

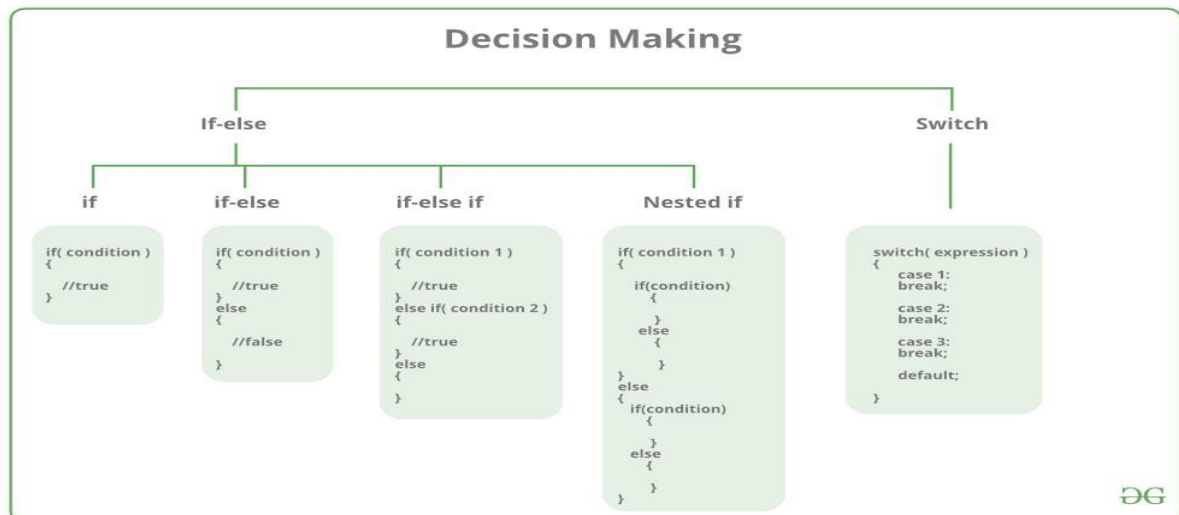
Module 4: C Conditional Statements

- **What is Control or Decision-making Statement in C**
- **If statement**
 - I. Simple If Statement
 - II. If...else Statement
 - III. Nested if Statement
 - IV. Nested If...else statement
 - V. Else if leader
- **Switch Case**
- **Conditional Operator**
- **Jump Statement**
 - i. Goto Statement
 - ii. Break Statement
 - iii. Continue Statement
 - iv. Return Statement

What is Control or Decision-making Statements in C

Decision making is about deciding the order of execution of statement based on certain conditions or repeat a group of statements until certain specified conditions are met. Decision-making statements in programming languages decide the direction of the flow of program execution. Decision-making statements available in C or C++ are:

1. If Statement
2. If-else Statements
3. Nested if-else statements
4. If-else-if leader
5. Switch statements
6. Jump Statements
 - Break
 - Continue
 - Goto



If Statement-

- It is one of the powerful decision-making statements.
- It is used to decide whether a certain statement or block of statements will be executed or not.
- i.e. if a certain condition is true then a block of statement is executed otherwise not.

Syntax-

```

if(condition)
{
    // Statements to execute if
    // condition is true
}
  
```

How if statement works?

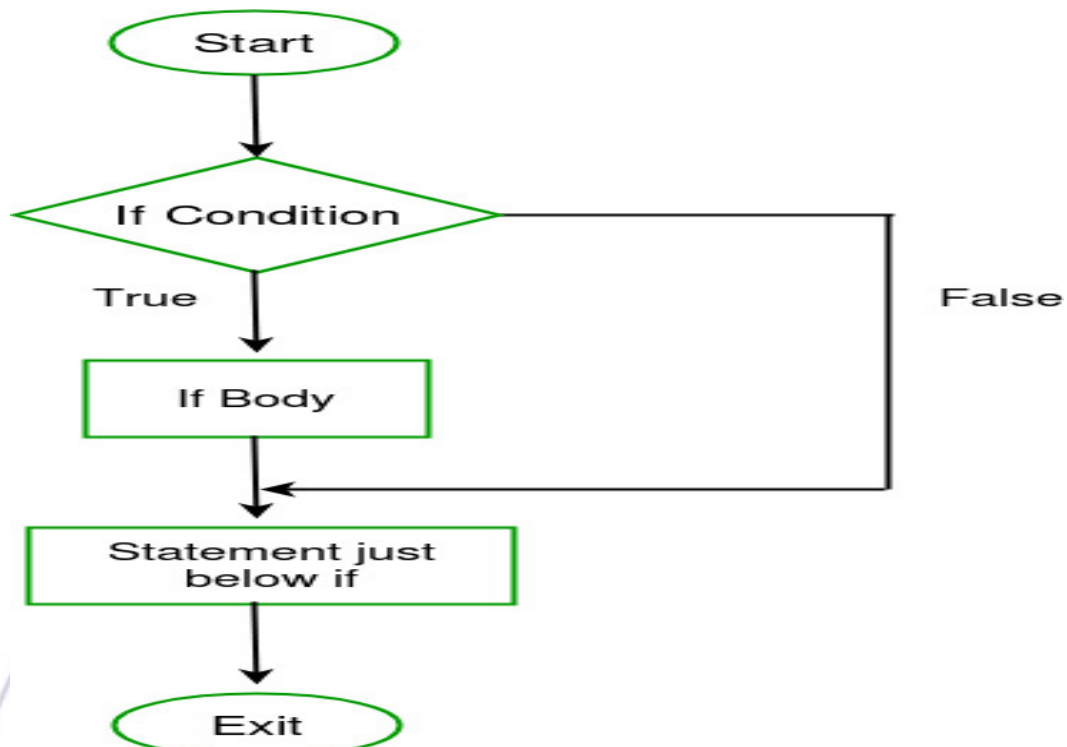
Here, the condition after evaluation will be either true or false. C if statement accepts Boolean values -true or false.

- If the condition is true, statements inside the body of if are executed.
- If the condition is false, statements inside the body of if are not executed.
- If we do not provide the curly braces '{' and '}' after if(condition) then by default if statement will consider the first immediately below statement to be inside its block.

```

if(condition)
    statement1;
    statement2;

// Here if the condition is true, if block
// will consider only statement1 to be inside
// its block.
  
```



Expression is true.

```

int test = 5;

if (test < 10)
{
    // codes
}

// codes after if
  
```

Expression is false.

```

int test = 5;

if (test > 10)
{
    // codes
}

// codes after if
  
```

Working of if Statement

Example:

If Ram can having 100 GeekBits then he can redeem these GeekBits and get the GFG T-shirt .

```

1 // C program to illustrate If statement
2 #include <stdio.h>
3
4 int main()
5 {
6     int i = 10;
7
8     if (i > 15) {
9         printf("10 is greater than 15");
10    }
11
12    printf("I am Not in if");
13 }
14
  
```

input
I am Not in if

As the condition present in the if statement is false. So, the block below the if statement is not executed.

