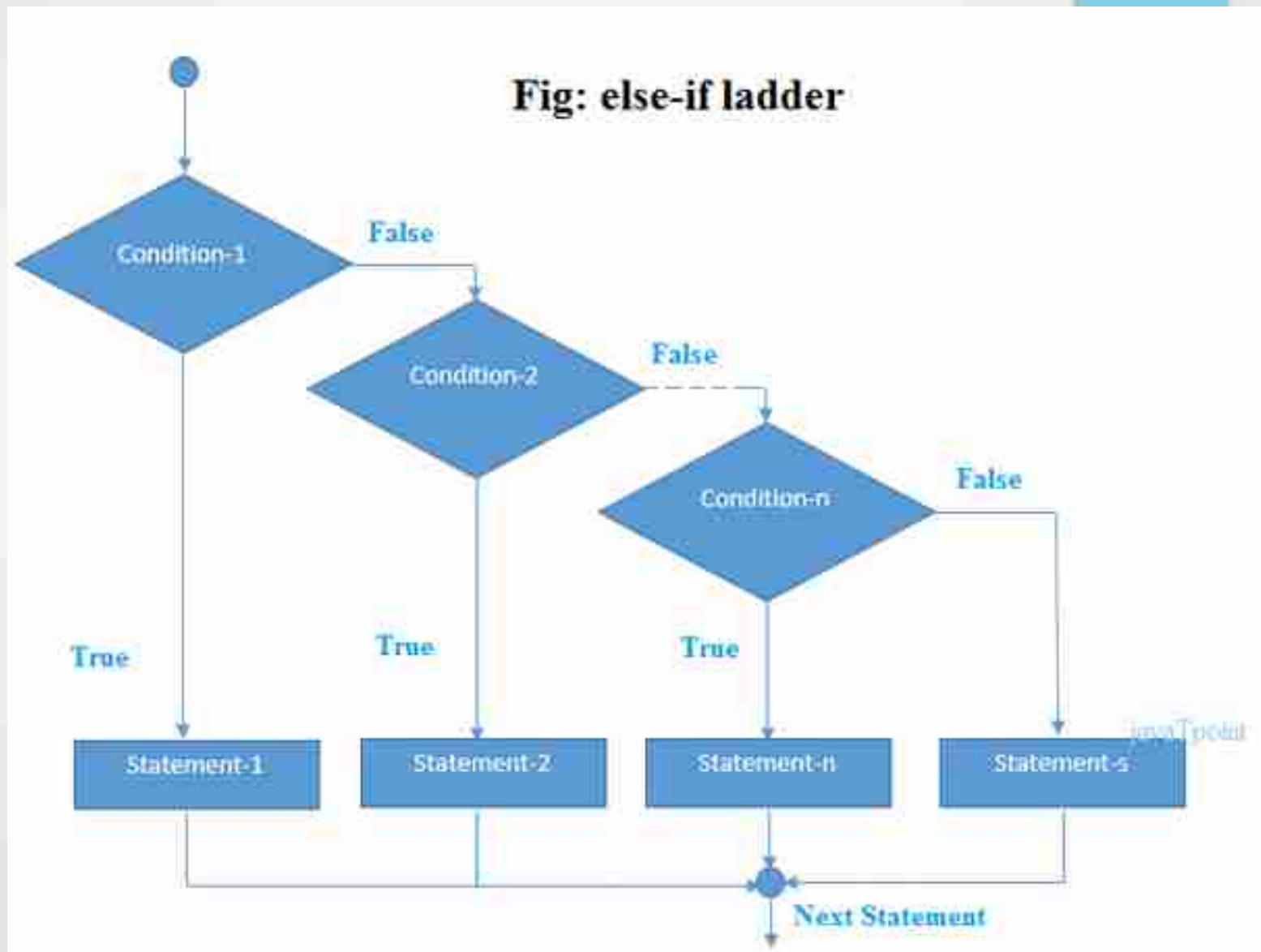
# If else-if ladder Statement

- **Syntax:**

```
if(condition1){  
    //code to be executed if condition1 is true  
}  
else if(condition2){  
    //code to be executed if condition2 is true  
}  
else if(condition3){  
    //code to be executed if condition3 is true  
}  
...  
else{  
    //code to be executed if all the conditions are false  
}
```



