Operators in C

An operator is a special symbol that tells the compiler to perform specific mathematical or logical functions. They are symbols that represent some kind of operation, such as mathematical, relational, bitwise, conditional, or logical computations, which are to be performed on values or variables. The values and variables used with operators are called **operands**.

Types of Operators

Operator can be classified according to :

1. **According to No of operand used**

    - **Unary Operators:** Operators that work on single operand.
      Example: Increment(++) , Decrement(--)
    - **Binary Operators:** Operators that work on two operands.
      Example: Addition (+), Subtraction(-) , Multiplication (*)
    - **Ternary Operators:** Operators that work on three operands.
      Example**:** Conditional Operator(? :)

2. **According to functionality**

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

1. Arithmetic Operators:

These operators are used to perform arithmetic/mathematical operations on operands. There are **9 arithmetic** operators in C language**:**

Examples: (+, -, *, /, %,++,–).

Symbol	Operator	Description	Syntax
+	**Plus**	Adds two numeric values.	**a + b**
-	**Minus**	Subtracts right operand from left operand.	**a - b**
*	**Multiply**	Multiply two numeric values.	**a * b**
/	**Divide**	Divide two numeric values.	**a / b**
%	**Modulus**	Returns the remainder after diving the left operand with the right operand.	**a % b**
+	**Unary Plus**	Used to specify the positive values.	**+a**

-	**Unary Minus**	Flips the sign of the value.	**-a**
++	**Increment**	Increases the value of the operand by 1.	**a++**
--	**Decrement**	Decreases the value of the operand by 1.	**a--**

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

2. Relational Operators:

The **relational operators** in C are used for the comparison of the two operands. All these operators are binary operators that return true or false values as the result of comparison.

Symbol	Operator	Description	Syntax
<	**Less than**	Returns true if the left operand is less than the right operand. Else false	**a < b**
>	**Greater than**	Returns true if the left operand is greater than the right operand. Else false	**a > b**
<=	**Less than or equal to**	Returns true if the left operand is less than or equal to the right operand. Else false	**a <= b**
>=	**Greater than or equal to**	Returns true if the left operand is greater than or equal to right operand. Else false	**a >= b**
==	**Equal to**	Returns true if both the operands are equal.	**a == b**
!=	**Not equal to**	Returns true if both the operands are NOT equal.	**a != b**

Example of C Relational Operators

```
#include <stdio.h>
#include<conio.h>
int main()
{
   int a = 25, b = 5;
   // using operators and printing results
   printf("a < b  : %d\n", a < b);
   printf("a > b  : %d\n", a > b);
   printf("a <= b: %d\n", a <= b);
   printf("a >= b: %d\n", a >= b);
   printf("a == b: %d\n", a == b);
   printf("a != b : %d\n", a != b);

   getch();
}
```

Output

```
a < b  : 0
a > b  : 1
a <= b: 0
a >= b: 1
a == b: 0
a != b : 1
```

Here, 0 means false and 1 means true.

3. Logical Operators

Logical Operators are used to combine two or more conditions/constraints or to complement the evaluation of the original condition in consideration. The result of the operation of a logical operator is a Boolean value either **true** or **false**.

There are **3** logical operators in C**:**

Symbol	Operator	Description	Syntax
&&	**Logical AND**	Returns true if both the operands are true.	**a && b**
||	**Logical OR**	Returns true if both or any of the operand is true.	**a || b**
!	**Logical NOT**	Returns true if the operand is false.	**!a**

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

Example of Logical Operators in C

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

int main()
{
   int a = 25, b = 5;
```

```
    // using operators and printing results
    printf("a && b : %d\n", a && b);
    printf("a || b : %d\n", a || b);
    printf("!a: %d\n", !a);