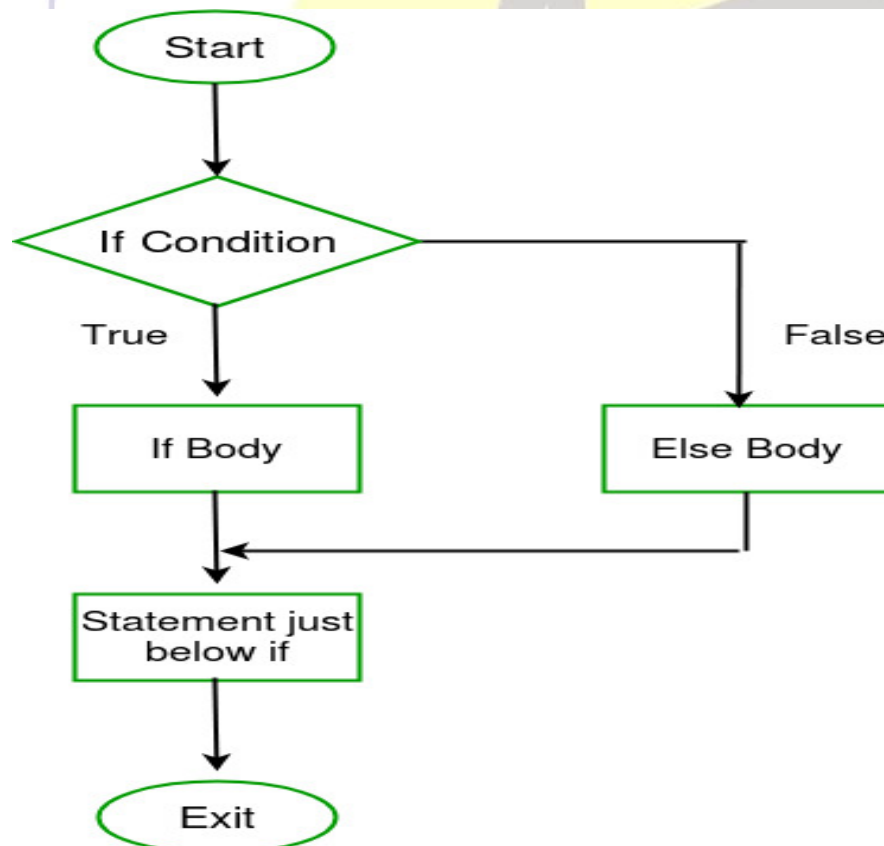
If-else

- The if-else statement is an extension of the simple if control statement.
- The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. But what if we want to do something else if the condition is false.
- Here comes the C else statement. We can use the else statement with the if statement to execute a block of code when the condition is false.

Syntax:

```
if (condition)
{
    // Executes this block if
    // condition is true
}
else
{
    // Executes this block if
    // condition is false
}
```

Flowchart:



Example:

The person who having correct 50 Geek Bits is redeem the gifts otherwise they can't redeem.

```
// C program to illustrate If-else statement
#include <stdio.h>

int main()
{
    int i = 20;
    if (i < 15) {
        printf("i is smaller than 15");
    }
    else {
        printf("i is greater than 15");
    }
    return 0;
}
```

Output:

```
i is greater than 15
```

The block of code following the else statement is executed as the condition present in the if statement is false.

Nested If-else Statement

When multiple decisions are involved, we can use more than one if-else statement in nested form. In the flowchart below we can see:

- If condition-1 is false the statement-3 will be executed, and condition-1 is true then the control is transferred to condition-2.
- If condition-2 is true, statement-1 will be executed; otherwise, statement-2 will be evaluated and then the control is transferred to another block of statement.

I) Form (A)

```
if(test expression 1)
{
    if(text expression 2)
    {
        statement(s);
    }else{
        statement(s);
    }
}else{
    statement(s);
}
```

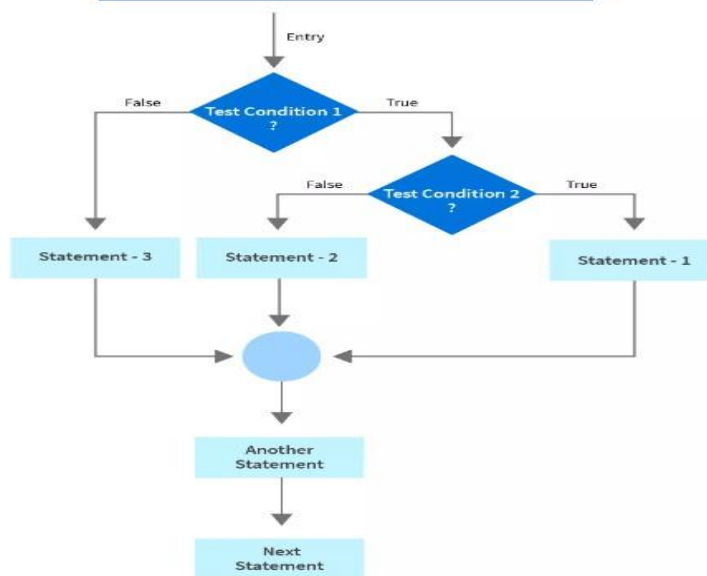
II) Form (B)

```
if(test expression 1)
{
    statement(s);
}else{
    if(test expression 2)
    {
        statement(s);
    }else{
        statement(s);
    }
}
```

III) Form (C)

```
if(test expression 1)
{
    if(test expression 2)
    {
        statement(s);
    }else{
        statement(s);
    }
}else{
    if(test expression 2)
    {
        statement(s);
    }else{
        statement(s);
    }
}
```

Flow Chart of nested If else Statements




```

#include <stdio.h>
int main()
{
    int n1, n2;
    printf("Input the value of n1:");
    scanf("%d", &n1);
    printf("Input the value of n2:");
    scanf("%d",&n2);
    if (n1 != n2)
    {
        printf("n1 is not equal to n2\n");
        //Nested if else
        if (n1 > n2)
        {
            printf("n1 is greater than n2\n");
        }
        else
        {
            printf("n2 is greater than n1\n");
        }
    }
    else
    {
        printf("n1 is equal to n2\n");
    }
    return 0;
}

```

Output:

```

Input the value of n1:90
Input the value of n2:80
n1 is not equal to n2
n1 is greater than n2