# If else-if ladder Statement



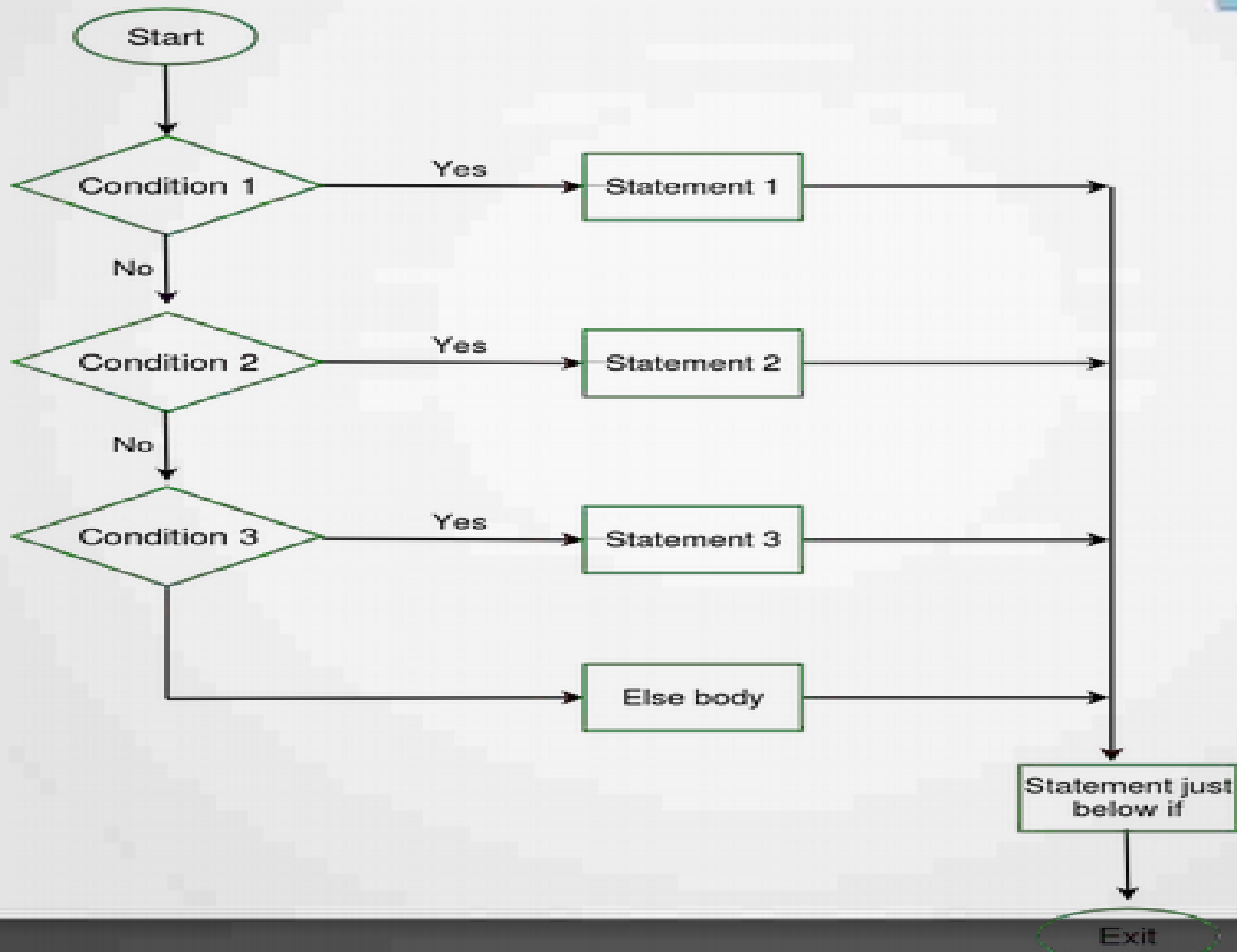
# Nested If

- **Nested If: Placing If Statement inside another IF Statement**
- When an if else statement is present inside the body of another “if” or “else” then this is called nested if else.

