Operators in C Language

C language supports a rich set of built-in operators. An operator is a symbol that tells the compiler to perform a certain mathematical or logical manipulation. Operators are used in programs to manipulate data and variables.

C operators can be classified into following types:

- Arithmetic operators
- Relational operators
- Logical operators
- Bitwise operators
- Assignment operators
- Conditional operators
- Special operators

Arithmetic operators

C supports all the basic arithmetic operators. The following table shows all the basic arithmetic operators.

Operator	Description
+	adds two operands
-	subtract second operands from first
*	multiply two operand
/	divide numerator by denominator
%	remainder of division
++	Increment operator - increases integer value by one
--	Decrement operator - decreases integer value by one

Relational operators

The following table shows all relation operators supported by C.

Operator	Description
==	Check if two operand are equal
!=	Check if two operand are not equal.
>	Check if operand on the left is greater than operand on the right
<	Check operand on the left is smaller than right operand
>=	check left operand is greater than or equal to right operand
<=	Check if operand on left is smaller than or equal to right operand

---

Logical operators

C language supports following 3 logical operators. Suppose `a = 1` and `b = 0`,

Operator	Description	Example
&&	Logical AND	(a && b) is false
||	Logical OR	(a || b) is true
!	Logical NOT	(!a) is false

Bitwise operators

Bitwise operators perform manipulations of data at **bit level**. These operators also perform **shifting of bits** from right to left. Bitwise operators are not applied to `float` or `double`(These are datatypes, we will learn about them in the next tutorial).

Operator	Description

&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	left shift
>>	right shift

Now lets see truth table for bitwise `&`, `|` and `^`

a	b	a & b	a | b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

The bitwise **shift** operator, shifts the bit value. The left operand specifies the value to be shifted and the right operand specifies the number of positions that the bits in the value have to be shifted. Both operands have the same precedence.

Example :

```
a = 0001000
b = 2
a << b = 0100000
a >> b = 0000010
```

Assignment Operators

Assignment operators supported by C language are as follows.

Operator	Description	Example
=	assigns values from right side operands to left side operand	a=b

+=	adds right operand to the left operand and assign the result to left	a+=b is same as a=a+b
-=	subtracts right operand from the left operand and assign the result to left operand	a-=b is same as a=a-b
*=	mutiply left operand with the right operand and assign the result to left operand	a*=b is same as a=a*b
/=	divides left operand with the right operand and assign the result to left operand	a/=b is same as a=a/b
%=	calculate modulus using two operands and assign the result to left operand	a%=b is same as a=a%b

---

Conditional operator

The conditional operators in C language are known by two more names

1. **Ternary Operator**
2. **? : Operator**

It is actually the `if` condition that we use in C language decision making, but using conditional operator, we turn the `if` condition statement into a short and simple operator.

The syntax of a conditional operator is :

```
expression 1 ? expression 2: expression 3
```

Explanation:

- The question mark **"?"** in the syntax represents the **if** part.
- The first expression (expression 1) generally returns either true or false, based on which it is decided whether (expression 2) will be executed or (expression 3)
- If (expression 1) returns true then the expression on the left side of **" : "** i.e (expression 2) is executed.
- If (expression 1) returns false then the expression on the right side of **" : "** i.e (expression 3) is executed.

Special operator

Operator	Description	Example
sizeof	Returns the size of an variable	**sizeof(x)** return size of the variable **x**
&	Returns the address of an variable	**&x ;** return address of the variable **x**
*	Pointer to a variable	***x ;** will be pointer to a variable **x**

Selective control structure

- If statements
- Switch statement

An **if** statement consists of a Boolean expression followed by one or more statements.