    getch();
}
```

Output

```
a && b : 1
a || b : 1
!a: 0
```

4. 4. Bitwise Operators

The Bitwise operators are used to perform bit-level operations on the operands. The operators are first converted to bit-level and then the calculation is performed on the operands.

Note: Mathematical operations such as addition, subtraction, multiplication, etc. can be performed at the bit level for faster processing.

Symbol	Operator	Description	Syntax
&	**Bitwise AND**	Performs bit-by-bit AND operation and returns the result.	**a & b**
|	**Bitwise OR**	Performs bit-by-bit OR operation and returns the result.	**a | b**
^	**Bitwise XOR**	Performs bit-by-bit XOR operation and returns the result.	**a ^ b**
~	**Bitwise First Complement**	Flips all the set and unset bits on the number.	**~a**
<<	**Bitwise Leftshift**	Shifts bits to the left by a given number of positions; multiplies the number by 2 for each shift.	**a << b**
>>	**Bitwise Rightshilft**	Shifts bits to the right by a given number of positions; divides the number by 2 for each shift.	**a >> b**

Example of Bitwise Operators

```
#include <stdio.h>
#include<conio.h>
int main()
{
    int a = 25, b = 5;

    // using operators and printing results
    printf("a & b: %d\n", a & b);
    printf("a | b: %d\n", a | b);
```

```
    printf("a ^ b: %d\n", a ^ b);
    printf("~a: %d\n", ~a);
    printf("a >> b: %d\n", a >> b);
    printf("a << b: %d\n", a << b);