```

Else-if-Ladder

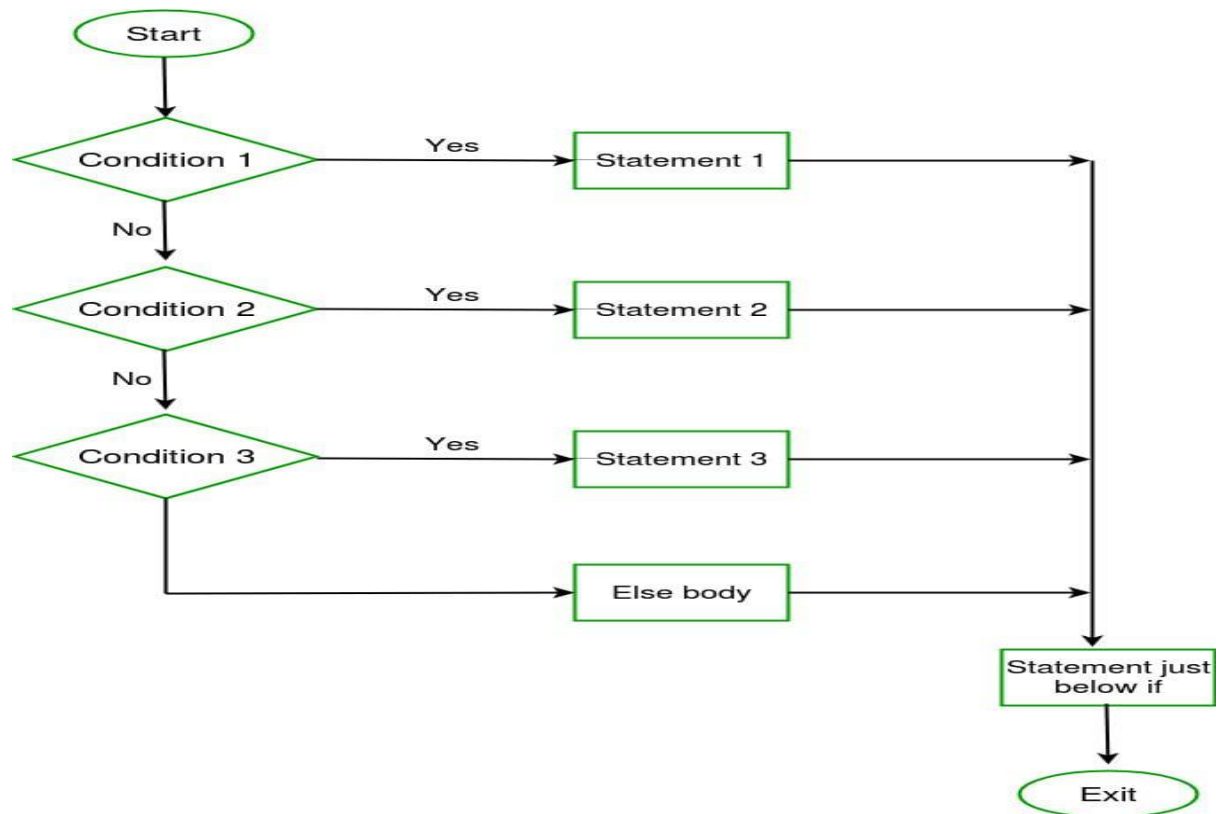
- There is another way of setting up if statement together when multi-way decisions are involved.
- A multi-way decision is a series of ifs in which the statement linked with each else statement is an if statement.
- The C if statements are executed from the top down.
- As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the C else-if ladder is bypassed.
- If none of the conditions is true, then the final else statement will be executed.
- if-else-if ladder is similar to switch statement.

Syntax:

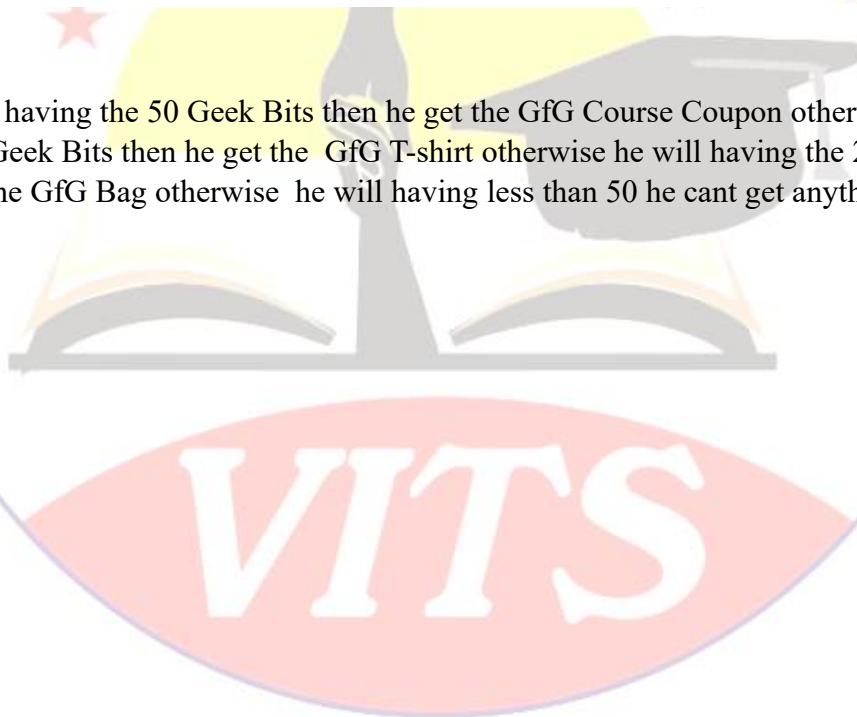
```

if (condition)
    statement;
else if (condition)
    statement;
.
.
else
    statement;

```

**Example:**

If the person having the 50 Geek Bits then he get the GfG Course Coupon otherwise he will having 100 Geek Bits then he get the GfG T-shirt otherwise he will having the 200 Geek Bits then he get the GfG Bag otherwise he will having less than 50 he cant get anything.




```
#include <stdio.h>

int main() {
    char button;
    printf("Input a character:");
    scanf("%c", &button);
    if(button == 'a')
    {
        printf("Hello");
    }
    else if(button == 'b')
    {
        printf("Namastey");
    }
    else if(button == 'c')
    {
        printf("Hola");
    }
    else if(button == 'd')
    {
        printf("Ciao");
    }
    else if(button == 'e')
    {
        printf("Salut");
    }
    else {
        printf("I am still learning more...");
    }
    return 0;
}
```

Output:

```
Input a character: e
Salut
```

Important Points Need to Remember

- Never put semicolon just after the if(expression).
- A non-zero value is considered as true and a zero(0) value is considered as false in C.
- We can use more than one condition inside the if statement using the logical operator.
- We should always use braces on separate lines to identify a block of statements.
- We should always align the opening and closing braces.
- Do not ignore placing parentheses for the if condition/expression.
- Be aware of dangling else statements.
- Avoid using operands that have side effects in a logical binary expression such as (a-- && ++b). The second operand may not be evaluated in any case.