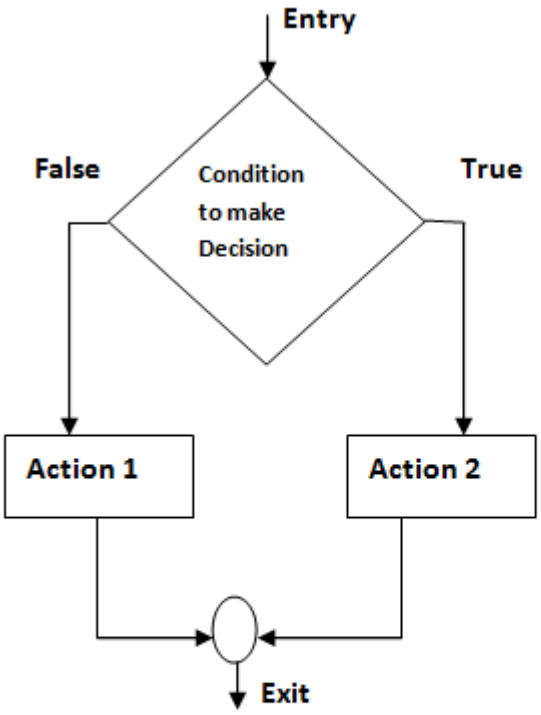
Syntax

The syntax of an 'if' statement in C programming language is −

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

If the Boolean expression evaluates to **true**, then the block of code inside the 'if' statement will be executed. If the Boolean expression evaluates to **false**, then the first set of code after the end of the 'if' statement (after the closing curly brace) will be executed.

C programming language assumes any **non-zero** and **non-null** values as **true** and if it is either **zero** or **null**, then it is assumed as **false** value.



Selection Structure

Implemented using:- If and If..else control statements

switch is used for multi branching

C if statement

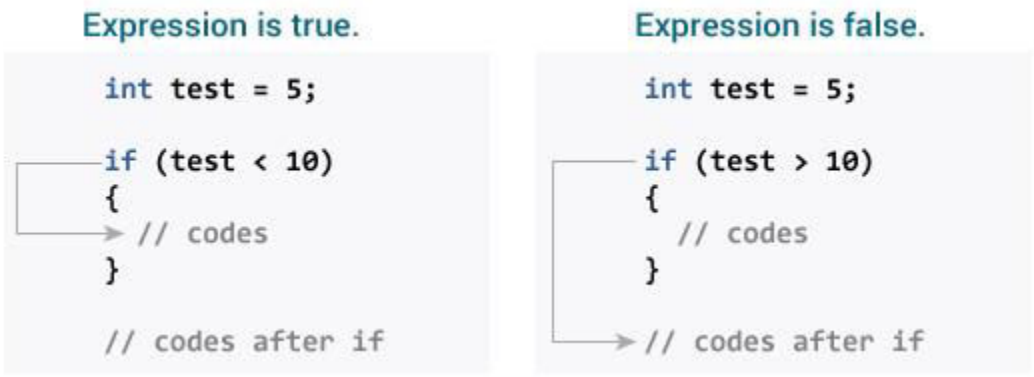
The syntax of *if* statement is:

```
if (testExpression)
{
   // statement(s)
}
```

How if statement works?

The `if` statement evaluates the test expression inside the parenthesis.

- If the test expression is evaluated to true (nonzero), statement(s) inside the body of `if` is executed.
- If the test expression is evaluated to false (0), statement(s) inside the body of `if` is skipped from execution.



Example 1: if statement

```
// Program to display a number if user enters negative number

#include <stdio.h>
int main()
{
    int number;

    printf("Enter an integer: ");
    scanf("%d", &number);

    // Test expression is true if number is less than 0
    if (number < 0)
    {
        printf("You entered %d.\n", number);
    }

    printf("The if statement is easy.");

    return 0;
}
```

Output 1

```
Enter an integer: -2
You entered -2.
The if statement is easy.
```

When user enters -2, the test expression `(number < 0)` is evaluated to true. Hence, `You entered -2` is displayed on the screen.

Output 2

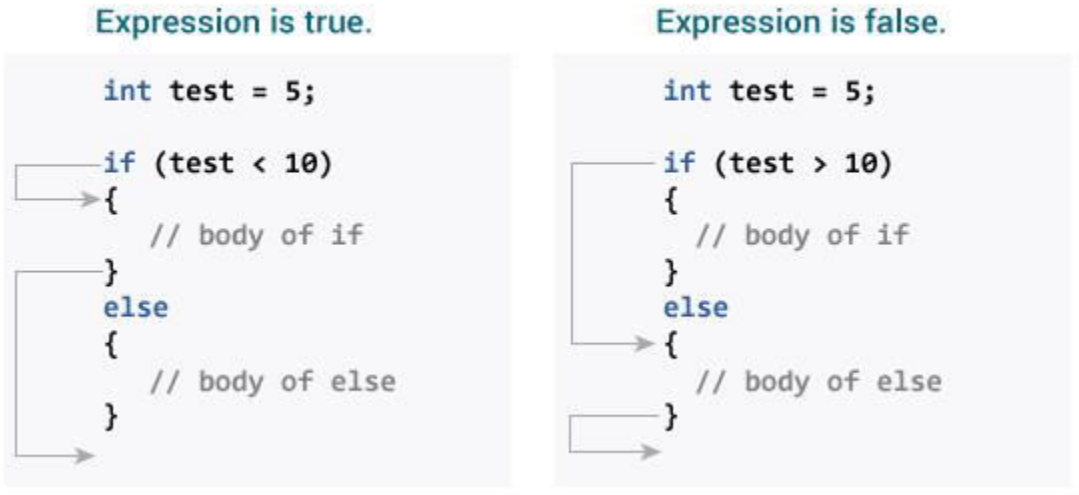
```
Enter an integer: 5
The if statement is easy.
```

When user enters 5, the test expression `(number < 0)` is evaluated to false and the statement inside the body of `if` is skipped.

C if...else statement

The `if` statement may have an optional `else` block. The syntax of `if..else` statement is:

```
if (testExpression) {
    // statement(s) inside the body of if
}
else {
    // statement(s) inside the body of else
}
```



Example 2: if...else statement

```
// Program to check whether an integer entered by the user is odd or even

#include <stdio.h>
int main()
{
    int number;
    printf("Enter an integer: ");
    scanf("%d",&number);

    // True if remainder is 0
    if( number%2 == 0 )
        printf("%d is an even integer.",number);
    else
        printf("%d is an odd integer.",number);
    return 0;
}
```

Output

```
Enter an integer: 7
7 is an odd integer.
```

if...else Ladder (if...else if....else Statement)

The `if...else` statement executes two different codes depending upon whether the test expression is true or false. Sometimes, a choice has to be made from more than 2 possibilities.

The if...else ladder allows you to check for multiple test expressions and execute different statement(s).

Syntax of nested if...else statement.

```
if (testExpression1)
{
   // statement(s)
}
else if(testExpression2)
{
   // statement(s)
}
else if (testExpression 3)
{
   // statement(s)
}
.
.
else
{
   // statement(s)
}
```

Example 3: C if...else Ladder

```
// Program to relate two integers using =, > or <

#include <stdio.h>
int main()
{
    int number1, number2;
    printf("Enter two integers: ");
    scanf("%d %d", &number1, &number2);

    //checks if two integers are equal.
    if(number1 == number2)
    {
        printf("Result: %d = %d",number1,number2);
    }

    //checks if number1 is greater than number2.
    else if (number1 > number2)
    {
```

```
        printf("Result: %d > %d", number1, number2);
    }

    // if both test expression is false
    else
    {
        printf("Result: %d < %d",number1, number2);
    }

    return 0;
}
```

Output

```
Enter two integers: 12
23
Result: 12 < 23
```

Nested if...else

It is possible to include `if...else` statement(s) inside the body of another `if...else` statement.

This program below relates two integers using either <, > and = similar like in `if...else` ladder example. However, we will use nested `if...else` statement to solve this problem.

Example 4: Nested if...else

```
#include <stdio.h>
int main()
{
    int number1, number2;
    printf("Enter two integers: ");
    scanf("%d %d", &number1, &number2);

    if (number1 >= number2)
    {
      if (number1 == number2)
      {
        printf("Result: %d = %d",number1,number2);
      }
      else
      {
        printf("Result: %d > %d", number1, number2);
      }
    }
    else
    {
        printf("Result: %d < %d",number1, number2);
    }
```

```
    return 0;
}
```

If the body of `if...else` statement has only one statement, you do not need to use parenthesis `{ }`.

This code

```
if (a > b) {
    print("Hello");
}
print("Hi");
```

is equivalent to

```
if (a > b)
    print("Hello");
print("Hi");
```

C switch...case Statement

The `if..else..if` ladder allows you to execute a block code among many alternatives. If you are checking on the value of a single variable in `if...else...if`, it is better to use `switch` statement.

The switch statement is often faster than nested `if...else` (not always). Also, the syntax of switch statement is cleaner and easy to understand.

Syntax of switch...case

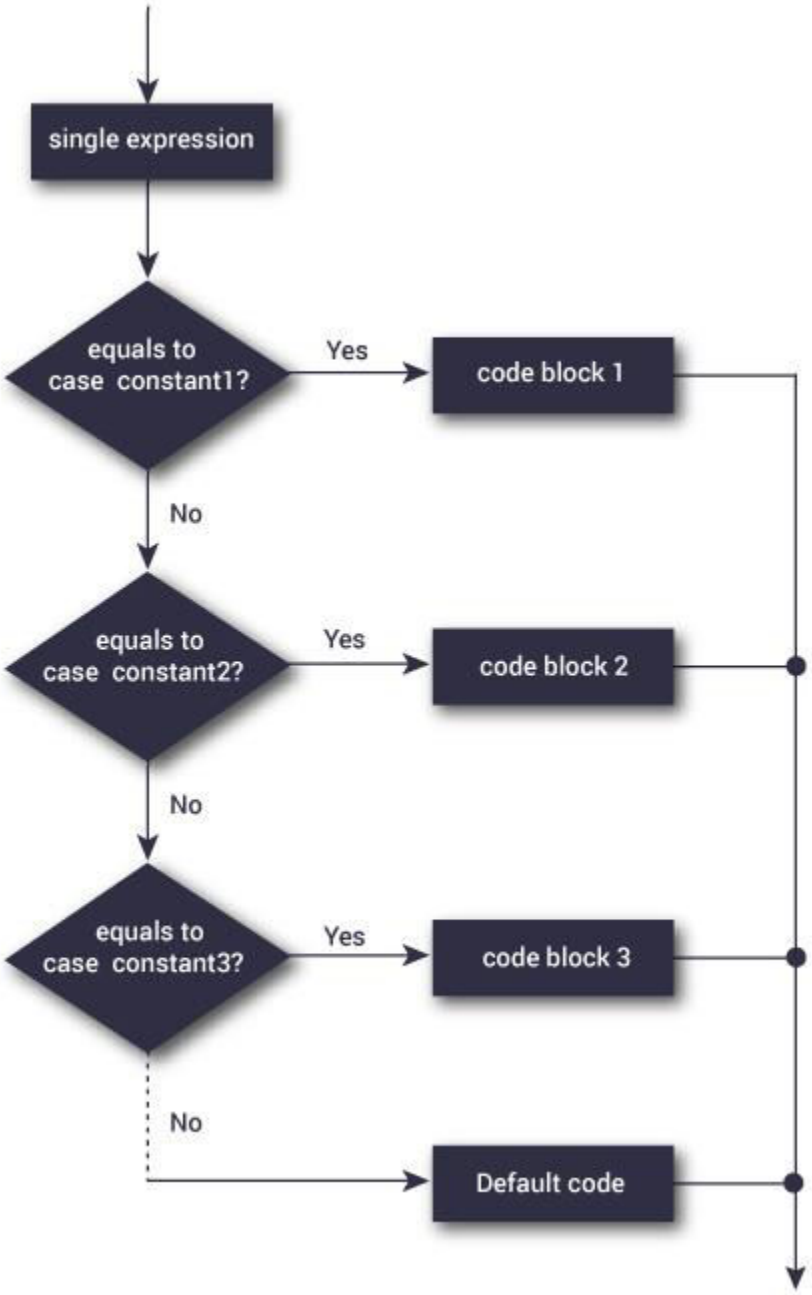
```
switch (n)
{
    case constant1:
        // code to be executed if n is equal to constant1;
        break;

    case constant2:
        // code to be executed if n is equal to constant2;
        break;
        .
        .
        .
    default:
        // code to be executed if n doesn't match any constant
}
```

When a case constant is found that matches the switch expression, control of the program passes to the block of code associated with that case.

Suppose, the value of *n* is equal to *constant2*. The compiler executes the statements after `case constant2:` until break is encountered. When break statement is encountered, switch statement terminates.

switch Statement Flowchart



Example: switch Statement

```
// Following is a simple program to demonstrate
// syntax of switch.
```

```
#include <stdio.h>
int main()
{
   int x = 2;
   switch (x)
   {
        case 1: printf("Choice is 1");
                break;
        case 2: printf("Choice is 2");
                 break;
        case 3: printf("Choice is 3");
                break;
        default: printf("Choice other than 1, 2 and 3");
                 break;
   }
   return 0;
}
```

Output:

```
Choice is 2
```

C programming language provides the following types of decision making statements.

Sr.No.	Statement & Description
1	if statement An **if statement** consists of a boolean expression followed by one or more statements.
2	if...else statement An **if statement** can be followed by an optional **else statement**, which executes when the Boolean expression is false.
3	nested if statements You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).
4	switch statement A **switch** statement allows a variable to be tested for equality against a list of values.
5	nested switch statements You can use one **switch** statement inside another **switch** statement(s).

The ? : Operator

We have covered **conditional operator ? :** in the previous chapter which can be used to replace **if...else** statements. It has the following general form −

```
Exp1 ? Exp2 : Exp3;
```

Where Exp1, Exp2, and Exp3 are expressions. Notice the use and placement of the colon.

The value of a ? expression is determined like this −

- Exp1 is evaluated. If it is true, then Exp2 is evaluated and becomes the value of the entire ? expression.
- If Exp1 is false, then Exp3 is evaluated and its value becomes the value of the expression.

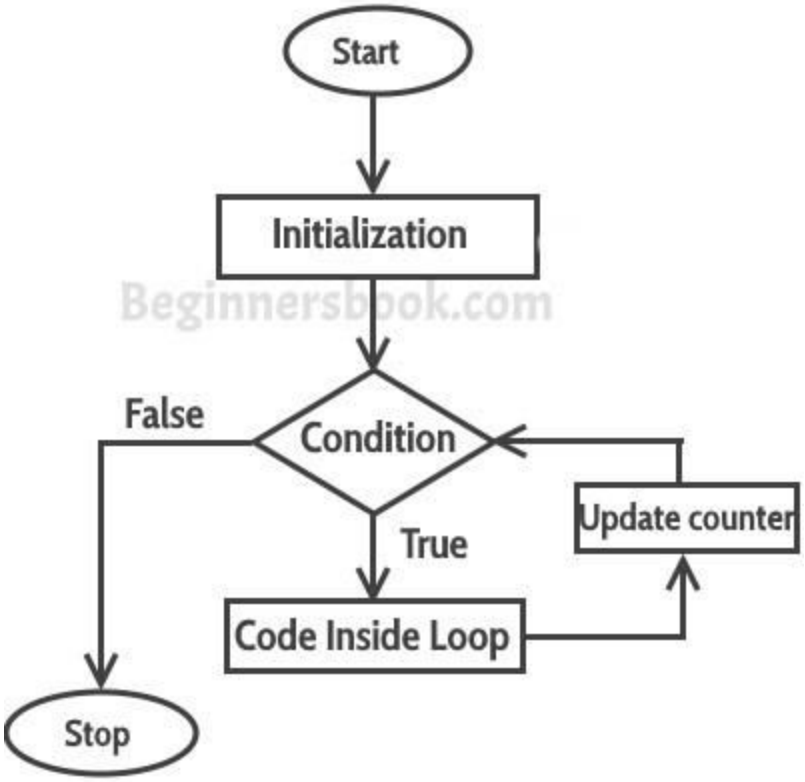
Iterative (looping) control statements

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
}
```



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.

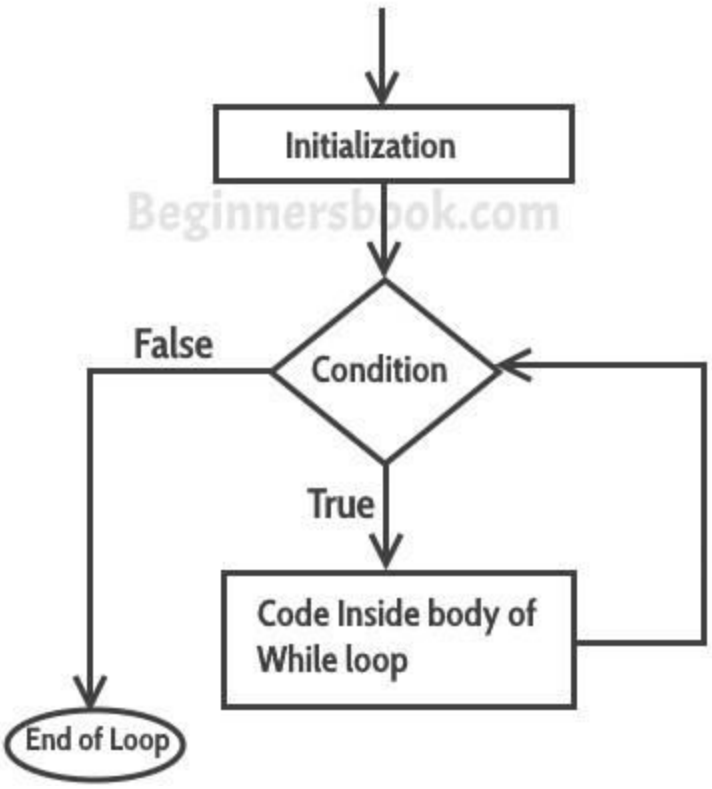
C – while loop in C programming with example

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
}
```



Example of while loop

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

Output:

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.

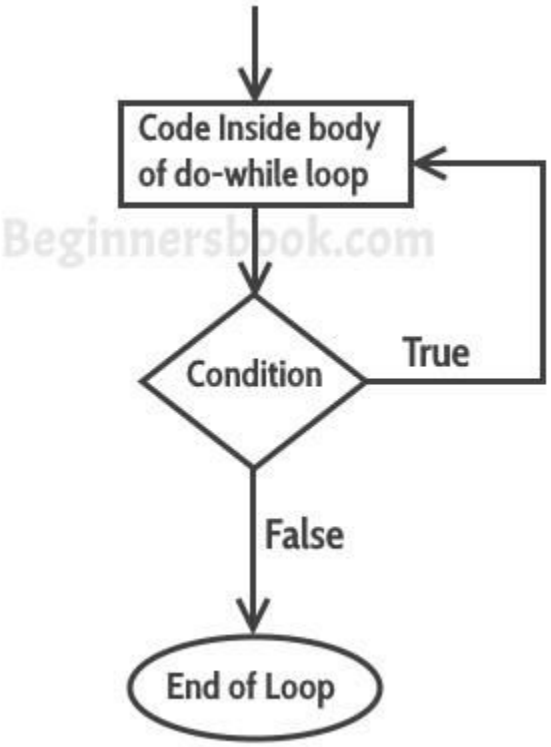
C – do..while loop

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.

Jump Statements in C

Jump statements are used to interrupt the normal flow of program.

Types of Jump Statements

- Break
- Continue
- GoTo

Break Statement

The break statement is used inside loop or switch statement. When compiler finds the break statement inside a loop, compiler will abort the loop and continue to execute statements followed by loop.

Example of break statement

```
#include<stdio.h>

void main()
{
        int a=1;

        while(a<=10)
         {
             if(a==5)
               break;

              printf("\nStatement %d.",a);
             a++;
          }
              printf("\nEnd of Program.");
}
```

```
Output :

        Statement 1.
        Statement 2.
        Statement 3.
        Statement 4.
        End of Program.
```

Continue Statement

The continue statement is also used inside loop. When compiler finds the break statement inside a loop, compiler will skip all the followling statements in the loop and resume the loop.

Example of continue statement

```
#include<stdio.h>

void main()
{
        int a=0;

        while(a<10)
         {

             a++;
```

```
                    if(a==5)
                      continue;

                     printf("\nStatement %d.",a);

                  }
                     printf("\nEnd of Program.");
      }

   Output :

              Statement 1.
              Statement 2.
              Statemnet 3.
              Statement 4.
              Statement 6.
              Statement 7.
              Statement 8.
              Statement 9.
              Statement 10.
              End of Program.
```

Goto Statement

The goto statement is a jump statement which jumps from one point to another point within a function.

Syntax of goto statement

```
              goto label;
                    - - - - - - - - - -
                    - - - - - - - - - -
              label:
                    - - - - - - - - - -
                    - - - - - - - - - -
```

In the above 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 after it.

Example of goto statement

```
       #include<stdio.h>

      void main()
      {
                    printf("\nStatement 1.");
                    printf("\nStatement 2.");
                    printf("\nStatement 3.");

                    goto last;
```

```
                    printf("\nStatement 4.");
                    printf("\nStatement 5.");

                    last:

                    printf("\nEnd of Program.");
    }
```

Output :

```
            Statement 1.
            Statement 2.
            Statement 3.
            End of Program.
```

Operator In c

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators −

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

We will, in this chapter, look into the way each operator works.

Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language. Assume variable **A** holds 10 and variable **B** holds 20 then −

Operator	Description	Example
+	Adds two operands.	A + B = 30
−	Subtracts second operand from the first.	A − B = -10
*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- = 9

Relational Operators

The following table shows all the relational operators supported by C. Assume variable **A** holds 10 and variable **B** holds 20 then −

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.

!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

Logical Operators

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then −

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows −

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0

1 0 0 1 1

Assume A = 60 and B = 13 in binary format, they will be as follows −

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

The following table lists the bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then −

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(60), i.e,. -0111101
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

Assignment Operators

The following table lists the assignment operators supported by the C language −

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Misc Operators ↦ sizeof & ternary

Besides the operators discussed above, there are a few other important operators including **sizeof** and **? :** supported by the C Language.

Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y

Operators Precedence in C

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right