- **Syntax:**

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

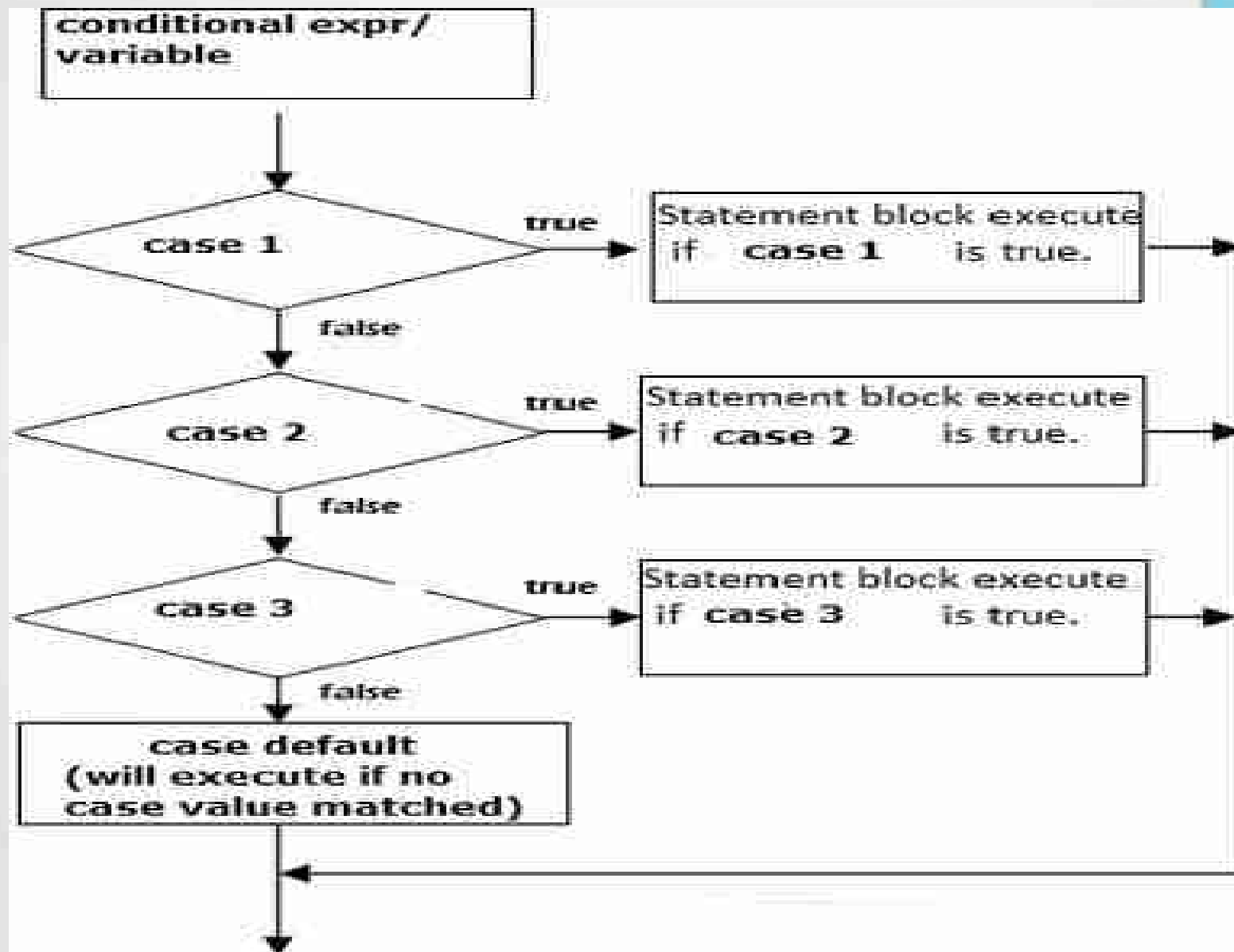
# Nested If



# Switch Case

- 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.

# Switch Case



# Switch Case Syntax

```
switch(expression) {
```

```
    case constant-expression :
```

```
        statement(s);
```

```
        break;
```

```
    case constant-expression :
```

```
        statement(s);
```

```
        break;
```

```
    /* you can have any number of case statements */
```

```
    default :
```

```
        statement(s);
```

```
}
```

# Terms And condition For Switch Case

- The variable used in switch must be of int or char datatype. If one is using char then the value should be in single quote ('a').
- 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.
- When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.

# Terms And condition For Switch Case

- 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.



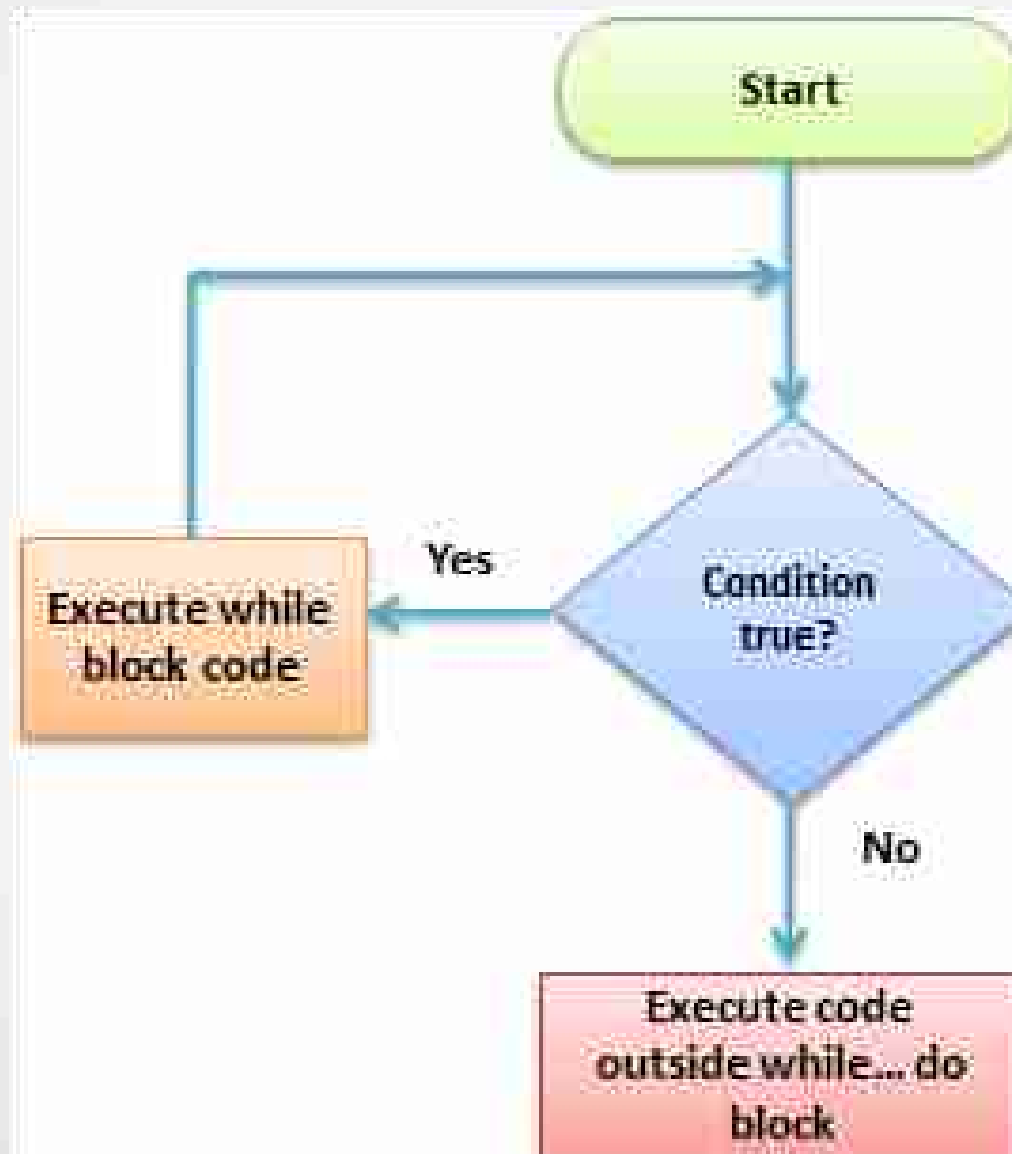
# Looping Techniques

- A Loop executes the sequence of statements many times until the stated condition becomes false.
- A loop consists of two parts, a body of a loop and a control statement. The control statement is a combination of some conditions that direct the body of the loop to execute until the specified condition becomes false.
- The purpose of the loop is to repeat the same code a number of times.
- Loops are used to execute statements or block of statements several times. For example, suppose we want to write a program to print "Hello" 5 times. One way to solve this problem is to write following statement 5 times.
- `printf("hello\n");`
- But what if we want to print it 100 or 1000 times. As you can see this is not ideal way to solve this problem. Using loops we can solve this kind of problem easily.

# Looping Techniques

- Depending upon the position of a control statement in a program, a loop is classified into two types:
  1. **Entry controlled loop**
  2. **Exit controlled loop**
- In an **entry controlled loop**, a condition is checked before executing the body of a loop. It is also called as a pre-checking loop.
- In an **exit controlled loop**, a condition is checked after executing the body of a loop. It is also called as a post-checking loop.

# Looping Techniques

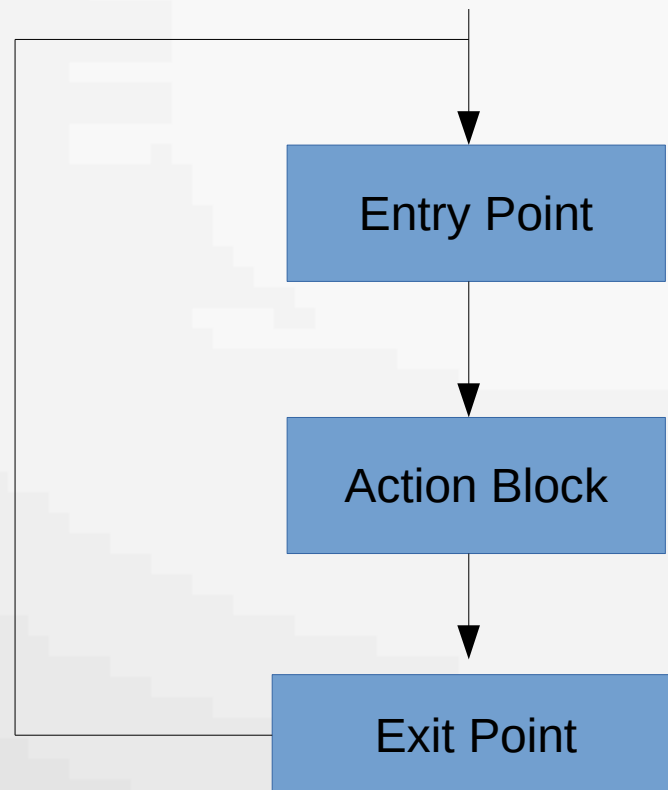



# Looping Techniques

- The control conditions must be well defined and specified otherwise the loop will execute an infinite number of times.
- The loop that does not stop executing and processes the statements number of times is called as an **infinite loop**. An infinite loop is also called as an "**Endless loop**." Following are some characteristics of an infinite loop:
  - 1) No termination condition is specified.
  - 2) The specified conditions never meet.
- The specified condition determines whether to execute the loop body or not.

# Types Of Loops

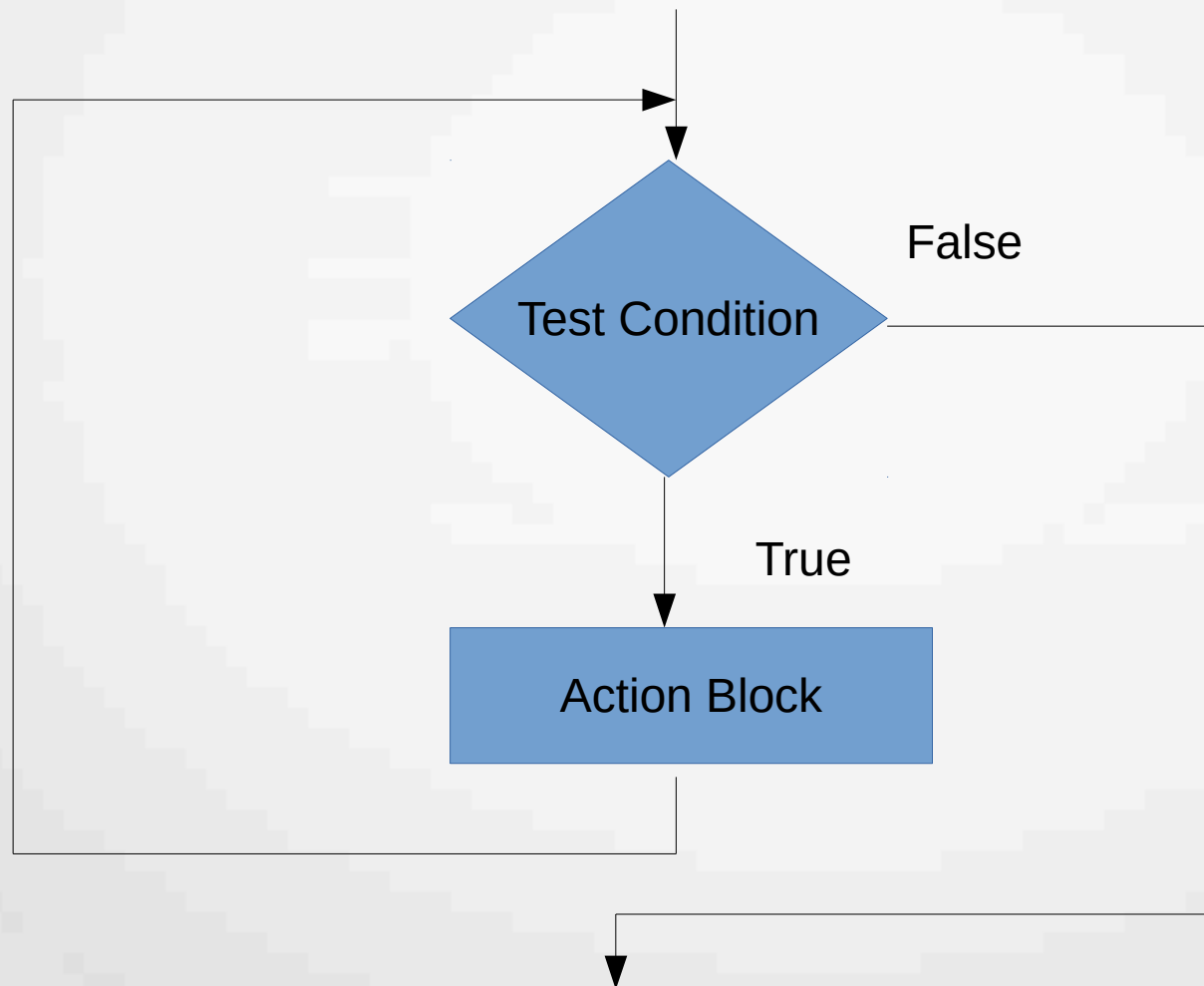
- The loop structure always has an entry point and an exit point.
- The entry point indicates beginning of the loop, and exit point indicates the breaking or terminating point of the loop.



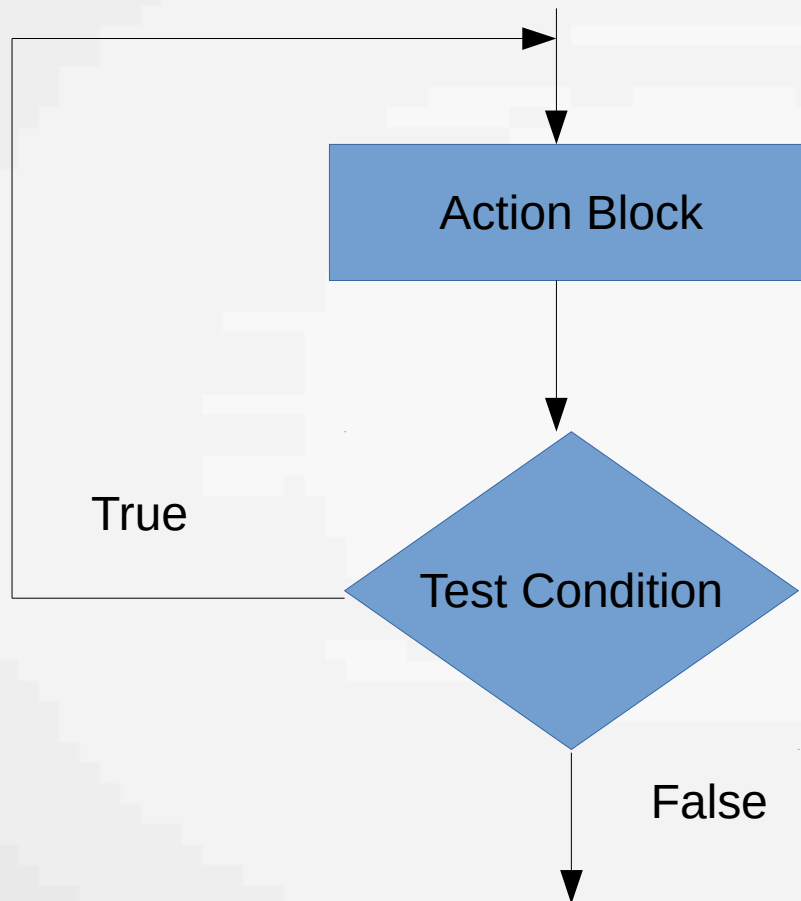
- 
- The figure shows that action block will be repeated infinitely.
  - To stop the loop from being executed infinitely, we need to introduce a test condition that controls the execution of the loop.
  - Loops irrespective of the programming language they are used in , are classified as pretest (entry controled) or posttest (exit control) loops.
  - A loop can be classified as pretest or posttest based on the point where the control test condition is checked.

# Pretest or Entry Control Loop


- If the control test condition is checked at the entry point , the loop is called a pretest loop.



# Post test or Exit Control Loop





- 
- 'C' programming language provides us with three types of loop constructs:
    - 1) The while loop
    - 2) The do-while loop
    - 3) The for loop
  - **Two loop is entry control or pritest**
    - A. For loop**
    - B. While loop**
  - **In exit control loop**
    - A. do...while loop**

# For loop

- For loop is an entry controlled looping statement. It is used to repeat set of statements until some condition is met.
- Looping statements whose condition is checked prior to the execution of its body is called as Entry controlled loop.

- **Syntax:**

```
for(initialization;test condition; increment / decrement)
{
    //statements
}
```

if the test condition is not match in any iteration then loop will be infinite.

For( ; ; ) is infinite loop.

# For loop

- The initial value of the for loop is performed only once.
- The condition is a Boolean expression that tests and compares the counter to a fixed value after each iteration, stopping the for loop when false is returned.
- The incrementation/decrementation increases (or decreases) the counter by a set value.
- **In for loop, a loop variable is used to control the loop. First initialize this loop variable to some value, then check whether this variable is less than or greater than counter value. If statement is true, then loop body is executed and loop variable gets updated . Steps are repeated till exit condition comes.**

- While loop

```
while(test condition)
```


```
{
```

```
    //statements
```

```
    increment / decrement
```

```
}
```

- Unlike for loop, while does not contain initialization and update part. It contains only two parts - condition and body of loop.
- condition is a boolean expression evaluating an integer value. It is similar to if...else condition and define loop termination condition.
- Body of loop contains single or set of statements. It define statements to repeat.

- 
- **Note:** initialize all important loop variable just before the loop.
  - Initially program control is received by condition block. It contains set of relational and logical expressions. If result of the conditional expression is 1 (true) then while transfers program control to body of loop. Else if result of conditional expression is 0 (false) then it exits from loop.
  - Body of loop contain single or set of statements to repeat. It execute all statements inside its body and transfer the program control to loop condition block.
  - Step 1 and 2 are repeated until the loop condition is met.
  - The above two steps are repeated, until loop condition is true
  - in for and while condition will check before every iteration where as in do while statement will execute atleast once before condition check.

- While loop

```
while(test condition)
```

```
{
```

```
    //statements
```

```
    increment / decrement
```

```
}
```

in for and while condition will check before every iteration where as in do while statement will execute atleast once before condition check.

# Do while loop

- A do-while loop is similar to the while loop except that the condition is always executed after the body of a loop. It is also called an exit-controlled loop.
- In the do-while loop, the body of a loop is always executed at least once. After the body is executed, then it checks the condition. If the condition is true, then it will again execute the body of a loop otherwise control is transferred out of the loop

Do

{

    //statements

    //increment / decrement

}

while(test condition);

# Do while loop

- Difference between the while and do-while loop is that in while loop the **while** is written at the **beginning**. In do-while loop, the while condition is written at the end and terminates with a semi-colon (;)

```
int num=1;
do
{
    printf("%d\n", num);
    num++;
} while (num <= 10);
return 0;
```

Diagram illustrating the execution flow of a do-while loop:

1. Initialization: `int num=1;`
2. Loop body execution: `printf("%d\n", num);`
3. Loop body execution: `num++;`
4. Loop condition check: `while (num <= 10);`



# Nested loop

- C programming allows to use one loop inside another loop. The following section shows a few examples to illustrate the concept.

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);  
}
```

- User can nest any number of loops in any type.

Ex :

```
for(i=1;i<=5;i++)  
{  
    for(j=1;j<=i;j++)  
    {  
        printf("%d",i);  
    }  
    printf("\n");  
}
```

# Break,continue and goto

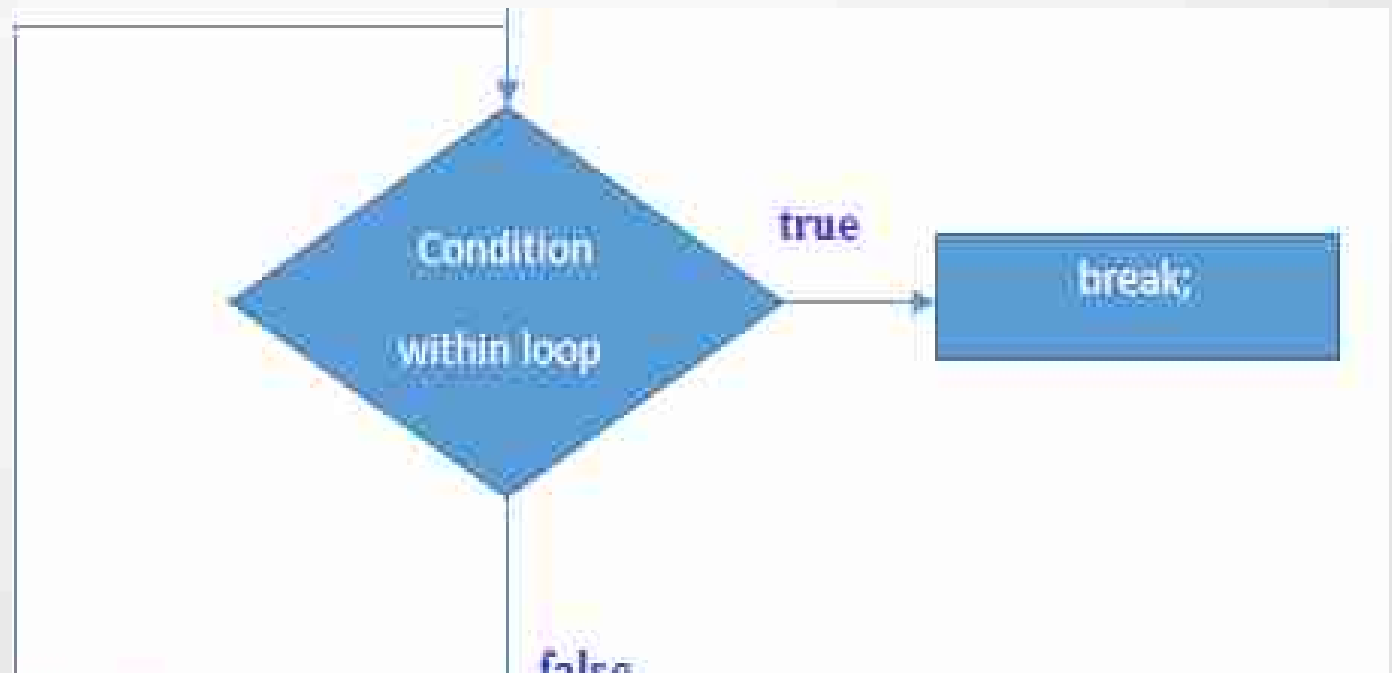
- The break; continue; and goto; statements are used to alter the normal flow of a program.
- Loops perform a set of repetitive task until text expression becomes false but it is sometimes desirable to skip some statement/s inside loop or terminate the loop immediately without checking the test expression. In such cases, break and continue statements are used.


# break

- The break is a keyword in C which is used to bring the program control out of the loop.
- The break statement is used inside loops or switch statement.
- The break is used in terminating the loop immediately after it is encountered.
- it is also used in switch...case statement.

Syntax :

`break;`



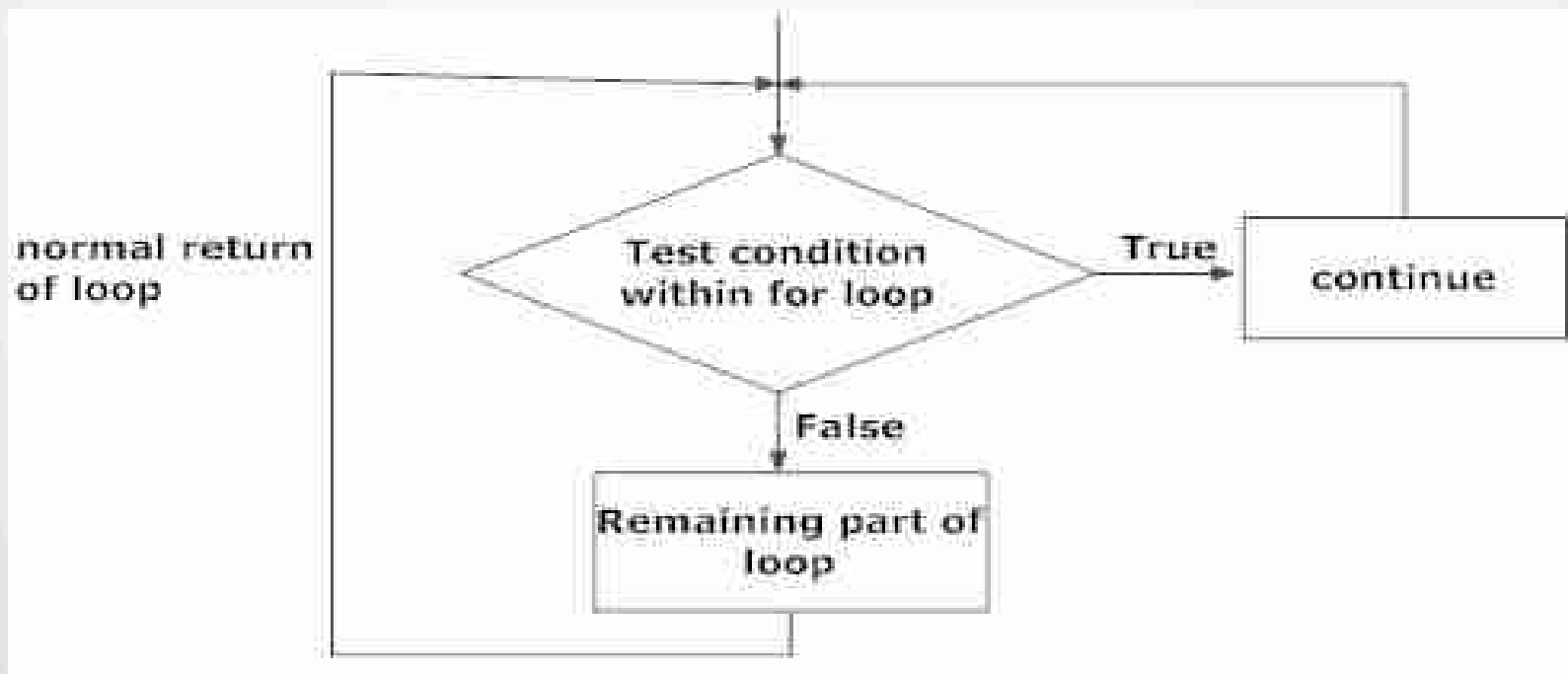


```
for(i=1;i<=10;++i)
{
if(num==5)
{
    break;    /*this breaks for loop if num == 0 */
    printf("Loop Break\n");
}
printf("%d",i);
}
```

output : 1 2 3 4 loop break

# continue

- The continue statement skips some lines of code inside the loop and continues with the next iteration.
- It is mainly used for a condition so that we can skip some code for a particular condition.



# syntax

Continue;

- Ex : for(i=1;i<=10;++i)

```
{
```

```
if(num==5)
```

```
{
```

```
    continue;    /*skip iteration 5 and conitinue */
```

```
}
```

```
Printf("%d",i);
```

```
}
```

- Output : 1 2 3 4 6 7 8 9 10

# goto

- The goto statement is known as jump statement in C. goto is used to transfer the program control to a predefined label.
- The goto statement can be used to repeat some part of the code for a particular condition.
- Also goto statement is used for altering the normal sequence of program execution by transferring control to some other part of the program.
- It is also known as **unconditional jump statement**. The goto statement can be used to jump from anywhere to anywhere within a function.



# goto

- **Syntax:**

```
goto label;
```

```
.....
```

```
.....
```

```
label:
```

```
statement;
```

- In this syntax, label is an identifier.
- When, the control of program reaches to goto statement, the control of the program will jump to the label: and executes the code below it.

## Forward jump

## Backward jump