Advantages and Disadvantages of C If Statement

Advantages

- It checks every condition, It also makes a program more robust by allowing only a portion of code to run if a condition has been met.
- If statements are required to make decisions in the programs. Technically, this could be done with loops and goto(break). But in reality, the if statement is most concise.

Disadvantages

During execution as it checks every condition:

- This makes it difficult to test the code.
- It is a little bit complex in terms of reading conditions of programs as compared to the switch case.
- It takes more time for each possible condition because it does fall through all the if statements compared to switch case.

Conclusion

- Using if statement we can control the flow of statement(s) in the program.
- There are four types of if Statement in c: simple if, if-else, nested if-else, and else-if ladder.
- in C, if statement supports two-way branching statement and multi-way branching statement.
- We can ignore the 'else' part of the program statement and we can simply show the result of the 'if' condition/expression in our program.

Interesting Fact

we can print "hello world" without using a single semicolon in the complete program, this is done with the help of if statement.

```
#include<stdio.h>

int main()
{
    if(printf("Hello World!!"))
        return 0;
}
```

Output

```
Hello World!!
```

Is it possible because the printf statement evaluates to true, and hence expression is executed.

From the above example, we also learn that the if or else block could be empty, and it is not necessary to add statements in the if or else block.

Switch Statement

- When you want to solve multiple option type problems, for example: Menu like program, where one value is associated with each option and you need to choose only one at a time, then switch statement is used.
- In the switch statement, we compare the condition value with multiple cases.
- When there is a match with any one of the cases, the block of code corresponding with that case is executed.
- Each case has a name or a number, which is known as its identifier.
- If none of the cases matches the condition, the block of code corresponding to the default case is executed.

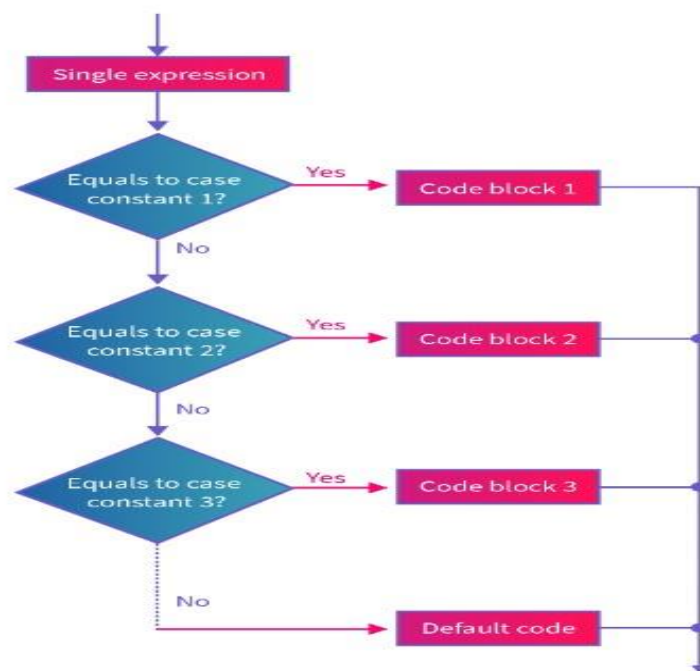
- Switch statement is a control statement that allows us to choose one choice among the many given choices.
- The expression in switch evaluates to return an integral value, which is then compared to the values present in different cases.
- It executes the block of code which matches the case value.
- If there is no match, then default block is executed (if present).

Syntax of the Switch Statement in C Programming Language.

```
switch(expression)
{
case value1:
    //code to be executed;
    break; //optional
case value2:
    //code to be executed;
    break; //optional
.....

default:
    //code to be executed if all cases are not matched;
}
```

Flowchart Representing the Working of the Switch Statement in C:



Rules for Switch Statement in C Language

Some essential and critical rules need to be followed when using the switch statement of the C programming language in your Program. The rules are listed below. Try to remember them and apply them when using the Switch statement.

1. The expression provided in the switch statement should always be either an integer value or a char value. You can also provide expressions, but the result of the expression should not return a Float or decimal value. The Switch statement will not accept float, double string, or other data types.

Eg:

```
switch(x==5) // this is valid.
switch(x/3) // this is not valid.
```

2. You can't use two case labels with the same value. Duplicate case values would throw an error. The case label must be unique.

Eg:

```
switch(x)
{
    case a:
        // set of statement
    case a:
        //set of statements
} // this kind of switch statement will throw you a compile-time error.
```

3. You can't define ranges with the case statement, nor can you use a single case label for more than one value. A case label can hold only one value.

Eg:

```
switch(x)
{
    case 1,10:
        // set of statements
    case 2-5:
        //set of statements
} // these 2 case statements will also throw a compile time error
```

4. The case label must end with a colon (:)
5. The next line, after the case statement, can be any valid C Statement.

Important Facts about Switch Case Statement in C:

You should be aware of some essential facts before getting your first hands-on experience with the switch statement of the C programming language. They are listed below:

- break statements are used to exit the switch block. It isn't necessary to use break after each block, but if you do not use it, then all the consecutive blocks of code will get executed after the matching block.

```

int i = 1;
switch(i)
{
    case 1:
        printf("A");           // No break
    case 2:
        printf("B");           // No break
    case 3:
        printf("C");
        break;
}

```

OUTPUT: A B C

- The default label for a switch statement is an optional one. You can use a switch statement without a default label, which would work fine.
- default case is executed when none of the mentioned case matches the switch expression. The default case can be placed anywhere in the switch case. Even if we don't include the default case, switch statement works.
- Just like the if-else statement, you can also nest more than one switch statement inside another. But this would make the code complex and less readable, so it is generally avoided.

Eg:

```

switch(x)
{
    switch(y) // this is valid
    {
        //set of statements
    }
}

```

Switch Statement in C Example

Lets write a simple C Program to design basic functionalities of a simple calculator that takes two integers, performs the requested operations, and display the output.


```

#include <stdio.h>
#include <conio.h>
void main()
{
    int choice;
    printf("Select an option from the list below:\n");
    printf("1. Addition\n");
    printf("2. Subtraction\n");
    printf("3. Multiplication\n");
    printf("4. Division\n");
    printf("5. Modulus\n");
    printf("6. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    int a, b;
    // read two numbers
    printf("Enter two numbers: ");
    scanf("%d %d", &a, &b);
    switch (choice)
    {
        case 1:
            printf("%d + %d = %d\n", a, b, a + b);
            break;
        case 2:
            printf("%d - %d = %d\n", a, b, a - b);
            break;
        case 3:
            printf("%d * %d = %d\n", a, b, a * b);
            break;
        case 4:
            printf("%d / %d = %d\n", a, b, a / b);
            break;
        case 5:
            printf("%d %% %d = %d\n", a, b, a % b);
            break;
        case 6:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice\n");
    }
}

```

The above Program will get the choice from the user as an integer input, then read two numbers also from the user, perform the requested operation and print the output back to the user.