getch();
}
```

Output

```
a & b: 1
a | b: 29
a ^ b: 28
~a: -26
a >> b: 0
a << b: 800
```

Operator	Description
<<	Binary Left Shift Operator
>>	Binary Right Shift Operator
~	Binary Ones Complement Operator
&	Binary AND Operator
^	Binary XOR Operator
|	Binary OR Operator

A	B	A & B	A | B	A^ B
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
&	Result is 1 if both bits are 1; otherwise, it's 0.	(A & B) = 12, i.e., 0000 1100
|	Result is 1 if at least one of the bits is 1; otherwise, it's 0.	(A | B) = 61, i.e., 0011 1101
^	Result is 1 if the bits are different; otherwise, it's 0	(A ^ B) = 49, i.e., 0011 0001
~	Turns 1s into 0s and 0s into 1s.	(~A) = ~(60), i.e,. -0111101
<<	Shifts the bits of the number to the left by a specified number of positions. And Zeros are shifted in from the right, and the leftmost bits are discarded.	A << 2 = 240 i.e., 1111 0000
>>	Shifts the bits of the number to the right by a specified number of positions. For unsigned numbers, zeros are shifted in from the left. For signed numbers, the behavior is implementation-defined (usually, the sign bit is preserved).	A >> 2 = 15 i.e., 0000 1111

5. 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	C %= A is equivalent to C

	modulus using two operands and assigns the result to the left operand.	= 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

6. Misc Operators ↦ sizeof & ternary

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

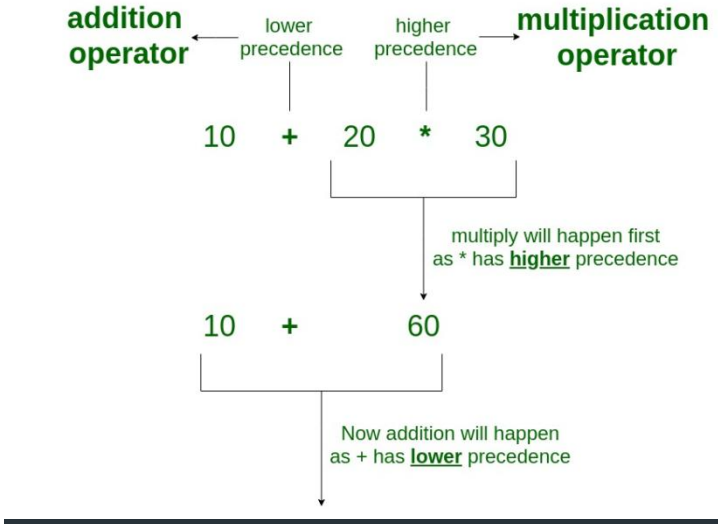
Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise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

Operator Precedence and Associativity in C

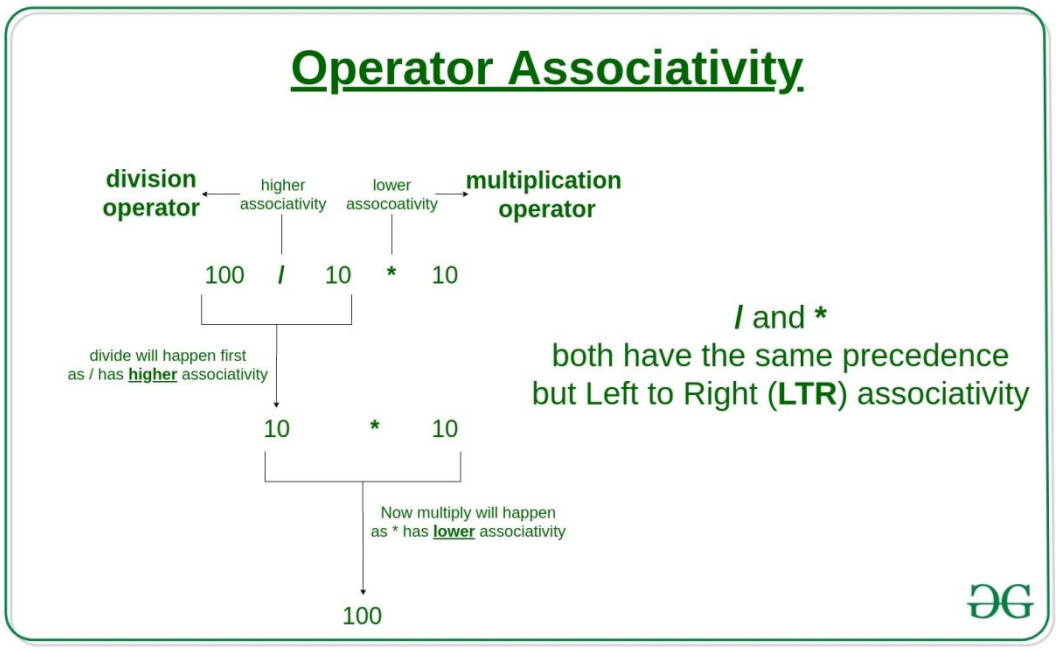
Operators Precedence and Associativity are two characteristics of operators that determine the evaluation order.

Operator precedence determines which operator is performed first in an expression with more than one operator with different precedence.
For example: Solve : 10 + 20 * 30

Operator Precedence



Operators Associativity is used when two operators of same precedence appear in an expression. Associativity can be either **L**eft **t**o **R**ight or **R**ight **t**o **L**eft. **Associativity is only used when there are two or more operators of same precedence. For example:** '*' and '/' have same precedence and their associativity is **L**eft **t**o **R**ight, so the expression "100 / 10 * 10" is treated as "(100 / 10) * 10".



Operator	*Description*	*Associativity*
() **[]** **.** **->** **++ --**	Parentheses or function call Brackets or array subscript Dot or Member selection operator Arrow operator Postfix increment/decrement	left to right
++ -- **+ -** **! ~** **(type)** ***** **&** **sizeof**	Prefix increment/decrement Unary plus and minus not operator and bitwise complement type cast Indirection or dereference operator Address of operator Determine size in bytes	right to left
*** / %**	Multiplication, division and modulus	left to right
+ -	Addition and subtraction	left to right
<< >>	Bitwise left shift and right shift	left to right
< <= **> >=**	relational less than/less than equal to relational greater than/greater than or equal to	left to right
== !=	Relational equal to and not equal to	left to right
&	Bitwise AND	left to right
^	Bitwise exclusive OR	left to right
|	Bitwise inclusive OR	left to right
&&	Logical AND	left to right
||	Logical OR	left to right
? :	Ternary operator	right to left
= **+= -=** ***= /=** **%= &=** **^= |=** **<<= >>=**	Assignment operator Addition/subtraction assignment Multiplication/division assignment Modulus and bitwise assignment Bitwise exclusive/inclusive OR assignment	right to left
,	Comma operator	left to right

Type Casting / Type Conversion

Converting one data type into another is known as type casting or, type-conversion.

Conditional Statements/ Decision Control Instruction/ Statement

Conditional Statements in C programming is used to make decisions based on some given conditions. It is the statement where the compiler executes the particular section of code depending on whether the given condition is true or false.

There are 3 types of decision making control statements in C language.

1. if statements
2. if else statements
3. nested if statements

if Statement

if statement is used to check some given condition and perform some operations when the given condition is true. The statements inside **if** body executes only when the condition defined by **if statement** is true.
If the condition is false then compiler skips the statement enclosed in if's body. C uses the keyword **if** to implement the decision control instruction.

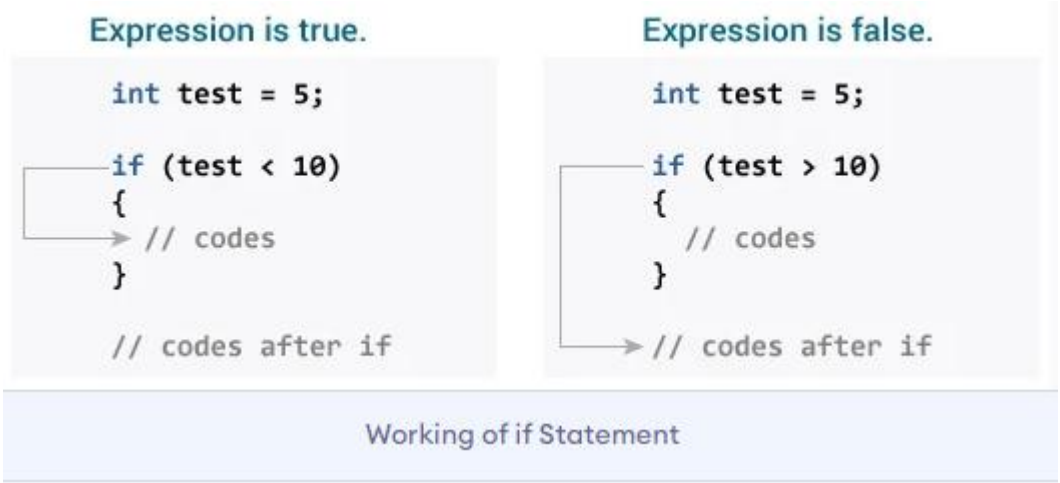
Syntax of if statement:

```
if ( Condition)
do this;
```

- The condition following the keyword if is always enclosed within a pair of parentheses.
- If the condition is true, then the statement is executed. If the condition is not true, then the statement is not executed.

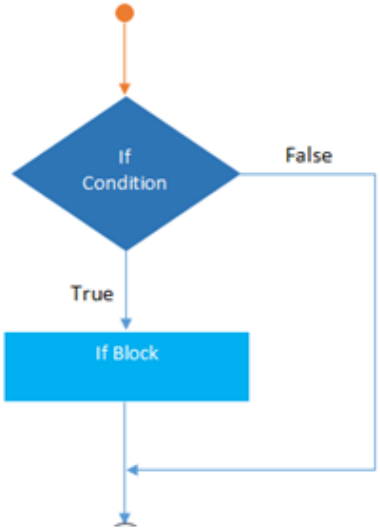
How if statement works?

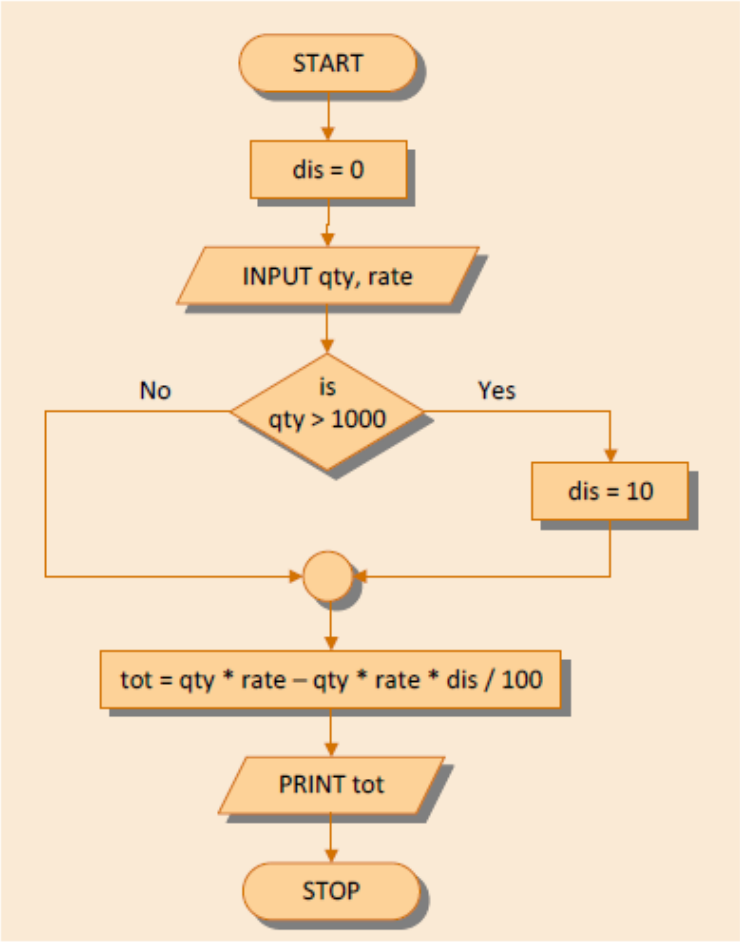
- if statement evaluates the test expression / condition inside the parenthesis ( ) .
- If the test expression / condition becometrue, statements inside the body of if are executed.
- If the test expression / condition become false, statements inside the body of if are not executed.



Working of if Statement

Example of if Statement





EXAMPLE:

```
#include<stdio.h>
#include<conio.h>
main()
{
        int num1, num2;

        printf(“enter two numbers”);
        scanf(“%d%d”, &num1, &num2)

        if(num1<num2)        //test-condition

        printf("Smaller num is = %d”, num1);

        getch();
}
```

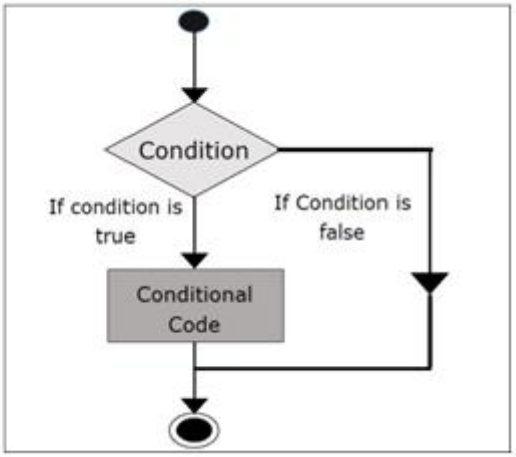
if else Statement

In **if else Statement,** the compiler executes a portion of code when condition is true and executes some other portion of code when condition is false.

- if part executes when condition becomes true.
- else part executes when condition becomes false.

Syntax of if statement:

```
if ( Condition)
    do this;
else
    do this;
```



How if...else statement works?

- **If the test expression is evaluated to true:**
  - statements inside the body of if are executed.
  - statements inside the body of else are skipped from execution.
- **If the test expression is evaluated to false:**
  - statements inside the body of else are executed
  - statements inside the body ofifare skipped from execution.

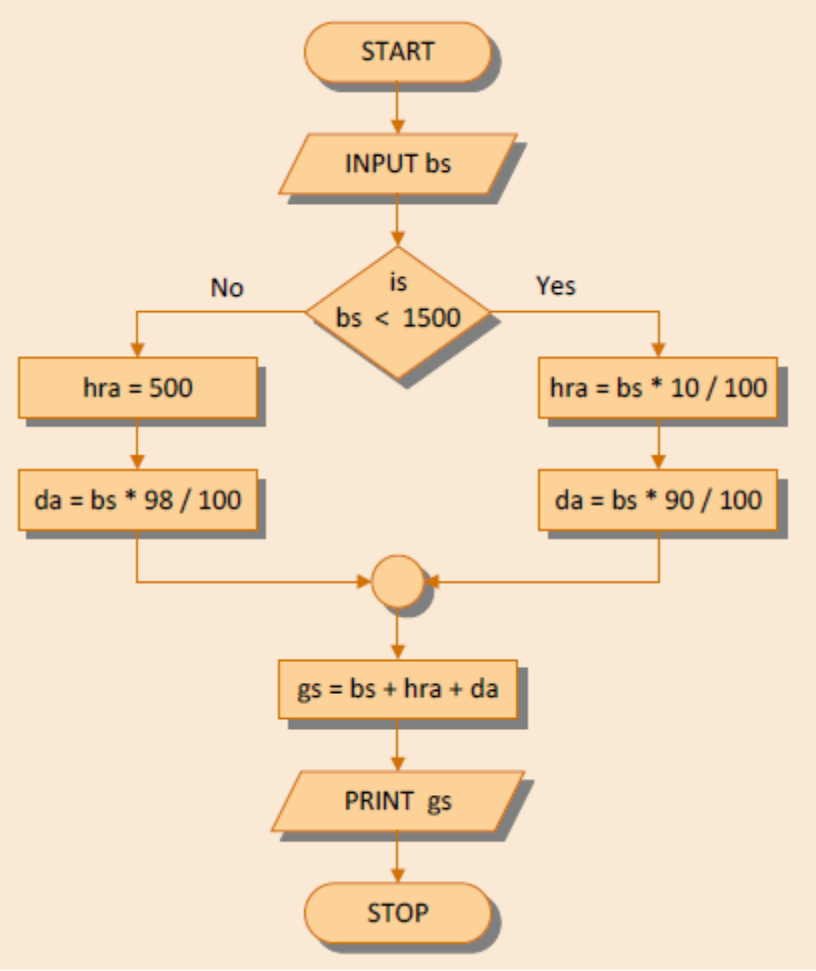
Expression is True

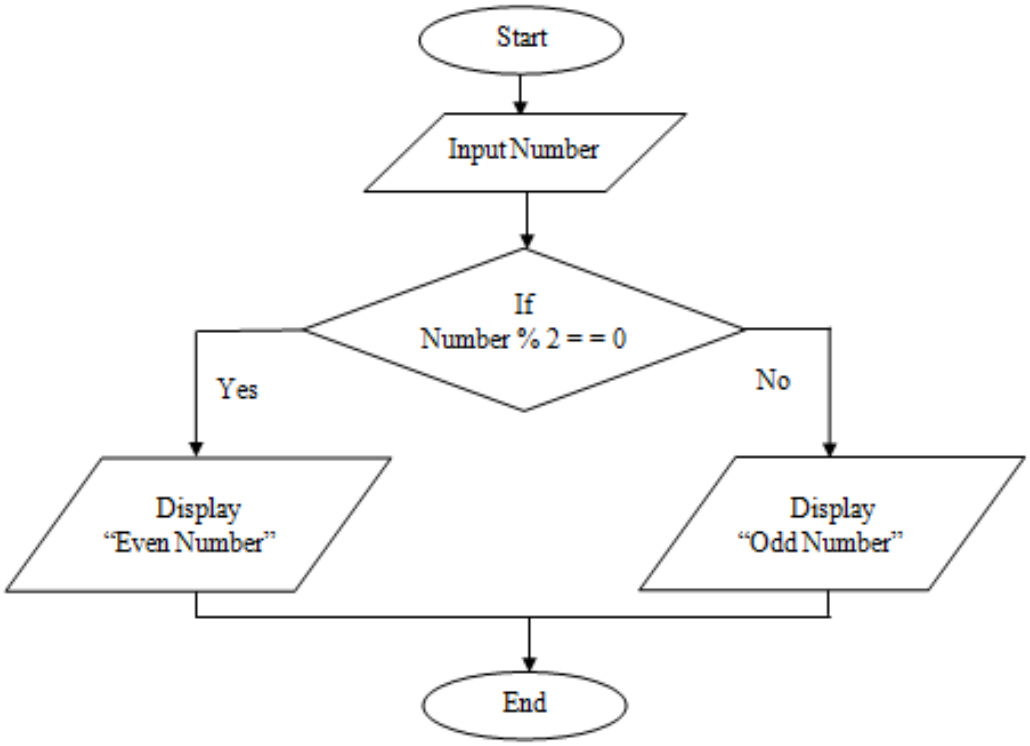
```
int test = 5;
if (test < 10 )
{
  // body of if
}
else
{
  // body of else
}
```

Expression is False

