Control Statements

If statement in C programming with example

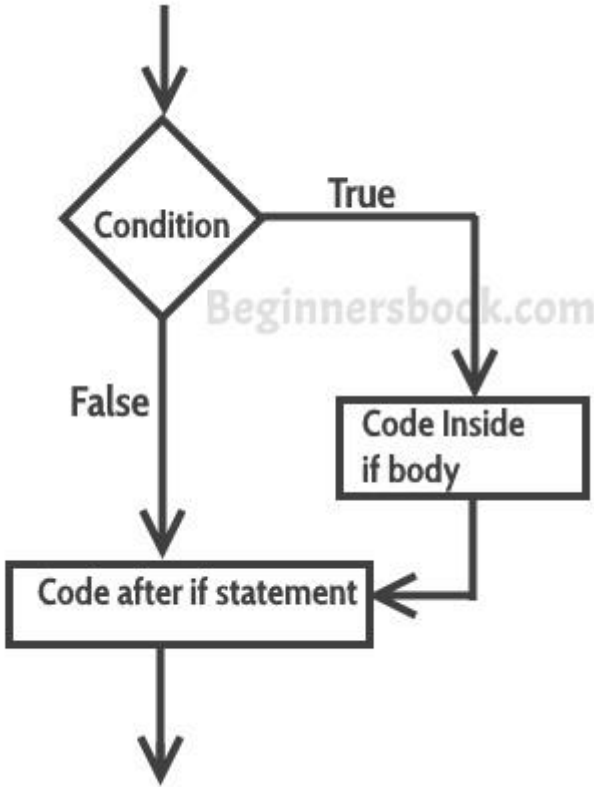
When we need to execute a block of statements only when a given condition is true then we use if statement. In the next tutorial, we will learn C if..else, nested if..else and else..if.

C – If statement

Syntax of if statement:
The statements inside the body of "if" only execute if the given condition returns true. If the condition returns false then the statements inside "if" are skipped.

```
if (condition)
{
    //Block of C statements here
    //These statements will only execute if the condition is true
}
```

Flow Diagram of if statement



Example of if statement

```
#include <stdio.h>
int main()
{
   int x = 20;
   int y = 22;
```

```
   if (x<y)
   {
      printf("Variable x is less than y");
   }
   return 0;
}
```

Output:

```
Variable x is less than y
```

Explanation: The condition (x<y) specified in the "if" returns true for the value of x and y, so the statement inside the body of if is executed.

Example of multiple if statements

We can use multiple if statements to check more than one conditions.

```
#include <stdio.h>
int main()
{
   int x, y;
   printf("enter the value of x:");
   scanf("%d", &x);
   printf("enter the value of y:");
   scanf("%d", &y);
   if (x>y)
   {
            printf("x is greater than y\n");
   }
   if (x<y)
   {
            printf("x is less than y\n");
   }
   if (x==y)
   {
            printf("x is equal to y\n");
   }
   printf("End of Program");
   return 0;
}
```

In the above example the output depends on the user input.

Output:

```
enter the value of x:20
enter the value of y:20
x is equal to y
End of Program
```

C – If..else, Nested If..else and else..if Statement with example

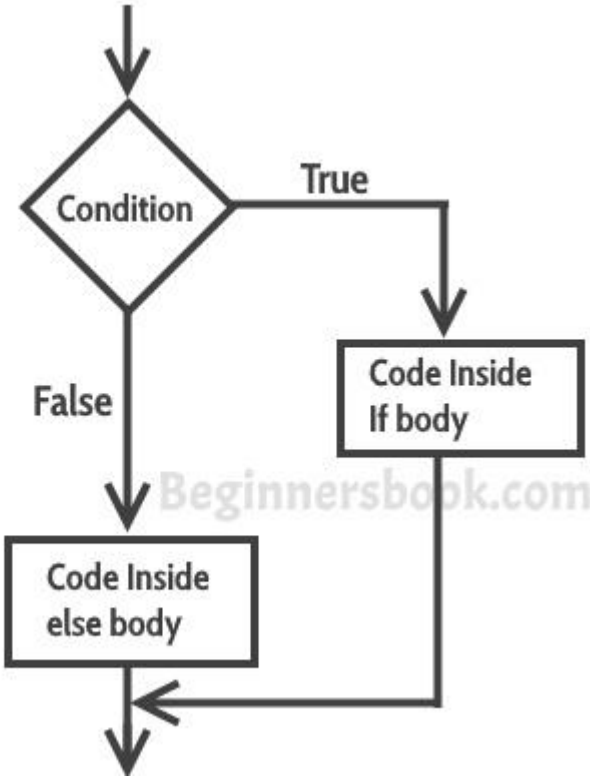
In the last tutorial we learned how to use if statement in C. In this guide, we will learn how to use if else, nested if else and else if statements in a C Program.

C If else statement

Syntax of if else statement:
If condition returns true then the statements inside the body of "if" are executed and the statements inside body of "else" are skipped.
If condition returns false then the statements inside the body of "if" are skipped and the statements in "else" are executed.

```
if(condition) {
  // Statements inside body of if
}
else {
  //Statements inside body of else
}
```

Flow diagram of if else statement



Example of if else statement

In this program user is asked to enter the age and based on the input, the if..else statement checks whether the entered age is greater than or equal to 18. If this condition meet then display message "You are eligible for voting", however if the condition doesn't meet then display a different message "You are not eligible for voting".

```
#include <stdio.h>
int main()
{
  int age;
  printf("Enter your age:");
  scanf("%d",&age);
  if(age >=18)
  {
        /* This statement will only execute if the
         * above condition (age>=18) returns true
         */
        printf("You are eligible for voting");
  }
  else
  {
        /* This statement will only execute if the
         * condition specified in the "if" returns false.
         */
        printf("You are not eligible for voting");
  }
  return 0;
}
```

Output:

```
Enter your age:14
You are not eligible for voting
```

Note: If there is **only one statement** is present in the “if” or “else” body then you do not need to use the braces (parenthesis). For example the above program can be rewritten like this:

```
#include <stdio.h>
int main()
{
  int age;
  printf("Enter your age:");
  scanf("%d",&age);
  if(age >=18)
        printf("You are eligible for voting");
  else
        printf("You are not eligible for voting");
  return 0;
}
```

C Nested If..else statement

When an if else statement is present inside the body of another “if” or “else” then this is called nested if else.

Syntax of Nested if else statement:

```
if(condition) {
```

```
   //Nested if else inside the body of "if"
   if(condition2) {
     //Statements inside the body of nested "if"
   }
   else {
     //Statements inside the body of nested "else"
   }
}
else {
   //Statements inside the body of "else"
}
```

Example of nested if..else

```
#include <stdio.h>
int main()
{
  int var1, var2;
  printf("Input the value of var1:");
  scanf("%d", &var1);
  printf("Input the value of var2:");
  scanf("%d",&var2);
  if (var1 != var2)
  {
        printf("var1 is not equal to var2\n");
        //Nested if else
        if (var1 > var2)
        {
              printf("var1 is greater than var2\n");
        }
        else
        {
              printf("var2 is greater than var1\n");
        }
  }
  else
  {
        printf("var1 is equal to var2\n");
  }
  return 0;
}
```

Output:

```
Input the value of var1:12
Input the value of var2:21
var1 is not equal to var2
var2 is greater than var1
```

C – else..if statement

The else..if statement is useful when you need to check multiple conditions within the program, nesting of if-else blocks can be avoided using else..if statement.

Syntax of else..if statement:

```
if (condition1)
{
  //These statements would execute if the condition1 is true
}
else if(condition2)
{
  //These statements would execute if the condition2 is true
}
else if (condition3)
{
  //These statements would execute if the condition3 is true
}
.
.
else
{
  //These statements would execute if all the conditions return false.
}
```

C – for loop in C programming with example

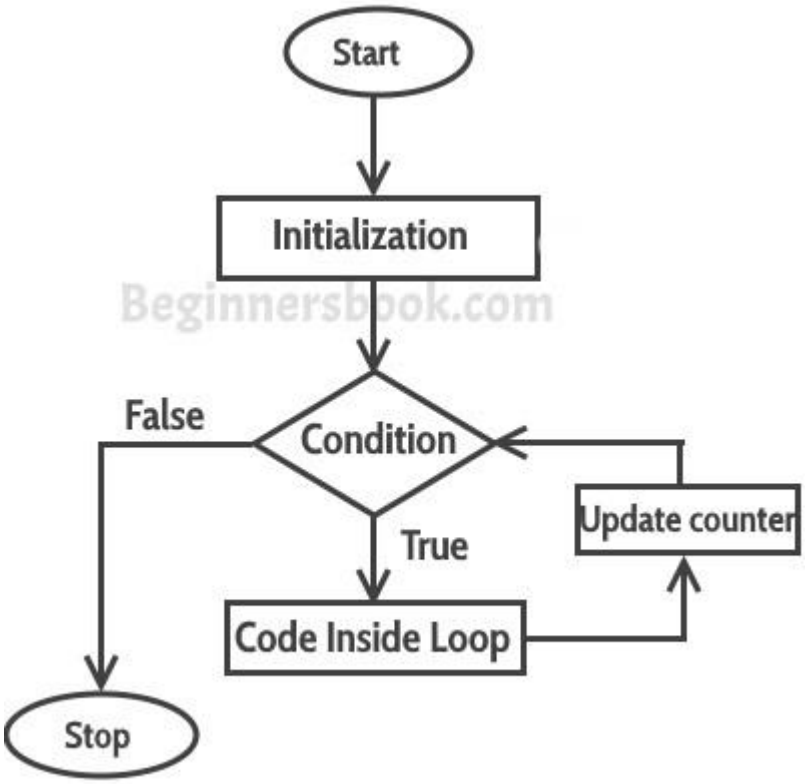
A loop is used for executing a block of statements repeatedly until a given condition returns false.

C For loop

This is one of the most frequently used loop in C programming.
Syntax of for loop:

```
for (initialization; condition test; increment or decrement)
{
      //Statements to be executed repeatedly
}
```

Flow Diagram of For loop



Step 1: First initialization happens and the counter variable gets initialized.
Step 2: In the second step the condition is checked, where the counter variable is tested for the given condition, if the condition returns true then the C statements inside the body of for loop gets executed, if the condition returns false then the for loop gets terminated and the control comes out of the loop.
Step 3: After successful execution of statements inside the body of loop, the counter variable is incremented or decremented, depending on the operation (++ or –).

Example of For loop

```
#include <stdio.h>
int main()
{
  int i;
  for (i=1; i<=3; i++)
  {
     printf("%d\n", i);
  }
  return 0;
}
```

Output:

```
1
2
3
```

Various forms of for loop in C

I am using variable num as the counter in all the following examples –
1) Here instead of num++, I'm using num=num+1 which is same as num++.

```
for (num=10; num<20; num=num+1)
```

2) Initialization part can be skipped from loop as shown below, the counter variable is declared before the loop.

```
int num=10;
for (;num<20;num++)
```

Note: Even though we can skip initialization part but semicolon (;) before condition is must, without which you will get compilation error.
3) Like initialization, you can also skip the increment part as we did below. In this case semicolon (;) is must after condition logic. In this case the increment or decrement part is done inside the loop.

```
for (num=10; num<20; )
{
    //Statements
    num++;
}
```

4) This is also possible. The counter variable is initialized before the loop and incremented inside the loop.

```
int num=10;
for (;num<20;)
{
    //Statements
    num++;
}
```

5) As mentioned above, the counter variable can be decremented as well. In the below example the variable gets decremented each time the loop runs until the condition num>10 returns false.

```
for(num=20; num>10; num--)
```

Nested For Loop in C

Nesting of loop is also possible. Lets take an example to understand this:

```
#include <stdio.h>
int main()
{
  for (int i=0; i<2; i++)
  {
            for (int j=0; j<4; j++)
            {
              printf("%d, %d\n",i ,j);
            }
  }
```

```
  return 0;
}
```
Output:

```
0, 0
0, 1
0, 2
0, 3
1, 0
1, 1
1, 2
1, 3
```
In the above example we have a for loop inside another for loop, this is called nesting of loops. One of the example where we use nested for loop is Two dimensional array.

Multiple initialization inside for Loop in C

We can have multiple initialization in the for loop as shown below.

```
for (i=1,j=1;i<10 && j<10; i++, j++)
```
What's the difference between above for loop and a simple for loop?
1. It is initializing two variables. Note: both are separated by comma (,).
2. It has two test conditions joined together using AND (&&) logical operator. Note: You cannot use multiple test conditions separated by comma, you must use logical operator such as && or || to join conditions.
3. It has two variables in increment part. **Note:** Should be separated by comma.

Example of for loop with multiple test conditions

```
#include <stdio.h>
int main()
{
  int i,j;
  for (i=1,j=1 ; i<3 || j<5; i++,j++)
  {
          printf("%d, %d\n",i ,j);
  }
  return 0;
}
```

C – while loop in C programming with example

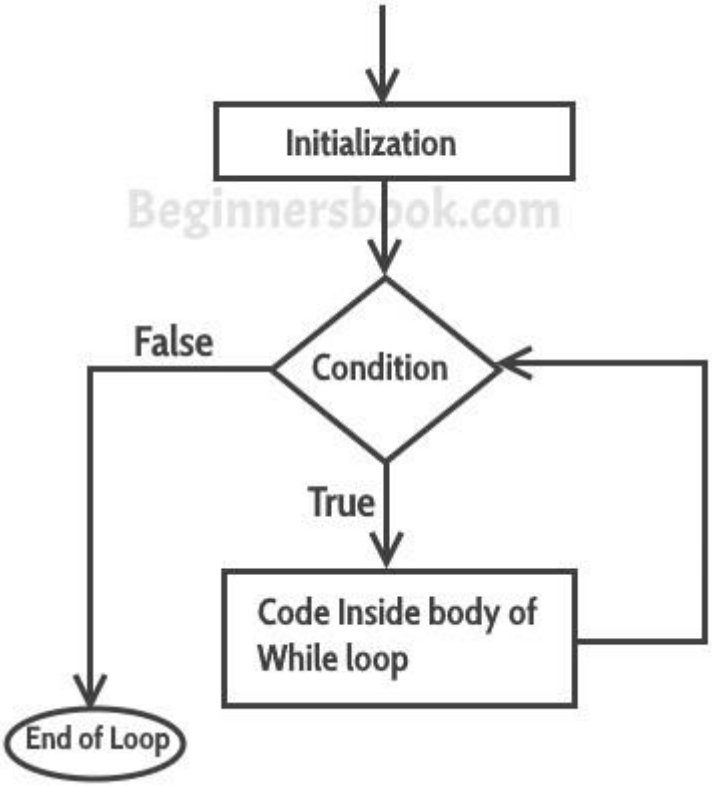
A loop is used for executing a block of statements repeatedly until a given condition returns false. In the previous tutorial we learned for loop. In this guide we will learn while loop in C.

C – while loop

Syntax of while loop:

```
while (condition test)
{
    //Statements to be executed repeatedly
    // Increment (++) or Decrement (--) Operation
}
```

Flow Diagram of while loop



Example of while loop

```
#include <stdio.h>
int main()
{
  int count=1;
  while (count <= 4)
  {
            printf("%d ", count);
            count++;
  }
  return 0;
}
```

Output:

```
1 2 3 4
```
step1: The variable count is initialized with value 1 and then it has been tested for the condition.
step2: If the condition returns true then the statements inside the body of while loop are executed else control comes out of the loop.
step3: The value of count is incremented using ++ operator then it has been tested again for the loop condition.

Guess the output of this while loop

```
#include <stdio.h>
int main()
{
    int var=1;
    while (var <=2)
    {
      printf("%d ", var);
    }
}
```
The program is an example of **infinite while loop**. Since the value of the variable var is same (there is no ++ or – operator used on this variable, inside the body of loop) the condition var<=2 will be true forever and the loop would never terminate.

Examples of infinite while loop

Example 1:

```
#include <stdio.h>
int main()
{
    int var = 6;
    while (var >=5)
    {
      printf("%d", var);
      var++;
    }
  return 0;
}
```
Infinite loop: var will always have value >=5 so the loop would never end.

Example 2:

```
#include <stdio.h>
int main()
{
   int var =5;
   while (var <=10)
   {
```

```
      printf("%d", var);
      var--;
   }
   return 0;
}
```

Infinite loop: var value will keep decreasing because of –- operator, hence it will always be <= 10.

Use of Logical operators in while loop

Just like relational operators (<, >, >=, <=, ! =, ==), we can also use logical operators in while loop. The following scenarios are valid :

```
while(num1<=10 && num2<=10)
```

-using AND(&&) operator, which means both the conditions should be true.

```
while(num1<=10||num2<=10)
```

– OR(||) operator, this loop will run until both conditions return false.

```
while(num1!=num2 &&num1 <=num2)
```

– Here we are using two logical operators NOT (!) and AND(&&).

```
while(num1!=10 ||num2>=num1)
```

Example of while loop using logical operator

In this example we are testing multiple conditions using logical operator inside while loop.

```
#include <stdio.h>
int main()
{
  int i=1, j=1;
  while (i <= 4 || j <= 3)
  {
            printf("%d %d\n",i, j);
            i++;
            j++;
  }
  return 0;
}
```

Output:

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

C – do while loop in C programming with example

In the previous tutorial we learned while loop in C. A do while loop is similar to while loop with one exception that it executes the statements inside the body of do-while before checking the condition. On the other hand in the while loop, first the condition is checked and then the statements in while loop are executed. So you can say that if a condition is false at the first place then the do while would run once, however the while loop would not run at all.

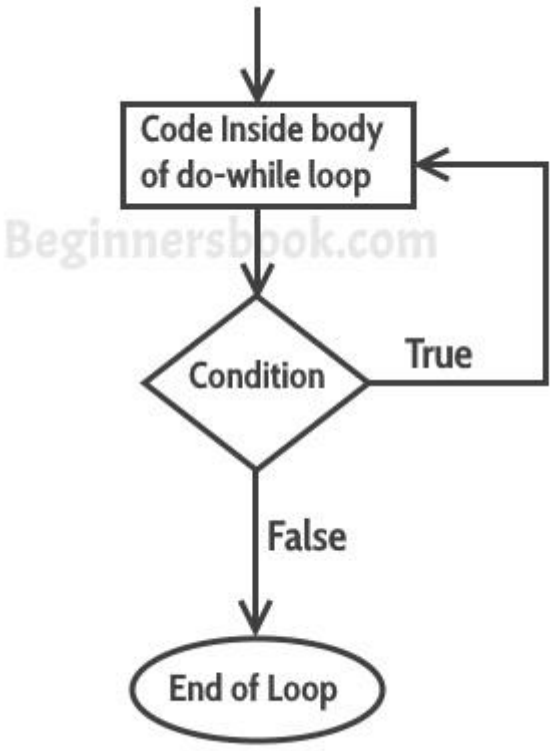
C – do..while loop

Syntax of do-while loop

```
do
{
   //Statements

}while(condition test);
```

Flow diagram of do while loop



Example of do while loop

```
#include <stdio.h>
int main()
{
        int j=0;
```

```
        do
        {
                printf("Value of variable j is: %d\n", j);
                j++;
        }while (j<=3);
        return 0;
}
```

Output:

```
Value of variable j is: 0
Value of variable j is: 1
Value of variable j is: 2
Value of variable j is: 3
```

While vs do..while loop in C

Using while loop:

```
#include <stdio.h>
int main()
{
   int i=0;
   while(i==1)
   {
        printf("while vs do-while");
   }
   printf("Out of loop");
}
```

Output:

```
Out of loop
```

Same example using do-while loop

```
#include <stdio.h>
int main()
{
  int i=0;
  do
  {
        printf("while vs do-while\n");
  }while(i==1);
  printf("Out of loop");
}
```

Output:

```
while vs do-while
Out of loop
```

Explanation: As I mentioned in the beginning of this guide that do-while runs at least once even if the condition is false because the condition is evaluated, after the execution of the body of loop.

C – continue statement with example

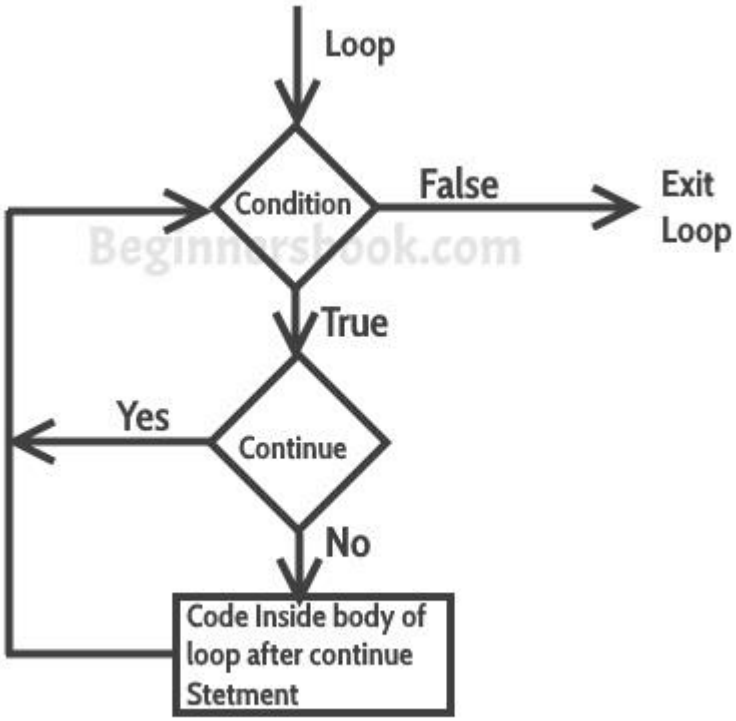
The **continue statement** is used inside loops. When a continue statement is encountered inside a loop, control jumps to the beginning of the loop for next iteration, skipping the execution of statements inside the body of loop for the current iteration.

C – Continue statement

Syntax:

```
continue;
```

Flow diagram of continue statement



Example: continue statement inside for loop

```
#include <stdio.h>
int main()
{
  for (int j=0; j<=8; j++)
  {
    if (j==4)
    {
```

```
            /* The continue statement is encountered when
             * the value of j is equal to 4.
             */
            continue;
    }

    /* This print statement would not execute for the
          * loop iteration where j ==4  because in that case
          * this statement would be skipped.
          */
    printf("%d ", j);
  }
  return 0;
}
```

Output:

```
0 1 2 3 5 6 7 8
```

Value 4 is missing in the output, why? When the value of variable j is 4, the program encountered a continue statement, which makes the control to jump at the beginning of the for loop for next iteration, skipping the statements for current iteration (that's the reason printf didn't execute when j is equal to 4).

Example: Use of continue in While loop

In this example we are using continue inside while loop. When using while or do-while loop you need to place an increment or decrement statement just above the continue so that the counter value is changed for the next iteration. For example, if we do not place counter– statement in the body of "if" then the value of counter would remain 7 indefinitely.

```
#include <stdio.h>
int main()
{
   int counter=10;
   while (counter >=0)
   {
            if (counter==7)
            {
                counter--;
                continue;
            }
            printf("%d  ", counter);
            counter--;
   }
   return 0;
}
```

Output:

```
10 9 8 6 5 4 3 2 1 0
```

The print statement is skipped when counter value was 7.

Another Example of continue in do-While loop

```
#include <stdio.h>
int main()
{
  int j=0;
  do
  {
    if (j==7)
    {
      j++;
      continue;
    }
    printf("%d ", j);
    j++;
  }while(j<10);
  return 0;
}
```

Output:

```
0 1 2 3 4 5 6 8 9
```

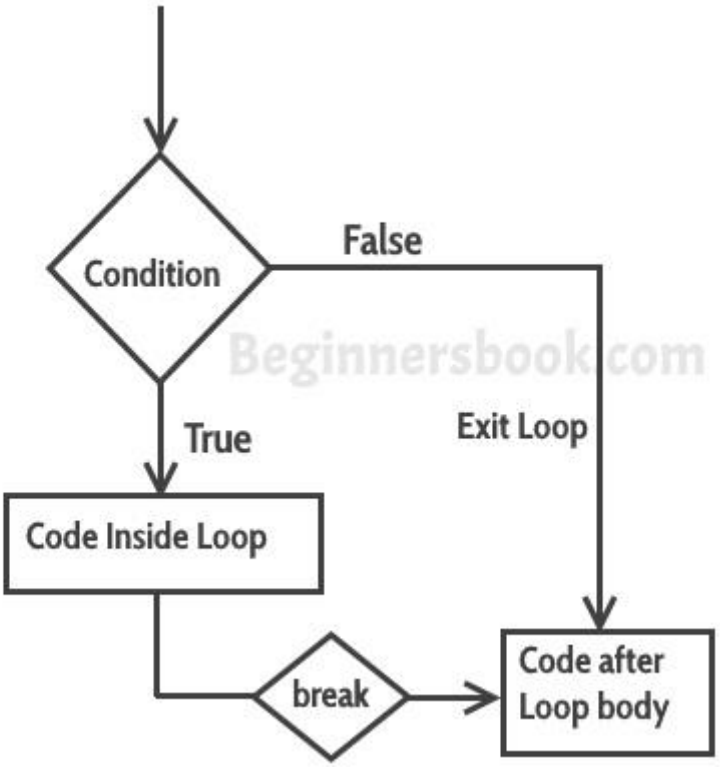
C – break statement in C programming

The break statement is used inside loops and switch case.

C – break statement

1. It is used to come out of the loop instantly. When a break statement is encountered inside a loop, the control directly comes out of loop and the loop gets terminated. It is used with if statement, whenever used inside loop.
2. This can also be used in switch case control structure. Whenever it is encountered in switch-case block, the control comes out of the switch-case(see the example below).

Flow diagram of break statement



Syntax:

```
break;
```

Example – Use of break in a while loop

```
#include <stdio.h>
int main()
{
    int num =0;
    while(num<=100)
    {
       printf("value of variable num is: %d\n", num);
       if (num==2)
       {
          break;
       }
       num++;
    }
    printf("Out of while-loop");
    return 0;
}
```

Output:

```
value of variable num is: 0
value of variable num is: 1
value of variable num is: 2
```

```
Out of while-loop
```

Example – Use of break in a for loop

```
#include <stdio.h>
int main()
{
     int var;
     for (var =100; var>=10; var --)
     {
          printf("var: %d\n", var);
          if (var==99)
          {
             break;
          }
     }
    printf("Out of for-loop");
    return 0;
}
```

Output:

```
var: 100
var: 99
Out of for-loop
```

Example – Use of break statement in switch-case

```
#include <stdio.h>
int main()
{
     int num;
     printf("Enter value of num:");
     scanf("%d",&num);
     switch (num)
     {
        case 1:
          printf("You have entered value 1\n");
          break;
        case 2:
          printf("You have entered value 2\n");
          break;
        case 3:
          printf("You have entered value 3\n");
          break;
        default:
          printf("Input value is other than 1,2 & 3 ");
    }
    return 0;
}
```

Output:

```
Enter value of num:2
```

```
You have entered value 2
```

You would always want to use break statement in a switch case block, otherwise once a case block is executed, the rest of the subsequent case blocks will execute. For example, if we don't use the break statement after every case block then the output of this program would be:

```
Enter value of num:2
You have entered value 2
You have entered value 3
Input value is other than 1,2 & 3
```

C – switch case statement in C Programming with example

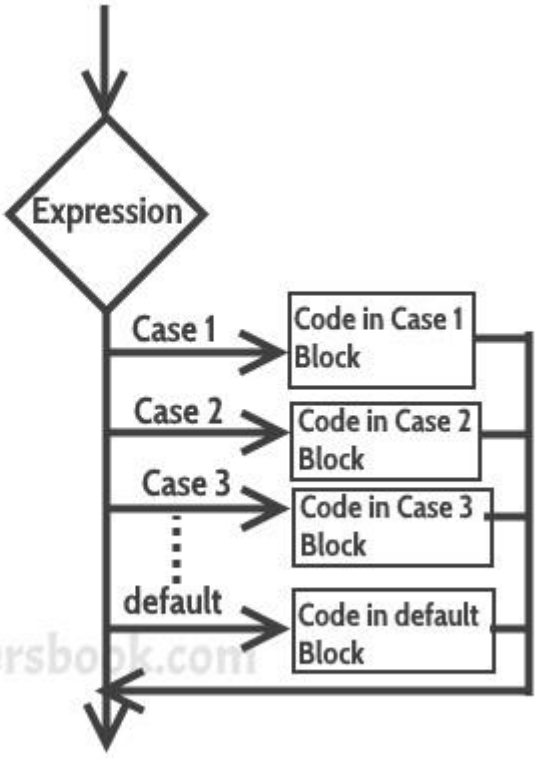
The **switch case statement** is used when we have multiple options and we need to perform a different task for each option.

C – Switch Case Statement

Before we see how a switch case statement works in a C program, let's checkout the syntax of it.

```
switch (variable or an integer expression)
{
    case constant:
    //C Statements
    ;
    case constant:
    //C Statements
    ;
    default:
    //C Statements
    ;
}
```

Flow Diagram of Switch Case



Example of Switch Case in C

Let's take a simple example to understand the working of a switch case statement in C program.

```
#include <stdio.h>
int main()
{
    int num=2;
    switch(num+2)
    {
        case 1:
          printf("Case1: Value is: %d", num);
        case 2:
          printf("Case1: Value is: %d", num);
        case 3:
          printf("Case1: Value is: %d", num);
        default:
          printf("Default: Value is: %d", num);
   }
   return 0;
}
```

Output:

```
Default: value is: 2
```

Explanation: In switch I gave an expression, you can give variable also. I gave num+2, where num value is 2 and after addition the expression resulted 4. Since there is no case defined with value 4 the default case is executed.

Twist in a story – Introducing Break statement

Before we discuss more about break statement, guess the output of this C program.

```
#include <stdio.h>
int main()
{
    int i=2;
    switch (i)
    {
       case 1:
          printf("Case1 ");
       case 2:
          printf("Case2 ");
       case 3:
          printf("Case3 ");
       case 4:
          printf("Case4 ");
       default:
          printf("Default ");
    }
   return 0;
}
```

Output:

```
Case2 Case3 Case4 Default
```

I passed a variable to switch, the value of the variable is 2 so the control jumped to the case 2, However there are no such statements in the above program which could break the flow after the execution of case 2. That's the reason after case 2, all the subsequent cases and default statements got executed.

How to avoid this situation?
We can use break statement to break the flow of control after every case block.

Break statement in Switch Case

Break statements are useful when you want your program-flow to come out of the switch body. Whenever a break statement is encountered in the switch body, the control comes out of the switch case statement.

Example of Switch Case with break
I'm taking the same above that we have seen above but this time we are using break.

```
#include <stdio.h>
int main()
{
    int i=2;
    switch (i)
    {
        case 1:
          printf("Case1 ");
          break;
        case 2:
          printf("Case2 ");
          break;
        case 3:
          printf("Case3 ");
          break;
        case 4:
          printf("Case4 ");
          break;
        default:
          printf("Default ");
    }
    return 0;
}
```

Output:

```
Case 2
```

Why didn't I use break statement after default?

The control would itself come out of the switch after default so I didn't use it, however if you want to use the break after default then you can use it, there is no harm in doing that.

Few Important points regarding Switch Case

1) Case doesn't always need to have order 1, 2, 3 and so on. They can have any integer value after case keyword. Also, case doesn't need to be in an ascending order always, you can specify them in any order as per the need of the program.

2) You can also use characters in switch case. for example –

```
#include <stdio.h>
int main()
{
    char ch='b';
    switch (ch)
    {
       case 'd':
         printf("CaseD ");
         break;
       case 'b':
         printf("CaseB");
```

```
            break;
        case 'c':
            printf("CaseC");
            break;
        case 'z':
            printf("CaseZ ");
            break;
        default:
            printf("Default ");
    }
    return 0;
}
```

Output:

```
CaseB
```

3) The expression provided in the switch should result in a constant value otherwise it would not be valid.
For example:
Valid expressions for switch –

```
switch(1+2+23)
switch(1*2+3%4)
```

Invalid switch expressions –

```
switch(ab+cd)
switch(a+b+c)
```

4) Nesting of switch statements are allowed, which means you can have switch statements inside another switch. However nested switch statements should be avoided as it makes program more complex and less readable.
5) Duplicate case values are not allowed. For example, the following program is incorrect:
This program is **wrong** because we have two case ‘A’ here which is wrong as we cannot have duplicate case values.

```
#include <stdio.h>
int main()
{
    char ch='B';
    switch (ch)
    {
        case 'A':
            printf("CaseA");
            break;
        case 'A':
            printf("CaseA");
            break;
        case 'B':
            printf("CaseB");
            break;
```

```
        case 'C':
          printf("CaseC ");
          break;
        default:
          printf("Default ");
    }
    return 0;
}
```

6) The **default** statement is optional, if you don't have a default in the program, it would run just fine without any issues. However it is a good practice to have a default statement so that the default executes if no case is matched. This is especially useful when we are taking input from user for the case choices, since user can sometime enter wrong value, we can remind the user with a proper error message that we can set in the default statement.

C – goto statement with example

The goto statement is rarely used because it makes program confusing, less readable and complex. Also, when this is used, the control of the program won't be easy to trace, hence it makes testing and debugging difficult.

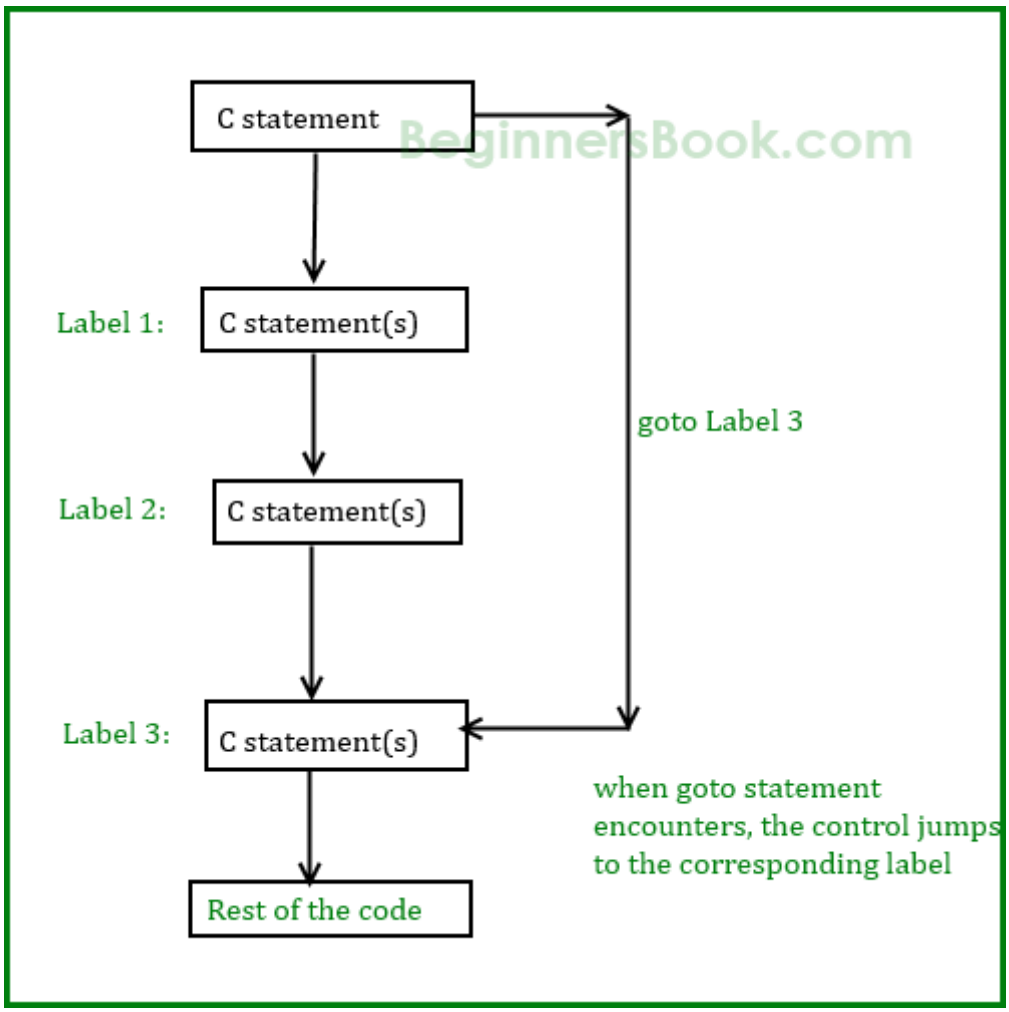
C – goto statement

When a goto statement is encountered in a C program, the control jumps directly to the label mentioned in the goto stateemnt

Syntax of goto statement in C

```
goto label_name;
..
..
label_name: C-statements
```

Flow Diagram of goto



Example of goto statement

```
#include <stdio.h>
int main()
{
  int sum=0;
  for(int i = 0; i<=10; i++){
            sum = sum+i;
            if(i==5){
               goto addition;
            }
  }

  addition:
  printf("%d", sum);

  return 0;
}
```

Output:

```
15
```

Explanation: In this example, we have a label addition and when the value of i (inside loop) is equal to 5 then we are jumping to this label using goto. This is reason the sum is displaying the sum of numbers till 5 even though the loop is set to run from 0 to 10.