The Program uses a switch statement to perform the decision-making. At the end of each case label, the break statement makes sure the control transfers out of the switch statement and does not execute the other statements sequentially.

The default statement does not have a break statement cause the control will automatically get out of the switch statement after the last line.

Input:

```

3
5 4

```

Output:

```

5 * 4 = 20

```

Conclusion:

- The switch statement takes an expression and, based on the result, will execute a statement set from a wide array of choices, unlike the if-else statement, which only allows you to choose between two choices.
- You can use any expression for the switch statement, but the expression should return a single value, and the resulting value should never be of type float or double.
- Only use constant values in the case statement; never use variables.
- Terminate each of the case statements with a break, if you want the compiler to get out of the switch statement after the execution of that case statement.

- Use the default statement when you want to execute a set of the statement when none of the case statements get executed.

Difference between switch and if

- If statements can evaluate float conditions. Switch statements cannot evaluate float conditions.
- If statements can evaluate relational operators. Switch statement cannot evaluate relational operators i.e., they are not allowed in switch statement.
- For numerous statements, you can use several if statements. For numerous statements in Switch, you only have one expression.
- It's difficult to make changes to if-else statements because it's time-consuming to figure out where the change needs to be made. Switch statements, on the other hand, are simple to change since they are easy to trace.
- If-else values are determined by constraints, whereas switch case values are determined by user preferences.
- If the condition inside the if block is false, the statement inside the else block is executed. If the condition inside the switch statement is false, the default statements are run.
- The if-else statement is used to choose between two options, but the switch case statement is used to choose between numerous options.

Some of The Similarities Between If-Else and Switch Case Statement

- They are both employed to control the flow of execution of the program
- They both evaluate a condition, and then the flow of the program depends on that
- Their syntax and way of representation differ but can be used for the same purpose

Ternary Operator

Using the Ternary operator in c is a way to shorten the if-else code block in C/C++.

Ternary Operator in C takes three arguments:

- The first argument in the Ternary Operator in C is the comparison condition.
- The second argument in the Ternary Operator in C is the result if the condition is true.
- The third argument in the Ternary Operator in C is the result if the condition is false.

So, according to the above three arguments in the ternary operator in c, we can say that the Ternary operator in C allows us to execute different code depending on the first argument, i.e. based on condition.

The symbol for the Ternary operator in C is ? :.

Syntax:

```
exp1 ? exp2 : exp3
```

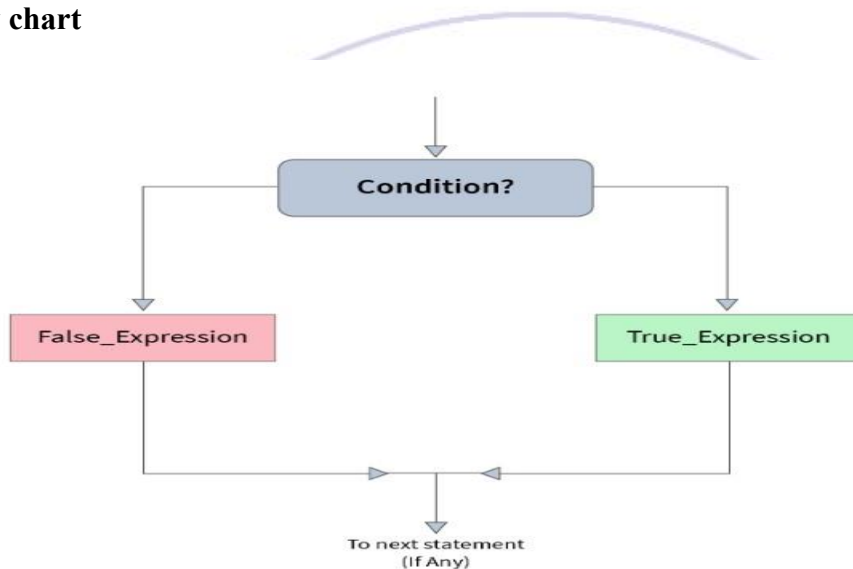
Working of Syntax:

- If the condition in the ternary operator is met (true), then the exp2 executes.
- If the condition is false, then the exp3 executes.

Example:

```
int mxNumber = 10 > 15 ? 10 : 15;
```

Flow chart



Ex. Find out the given number is even or not using ternary operator in c.

```
#include<stdio.h>
#include<stdlib.h>

int main()
{
    int num = 10;

    // condition to check number is even or not
    (num % 2 == 0) ? printf("Number is even") : printf("Number is not even");

    return 0;
}
```

Output:

```
Number is even
```

Jump Statement:

- Jump Statements interrupt the normal flow of the program while execution and jump when it gets satisfied given specific conditions.
- The main uses of just statements are to exit the loops like for, while, do-while also switch case and executes the given or next block of the code, skip the iterations of the loop, change the control flow to specific location, etc.

- Sometimes, while executing a loop, it becomes necessary to skip a part of the loop or to leave the loop as soon as certain conditions becomes true. This is known as jumping out of loop.

Types of Jump Statement in C

There are 4 types of jump statement in C Language.

1. Goto Statement
2. Break Statement
3. Continue Statement
4. Return Statement

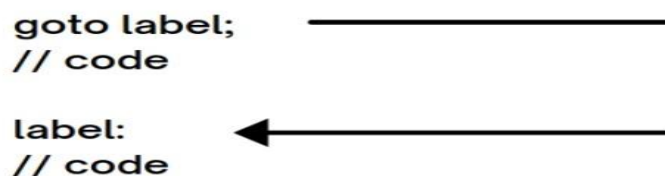
Goto Statement in C:

- The C Goto statement is a jump Statement which is sometimes also referred to as an unconditional jump statement.
- The goto statement can be used to jump from anywhere to anywhere within a function.
- The goto requires a label in order to identify the place where the branch is to be made.
- A label is any variable name, and must be followed by colon.
- The goto statement allows us to transfer control of the program to the specified label.

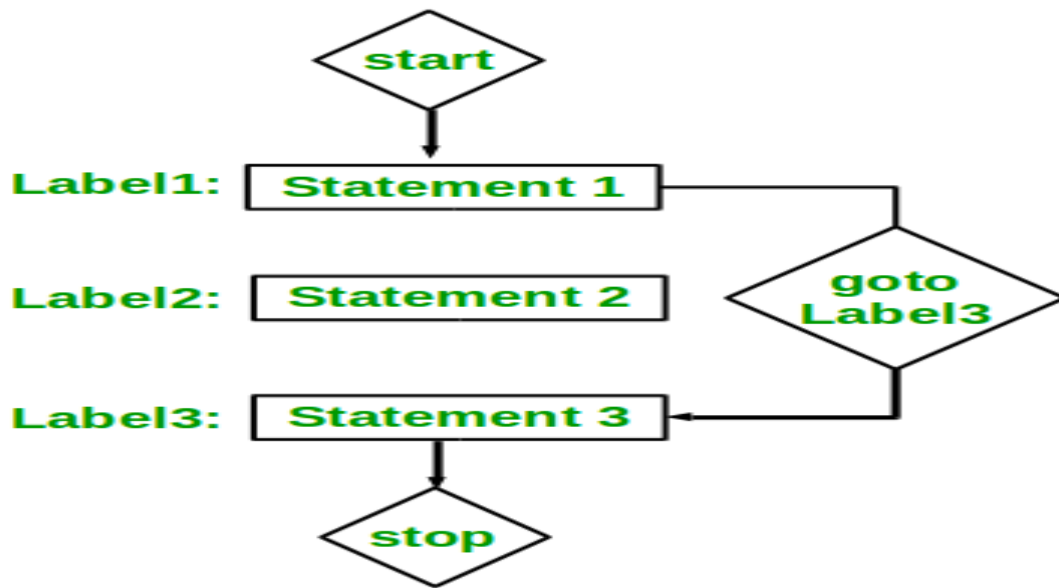
Syntax of goto Statement

Syntax1		Syntax2
goto label;		label:
.		.
.		.
.		.
label:		goto label;

- In the above syntax, the first line tells the compiler to go to or jump to the statement marked as a label.
- Here, the label is a user-defined identifier that indicates the target statement.
- The statement immediately followed after 'label:' is the destination statement.
- The 'label:' can also appear before the 'goto label;' statement in the above.



Control Flow of goto statement



Example:

Type 1: In this case, we will see a situation similar to as shown in syntax1 above. Suppose we need to write a program where we need to check if a number is even or not and print accordingly using the goto statement. The below program explains how to do this:

```

// C program to check if a number is
// even or not using goto statement
#include <stdio.h>

// function to check even or not
void checkEvenOrNot(int num)
{
    if (num % 2 == 0)
        // jump to even
        goto even;
    else
        // jump to odd
        goto odd;

even:
    printf("%d is even", num);
    // return if even
    return;
odd:
    printf("%d is odd", num);
}

int main()
{
    int num = 26;
    checkEvenOrNot(num);
    return 0;
}
  
```

Output:

26 is even

Type 2: In this case, we will see a situation to as shown in syntax2 above. Suppose we need to write a program that prints numbers from 1 to 10 using the goto statement. The below program explains how to do this.

```
// C program to print numbers
// from 1 to 10 using goto statement
#include <stdio.h>

// function to print numbers from 1 to 10
void printNumbers()
{
    int n = 1;
label:
    printf("%d ", n);
    n++;
    if (n <= 10)
        goto label;
}

// Driver program to test above function
int main()
{
    printNumbers();
    return 0;
}
```

Output:

```
1 2 3 4 5 6 7 8 9 10
```

Reasons to avoid goto

The use of goto statement may lead to code that is buggy and hard to follow. For example,

```
one:
for (i = 0; i < number; ++i)
{
    test += i;
    goto two;
}
two:
if (test > 5) {
    goto three;
}
... ..
```

Also, the goto statement allows you to do bad stuff such as jump out of the scope.

That being said, goto can be useful sometimes. For example: to break from nested loops.

Should you use goto?

If you think the use of goto statement simplifies your program, you can use it. That being said, goto is rarely useful and you can create any C program without using goto altogether.

Here's a quote from Bjarne Stroustrup, creator of C++, **"The fact that 'goto' can do anything is exactly why we don't use it."**

Disadvantage of using goto statement:

- The use of the goto statement is highly discouraged as it makes the program logic very complex.
- The use of goto makes tracing the flow of the program very difficult.
- The use of goto makes the task of analyzing and verifying the correctness of programs (particularly those involving loops) very difficult.
- The use of goto can be simply avoided by using break and continue statements.

Break Statement in C:

- The break statement is one of the four jump statement in the c language.
- The purpose of the break statement in c is for unconditional exit from the loop.
- Break Statement is used to break the loop or switch case statements execution and brings the control to the next block of code after that particular loop or switch case it was used in.
- Break statement are used to bring the program control out of the loop it was encountered in.
- The break statement is used inside loops or switch statements in C language.
- The break statement can only break out of a single loop at a time.

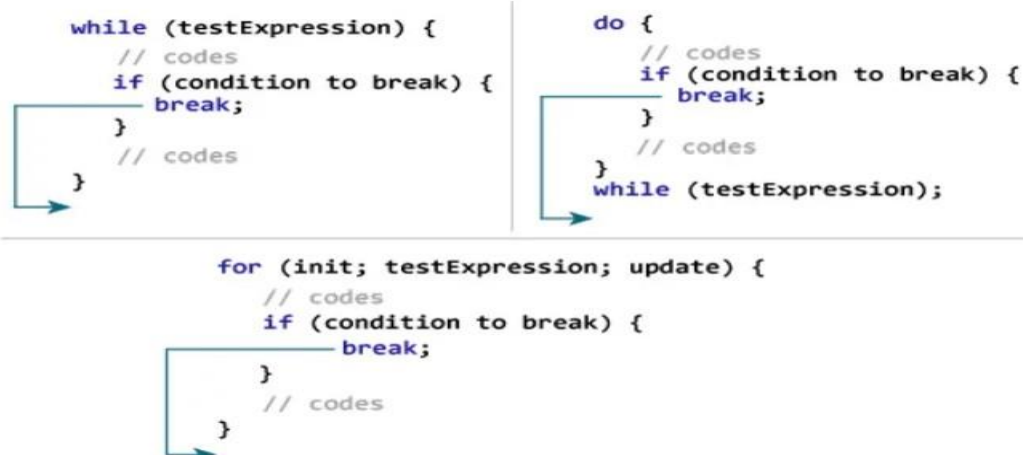
Syntax of break in C

```
break;
```

The break Statement ends the loop immediately when it is encountered.

The break statement is almost used with if...else statement inside the loop.

How break statement works?



Working of break in C

Use of Break in C:

- The break statement in C is used for breaking out of the loop.
- We can use it with any type of loop to bring the program control out of the loop.
- In C, we can use the break statement in the following ways:
 1. Simple loops
 2. Nested loops
 3. Infinite loops
 4. Switch case

Example of break in C

Example 1: C Program to use break Statement with Simple loops

Break statement in c can be used with simple loops i.e., for loops, while loops, and do-while loops.

```
// C Program to demonstrate break statement with for loop
#include <stdio.h>

int main()
{
    // using break inside for loop to terminate after 2
    // iteration
    printf("break in for loop\n");
    for (int i = 1; i < 5; i++) {
        if (i == 3) {
            break;
        }
        else {
            printf("%d ", i);
        }
    }

    // using break inside while loop to terminate after 2
    // iteration
    printf("\nbreak in while loop\n");
    int i = 1;
    while (i < 20) {
        if (i == 3)
            break;
        else
            printf("%d ", i);
        i++;
    }
    return 0;
}
```

Output

```
break in for loop
1 2
break in while loop
1 2
```

Example 2: C Program to use break Statement with Nested loops

Break statements can be used when working with nested loops. The control will come out of only that loop in which the break statement is used.

```
// C program to illustrate
// using break statement
// in Nested loops
#include <stdio.h>

int main()
{
    // nested for loops with break statement
    // at inner loop
    for (int i = 1; i <= 6; ++i) {
        for (int j = 1; j <= i; ++j) {
            if (i <= 4) {
                printf("%d ", j);
            }
            else {
                // if i > 4 then this innermost loop will
                // break
                break;
            }
        }
        printf("\n");
    }
    return 0;
}
```

Output

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

Note: Break statement only breaks out of one loop at a time. So if in nested loop, we have used break in inner loop, the control will come to outer loop instead of breaking out of all the loops at once. We will have to use multiple break statements if we want to break out of all the loops.

Example 3: C Program to use break Statement with Infinite Loops

An Infinite loop can be terminated with a break statement as a part of the condition.

```
// C Program to demonstrate infinite loop without using
// break statement
#include <stdio.h>

int main()
{
    int i = 0;

    // while loop which will always be true
    while (1) {
        printf("%d ", i);
        i++;
        if (i == 5) {
            break;
        }
    }
    return 0;
}
```

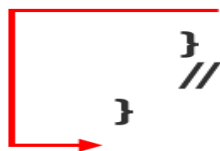
Output

```
0 1 2 3 4
```

In the above program, the loop condition is always true, which will cause the loop to execute indefinitely. This is corrected by using the break statement. Iteration of the loop is restricted to 5 iterations using break.

How break Statement Works?

```
for( init; condition; operation)
{
    // code
    if(condition to break)
    {
        break;
    }
    // code
}
```



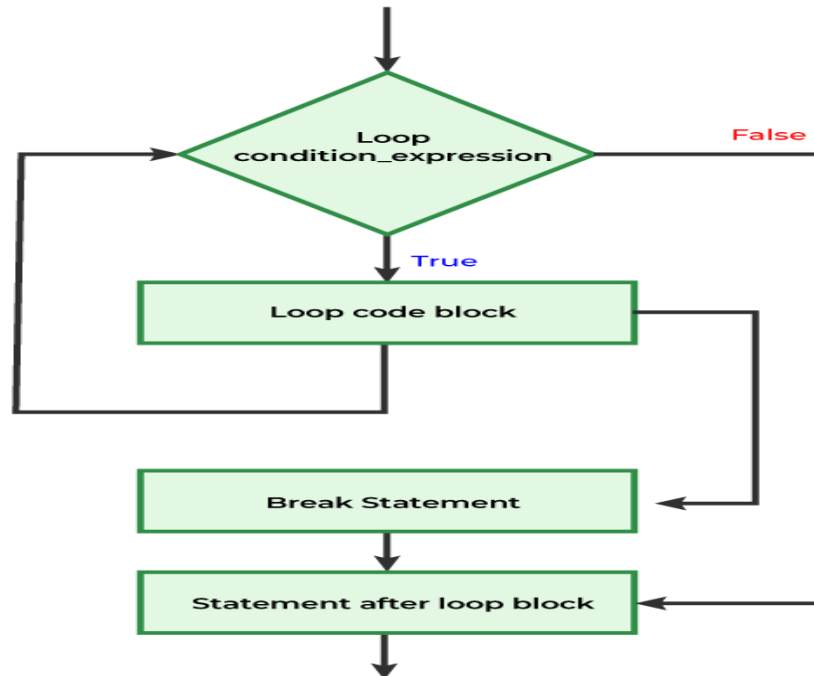
The working of the break statement in C is desired below:

1. Step 1: the loop execution starts after the test condition is evaluated.
2. Step 2: if the break condition is present the condition will be evaluated.
3. Step 3A: If the condition is true, the program control reaches the break statement and skips the further execution of the loop by jumping to the statements directly below the loop.

4. Step 3B: If the condition is false, the normal flow of the program control continues.

Flowchart of Break Statement:

Break Statement Flow Diagram



Break in C Switch Case :

- In general, the Switch case statement evaluates an expression, and depending on the value of the expression, it executes the statement associated with the value.
- Not only that, all the cases after the matching case after the matching case will also be executed.
- To prevent that, we can use the break statement in the switch case as shown:

Syntax of break in switch case

```

switch(expression)
{
  case value1:
    statement_1;
    break;

  case value2:
    statement_2;
    break;
  .....
  .....

  case value_n:
    statement_n;
    break;

  default:
    default statement;
}
  
```

Example of break in Switch Case:

```
// C Program to demonstrate working of break with switch
// case
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char c;
    float x, y;

    while (1) {
        printf("Enter an operator (+, -), if want to exit "
            "press x: ");
        scanf(" %c", &c);
        // to exit
        if (c == 'x')
            exit(0);

        printf("Enter Two Values:\n ");
        scanf("%f %f", &x, &y);

        switch (c) {
            // For Addition
            case '+':
                printf("%.1f + %.1f = %.1f\n", x, y, x + y);
                break;
            // For Subtraction
            case '-':
                printf("%.1f - %.1f = %.1f\n", x, y, x - y);
                break;
            default:
                printf(
                    "Error! please write a valid operator\n");
        }
    }
}
```

Output:

```
Enter an operator (+, -), if want to exit press x: +
Enter Two Values:
10
20
10.0 + 20.0 = 30.0
```

FAQs on C break Statement

1. What is the use of a break statement in C?

The break statement in C is used for early termination and exit from the loop.

2. What is the difference between break and continue?

The difference between the break and continue in C is listed in the below table:

break	continue
The break statement terminates the loop and brings the program control out of the loop.	The continue statement terminates only the current iteration and continues with the next iterations.
The syntax is: break;	The syntax is: continue;
The break can also be used in switch case.	Continue can only be used in loops.

C Continue:

- The continue statement in C is a jump statement that is used to bring the program control to the start of the loop.
- We can use the continue statement in the while loop, for loop, or do..while loop to alter the normal flow of the program execution.
- Unlike break, it cannot be used with a C switch case.
- The continue statement skips the current iteration of the loop and continues with the next iteration.
- Typically, the continue statement skips some code inside the loop and lets the program move on with the next iteration.
- It is mainly used for a condition so that we can skip some lines of code for a particular condition.
-

What is continue in C?

The C continue statement resets program control to the beginning of the loop when encountered. As a result, the current iteration of the loop gets skipped and the control moves on to the next iteration. Statements after the continue statement in the loop are not executed.

Syntax of continue in C

The syntax of continue is just the continue keyword placed wherever we want in the loop body.

```
continue;
```

Use of continue in C

The continue statement in C can be used in any kind of loop to skip the current iteration. In C, we can use it in the following types of loops

- Single Loops
- Nested Loops

Using continue in infinite loop is not useful as skipping the current iteration won't make a difference when the number of iterations is infinite.

Example of Continue In C:

Example 1: C Program to use Continue Statement in a single loop.

The continue statement can be used in for loop, while loop, and do-while loop.

```
// C program to explain the use
// of continue statement with for loop

#include <stdio.h>

int main()
{
    // for loop to print 1 to 8
    for (int i = 1; i <= 8; i++) {
        // when i = 4, the iteration will be skipped and for
        // will not be printed
        if (i == 4) {
            continue;
        }
        printf("%d ", i);
    }
    printf("\n");

    int i = 0;
    // while loop to print 1 to 8
    while (i < 8) {
        // when i = 4, the iteration will be skipped and for
        // will not be printed
        i++;
        if (i == 4) {
            continue;
        }
        printf("%d ", i);
    }
    return 0;
}
```

Output

```
1 2 3 5 6 7 8
1 2 3 5 6 7 8
```

Example 2: C Program to use continue in a nested loop

The continue statement will only work in a single loop at a time. So in case of nested loops, we can use the continue statement to skip the current iteration of the inner loop when using nested loops.

```
// C program to explain the use
// of continue statement with nested loops
#include <stdio.h>

int main()
{
    // outer loop with 3 iterations
    for (int i = 1; i <= 3; i++) {
        // inner loop to print integer 1 to 4
        for (int j = 0; j <= 4; j++) {
            // continue to skip printing number 3
            if (j == 3) {
                continue;
            }
            printf("%d ", j);
        }
        printf("\n");
    }
    return 0;
}
```

Output

```
0 1 2 4
0 1 2 4
0 1 2 4
```

The continue skips the current iteration of the inner loop when it executes in the above program. As a result, the program is controlled by the inner loop update expression. In this way, 3 is never displayed in the output.

How Continue statement works?

```

    for( init; condition; update)
    {
        // ...
        if(condition)
        {
            continue;
        }
        // ...
    }
  
```

```

while (testExpression) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
  
```

```

do {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
} while (testExpression);
  
```

```

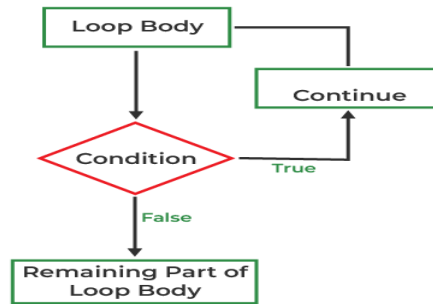
for (init; testExpression; update) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
  
```

Working of Continue in C

The working of the continue statement is as follows:

- STEP 1: The loop's execution starts after the loop condition is evaluated to be true.
- STEP 2: The condition of the continue statement will be evaluated.
- STEP 3A: If the condition is false, the normal execution will continue.
- STEP 3B: If the condition is true, the program control will jump to the start of the loop and all the statements below the continue will be skipped.
- STEP 4: Steps 1 to 4 will be repeated till the end of the loop.

Flowchart of continue in C



C break and continue statement differences

Break Statement: By using break statement, we terminate the smallest enclosing loop (e.g, a while, do-while, for, or switch statement).

Continue Statement: By using the continue statement, the loop statement is skipped and the next iteration takes place instead of the one prior.

Example: C program to demonstrate the difference between the working of break and continue statement in c.

```

// C program to demonstrate difference between
// continue and break
#include <stdio.h>

int main()
{
    printf("The loop with break produces output as: \n");

    for (int i = 1; i <= 7; i++) {
        // Program comes out of loop when
        // i becomes multiple of 3.
        if (i == 3)
            break;
        else
            printf("%d ", i);
    }

    printf("\nThe loop with continue produces output as: \n");
    for (int i = 1; i <= 7; i++) {
        // The loop prints all values except
        // those that are multiple of 3.
        if (i == 3)
            continue;
        printf("%d ", i);
    }
    return 0;
}
  
```

Output

```

The loop with break produces output as:
1 2
The loop with continue produces output as:
1 2 4 5 6 7
  
```

Explanation: In the above program, the first loop will print the value of i till 3 and will break the loop as we have used a break statement at i equal to 3. And in the second for loop program will continue but will not print the value of i when i will be equal to 3.

FAQs on C continue statement

1. What is the use of continue statement in C?

The continue statement in C is used in loops to skip the current iteration and move on to the next iteration without executing the statements below the continue in the loop body.

2. What type of statements are break and continue?

The break and continue in C are jump statements that are used to alter the flow of the normal execution of the loops.

Return Statement in C:

- The return statement is a type of jump statement in C which is used in a function to end it or terminate it immediately with or without value and returns the flow of program execution to the start from where it is called.
- The function declared with void type does not return any value.

Syntax:

```
return expression;  
or  
return;
```

Advantages and Disadvantages of Jump Statements in C

Advantages

- You can control the program flow or alter the follow of the program.
- Skipping the unnecessary code can be done using jump statements.
- You can decide when to break out of the loop using break statements.
- You get some sort of flexibility with your code using jump statements.

Disadvantages

- Readability of the code is disturbed because there are jumps from one part of the code to another part.
- Debugging becomes a little difficult.
- Modification of the code becomes difficult.

FAQs

1. What is the use of jump statements in C?

Jump Statements interrupt the normal flow of the program while execution and jump when it gets satisfied given specific conditions. It generally exits the loops and executes the given or next block of the code.

2. Why do we need Jump Function?

In between the loop, when we need to go to some other part of the code, jump statements shift the program control by breaking the loop, or if you need to skip the iteration in between the loop, the jump function does that.

It consists of the return statement by which you can return the values to the functions.

3. What is the main purpose of the goto function in C?

In the goto statement, we can specify where the program control should go in the current function by using the labels; you just need to specify the label name under which we want to shift the program control to.

4. Why should we avoid the goto statement in C?

If you are using goto, it is hard for the other person to trace the flow of the program and understand it. The program becomes hard to modify because it references the different labels, so rewriting is the only solution.

5. What is the rule of the continue function?

Continue statement skips the specific iteration in the loop and brings the program's control to the beginning of the loop. We could only use continue in loops.

6. Where do we use the continue function?

If you want a sequence to be executed but exclude a few iterations within the sequence, then the continue statement can be used.

For example, if you want to print 1 to 10 integers using the for loop and skip integer 5 from printing, then you can specify the condition and write the continue statement under it.