```
int test = 5;
if (test > 10 )
{
  // body of if
}
else
{
  // body of else
}
```

EXAMPLE of if else Statement

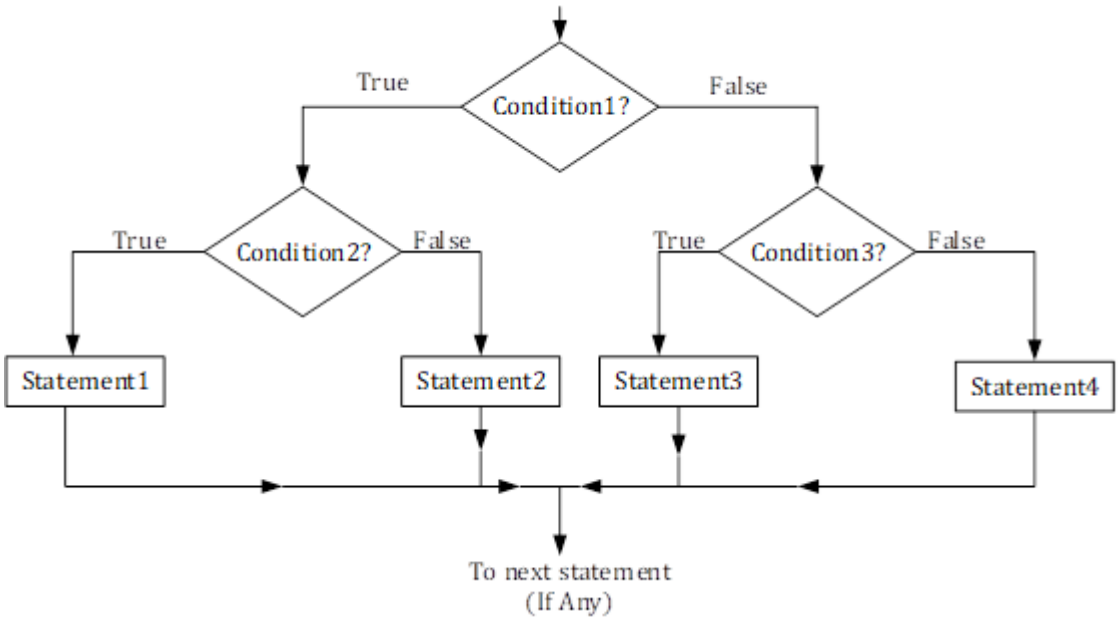




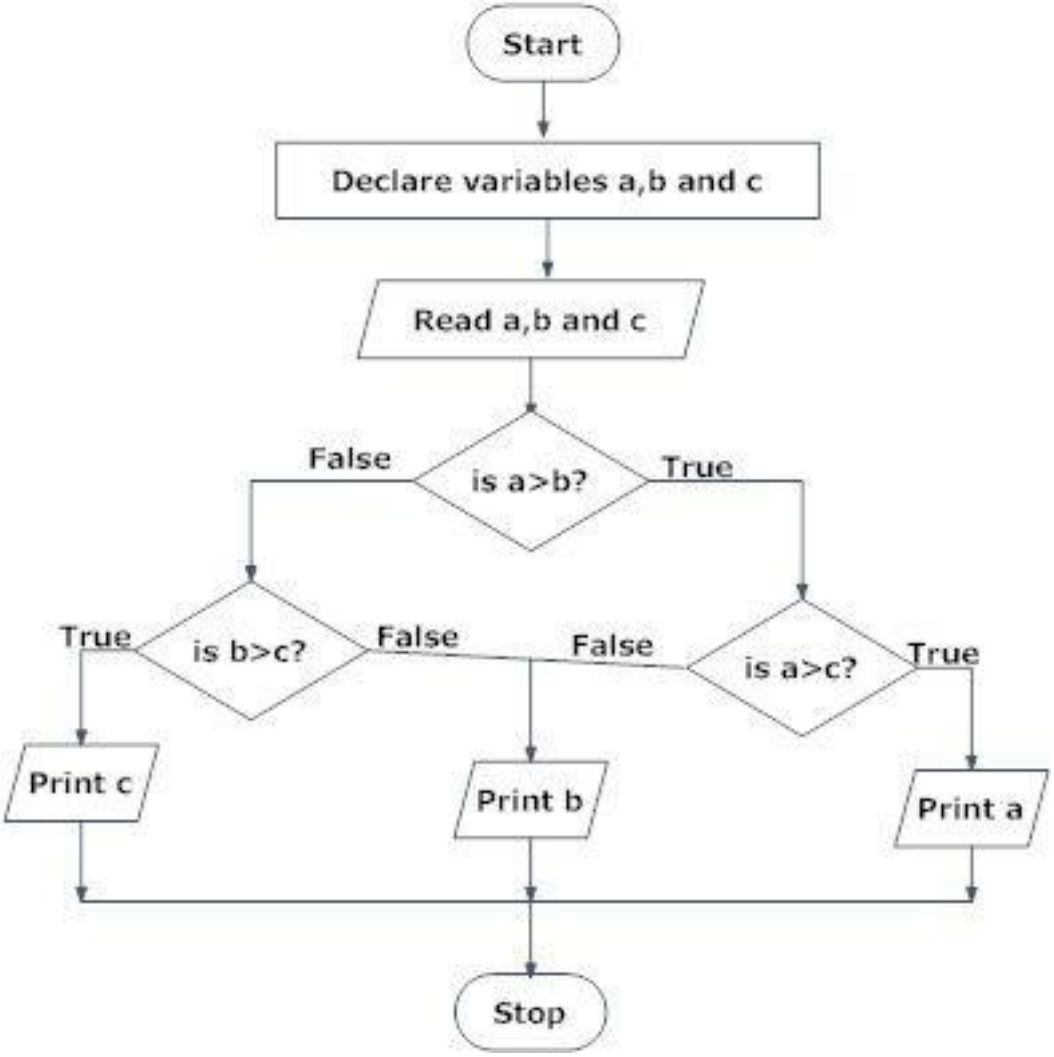

```
#include<stdio.h>
#include<conio.h>
```

```
main( )
{
int no;
printf("Enter any number: ");
scanf("%d",&no);

if(no%2==0)
printf("Even num");
else
printf("Odd num");

getch( );
}
```

Nested If else



Example of Nested if else




```
#include <stdio.h>
#include<conio.h>
main()
{
int num1, num2, num3;
printf(" Enter the number1 = ");
scanf("%d", &num1);
printf("\n Enter the number2 = ");
scanf("%d", &num2);
printf("\n Enter the number3 = ");
scanf("%d", &num3);
if (num1 > num2)
{
if (num1 > num3)
{
printf("\n Largest number = %d \n",num1);
}
else
{
printf("\n Largest number = %d \n",num3);
}
}
```

```
elseif (num2 > num3)
{
printf("\n Largest number = %d \n",num2);
}
else
{
printf("\n Largest number = %d \n",num3);
}
getch ( );
}
```

Switch Statement in C

The control statement that allows to make a decision from the number of choices is called a switch statement or switch-case default statement.

In C programming, the switch statement is a control flow statement that allows you to execute one code block among many alternatives based on the value of an expression. It is often used as an alternative to a series of if-else statements when you have multiple conditions to check against a single variable or expression.

```
switch (expression)
{
   case constant1:
      // Code to execute if expression = = constant1
      break;
   case constant 2:
      // Code to execute if expression = = constant2
      break;
      ……….……
  case constant N:
  braek;

   default:
      // Code to execute if expression doesn't match any case
}
```

The switch statement evaluates the expression once and compares its value against the case constants. Each case represents a possible value of the expression. If the expression matches a case constant, the corresponding block of code is executed.

1. **Break**: The break statement is used to exit the switch block once a matching case is found and executed. Without break, the program will continue executing the code for the following cases (this is called "fall-through").
2. **Default**: The default case is optional and is executed if none of the case constants match the expression.

Example: