



## 2.1 Operators & Expression

Operators are the symbols that instruct to perform some mathematical or logical operations. Operators operate on certain data types called operands, and they form a part of the mathematical or logical expressions. More complex expression use operators.

Example# + - X / &&

There are three categories of operators.

	Operator	Type
unary operator →	+, -, ++, --	Unary operator
Binary operator {	+, -, *, /, %	Arithmetic operator
	<, <=, >, >=, ==, !=	Relational operator
	&&,   , !	Logical operator
	&,  , <<, >>, ~, ^	Bitwise operator
	=, +=, -=, *=, /=, %=	Assignment operator
Ternary operator →	?:	Ternary or conditional operator

Expression means whose evaluation yields numeric value. Expression contains constant, variable and operators. In C, every expression evaluates to a value. C language supports rich set of Operators. Like arithmetic, logical, relational etc.

Example# a+b-c





## 2.1.1 Operators



### 2.1.1.1 Arithmetic operators

These operators are used to perform some basic operations like addition, subtraction, division, multiplication and modulo. These require two operands to perform its operation hence they are also called binary operators. They are also known as mathematical operators.

Operator	Meaning	Example
+	To perform addition operation.	a+b
-	To perform subtraction operation.	a-b
*	To perform multiplication operation.	a*b
/	To perform division operation.	a/b
%	Modulus: To give remainder of the division operation.	a% b

### Example#

```
#include<stdio.h>
#include<conio.h>
main()
{
    int a,b;
    clrscr();
    a=22;
```





```

    b=2;

    printf("\nAdd = %d",a+b);
    printf("\nSub = %d",a-b);
    printf("\nMul = %d",a*b);
    printf("\nDiv = %d",a/b);
    printf("\nModuls = %d",a%b);

    getch();
}

```

#### Output:

```

Add = 24
Sub = 20
Mul = 44
Div = 11
Moduls = 0

```

### 2.1.1.2 Increment & Decrement

These operators operate on one operand only. They are also known as unary operators. Examples are increment (++), Decrement (--).

These operators are used with variable only.

Symbol	Meaning	Example
++	To increment value by one in the operand.	a++, ++a
--	To decrement value by one in the operand.	a--, --a

Two type of notation are used with these operators: postfix and prefix

#### Postfix notation

Syntax# operand symbol

Here symbol is used after the operand means operator modify operand after it is used.

Example#

```

a=5;
b=a++;

```

After these two operations are executed, a variable has a 6 and b has 5.

```

a=5;
b=a--;

```

After these two operations are executed, a variable has a 4 and b has 5.

#### Prefix notation





Syntax# symbol operand

Here symbol is used before the operand means operators modify operand first then after it is used.

Example#

a=5;

b=++a;

After these two operations are executed, a variable has a 6 and b has 6.

a=5;

b=--a;

After these two operations are executed, a variable has a 4 and b has 4.

### 2.1.1.3 Assignment operators

Assignment operator are used to assign value to any variable. It is denoted by '=' sign. Assignment operator is a binary operator i.e. it operates on two values.

It has the lowest precedence

Operator	Meaning	Example	Equivalent to
=	Assign value to operand	A=5;	
+=	Add given value to operand value	a+=5;	a=a+5;
-=	Subtract the given value from operand value	a-=5;	a=a-5;
*=	Multiply operands value from the given value	a*=5;	a=a*5;
/=	Divides the operands value by the given value	a/=5;	a=a/5;
%=	Assign remainder of operators value by given value	a%=5;	a=a%5;

Example#

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

void main()
{
    clrscr();
    int a;

    a=50;
    printf("\na=%d",a);

    a+=5;
    printf("\na=%d",a);

    a-=10;
```





```
printf("\na=%d",a);

a*=2;
printf("\na=%d",a);

a/=2;
printf("\na=%d",a);

a%=5;
printf("\na=%d",a);

getch();
}
```

**Output:**

```
a =50
a =55
a =45
a =90
a =45
a =0
```

#### 2.1.1.4 Relational operators

It returns Boolean value only.

Relational operators are used to compare between two operands and evaluate as either true or false. For example, # 1 or 0. If the condition is true then return 1 otherwise return 0. They are used in conjunction with logical operators and conditional & looping statements.

Operator	Meaning	Example
<	Less than	a<b
>	Greater than	a>b
>=	Greater than or equal to.	a>=b
<=	Less than or equal to.	a<=b
!=	Not equal to	a!=b
==	Equal to.	a==b

Example#

```
#include<stdio.h>
#include<conio.h>
main()
{
int a,b,c;
clrscr();
```





```
a=22;
b=22;
c=5;

printf("\na = %d b = %d c = %d",a,b,c);

printf("\na>c = %d",a>c);
printf("\na<c = %d",a<c);
printf("\na<=b = %d",a<=b);
printf("\na>=b = %d",a>=b);
printf("\na!=b = %d",a!=b);
printf("\na==b = %d",a==b);

getch();
}
```

#### Output:

```
a = 22 b = 22 c = 5
a>c = 1
a<c = 0
a<=b = 1
a>=b = 1
a!=b = 0
a==b = 1
```

### 2.1.1.5 Logical operators

They are also known as binary operators. Logical operators return Boolean value.

Operator	Meaning	Example
	To perform logical OR operation.	a==5    a==7
&&	To perform logical AND operation	a>20 && a<40
!	To perform logical NOT operation	a!=5

Example#

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

void main()
{
    int a,b;
    clrscr();

    a=10,b=20;

    printf("a>=1 && a<=20=%d",a>=1 && a<=20);
```





```
printf("\na==1 || b==20=%d",a==1 || b==20);

printf("\na!=5=%d",a!=5);

getch();
}
```

**Output:**

```
a>=1 && a<=20= 1
a==1 || b==20 = 1
a!=5 = 1
```

### 2.1.1.6 Bit-wise special operators

We have performed various operations using various operators on entire numbers so far. But suppose we want to perform operation on bits rather than entire byte ,C supports bitwise operator to perform operation on bits of integral operands.

Bitwise operators are works on binary levels.

Syntax#

Operand1 symbol(bitwise) operand2

Here operand1, operand2 are the variable or constant.

Operator	Meaning	Example
&	Bitwise AND. If both the bits of the both operand are 1 then resulted bit is 1. If any one bit is 0 then resulted bit is 0. (referred AND true table)	a & b
	Bitwise OR. If any one bit of the both operand are 1 then resulted bit is 1. If both bit are 0 then resulted bit is 0. (referred OR true table)	a   b
~	Bitwise Complement. Set 1 to 0 and 0 to 1 bit.	~a
^	Bitwise Exclusive OR. If both bits of the both operand are 1 then resulted bit is 0 otherwise resulted bit is 1.	a ^ b
>>	Bitwise shift right. It moves the bit of the first operand to the right by the number of bits specified by the second operand . It discards the far right bit and fill from the right with 0 bits.	a >> 2
<<	Bitwise shift left. It moves the bit of the first operand to the left by the number of bits specified by the second operand. It discards the far left bit and fills from the right with 0 bits.	a << 2

Example# Program to demonstrate use of bitwise AND operator

```
#include<stdio.h>
#include<conio.h>
void main ()
{
    int a, b, c;
    clrscr();
    a=5, b=6 ;
    c=a & b;
```





```
printf ("\n %d ", c);  
getch();  
}
```

**Output:**

4

Example# **Program to demonstrate use of bitwise OR and shift operators**

```
#include<stdio.h>  
#include<conio.h>  
void main ()  
{  
    int a, b, c;  
    clrscr();  
    a=5, b=6 ;  
    c=a | b;  
    printf ("\n %d ", c);  
    c=a>>2;  
    printf("\n %d",c);  
    a=5;  
    c=a<<1;  
    printf("\n %d",c);  
    getch();  
}
```

**Output:**

7

1

10

### 2.1.1.7 Conditional operator

Conditional operator is also known as ternary operator. It checks the expression and returns either true (1) or false (0).

It is known as ternary operator because it has three operands.

Syntax#

condition **?** expression 1 **:** expression 2

If expression 1 is true then expression 2 is executed else expression 3 is executed.

Example# Maximum between two with conditional operator

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







```
void main()
{
    int a,b,max;
    clrscr();
    a=5,b=6;
    max=a>b?a:b;
    printf("Maximum=%d",max);
    getch();
}
```

**Output:**

Maximum=6

Example# Program to check whether a number is odd or even using conditional operator

```
#include<stdio.h>
#include<conio.h>
main()
{
    int a;
    clrscr();
    printf("\nEnter value of a=>");
    scanf("%d",&a);

    a%2==0?printf("\n%d is even",a):printf("\n%d is odd",a);

    getch();
}
```

**Output**

Enter value of a=>3  
3 is odd

### 2.1.1.8 Comma operator

The comma operator can be used to link the related expressions together. It allows evaluating two or more distinct expressions together.

A comma-linked list of expressions is evaluated left to right and the value of the right-most expression is the value of the combined expression. For example, the statement

Value = (x = 2, y = 3, x +y);

First 2 is assigned to x, then 3 is assigned to y and then finally x+y(2+3) is executed and assigned to Value.





Since comma operator has the lowest precedence of all operators, the parentheses are necessary.

#### **2.1.1.9 sizeof() operator**

sizeof() is a unary operator which returns the size of the variable passed as an argument.

Example#

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

void main()
{
    clrscr();
    printf("\nint=%d",sizeof(int));
    printf("\nchar=%d",sizeof(char));
    printf("\nlong=%d",sizeof(long));
    printf("\nfloat=%d",sizeof(float));
    getch();
}
```

**Output:**

```
int=2
char=1
long=4
float=4
```

#### **2.1.2 Expression, Precedence & Associativity**

In programming, an expression is any legal combination of symbols that represents a value.

Example# a+b-c

In c language expression evaluation is mainly depends on priority and associativity.

C Programming provides its own rules of Expression. Every expression contains at least one operand and can contain any number of operators.

Operands refer to the values and operator are symbol that represent particular action.

#### **Precedence**

This represents the evaluation of expression starts from "what" operator.





Operator precedence determines which operator is performed first in an expression with more than one operators with different precedence.

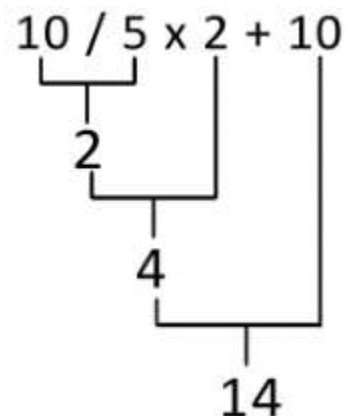
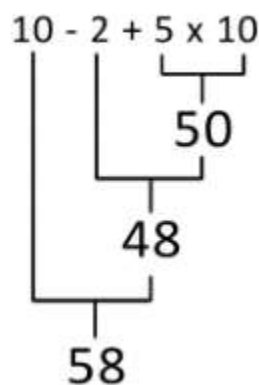
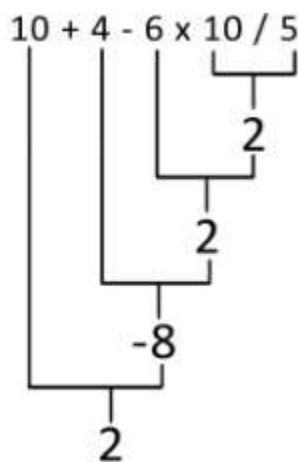
Example#  $5 + 10 * 20$  is calculated as  $5 + (10 * 20)$  and not as  $(5 + 10) * 20$ .

### Associativity

Associativity is only used when there are two or more operators of same precedence.

Operator	Precedence	Associativity
{ }, ( ), [ ]	1	Left to right
++, --, !	2	Right to left
*, /, %	3	Left to right
+, -	4	Left to right
<, <=, >, >=, ==, !=	5	Left to right
&&	6	Left to right
	7	Left to right
?:	8	Right to left
=, +=, -=, *=, /=, %=	9	Right to left

Example#





## 2.2 Console based I/O and related built-in I/O function

### 2.2.1 printf()

printf() function is used to print the "character, string, float, integer, octal and hexadecimal values" onto the output screen.

printf is a predefined function in "stdio.h" header file, by using this function, we can print the data or user defined message on console/monitor.



While working with printf(), it can take any number of arguments but first argument must be written within double cotes (" ") and every argument should be separated by comma ( , ). Within the double cotes, whatever we pass, compiler prints it as it is on the output screen.

Syntax#

```
printf("format specifiers",value1,value2,..);
```

Example#

```
#include<stdio.h>
#include<conio.h>
main()
{
    char ch = 'A';
    char siteName[40] = "www.shyamsir.com";
    float rsFloat = 22.778;
    int rollno = 21;
    double meterDbl = 33.7777889;

    clrscr();

    printf("Character is %c \n", ch);
    printf("String is %s \n", siteName);
    printf("Float value is %f \n", rsFloat);
    printf("Integer value is %d \n", rollno);
    printf("Double value is %lf \n", meterDbl);
    printf("Octal value is %o \n", rollno);
    printf("Hexadecimal value is %x \n", rollno);

    getch();
}
```

**Output**

Character is A





String is www.shyamsir.com  
 Float value is 22.778000  
 Integer value is 21  
 Double value is 33.777789  
 Octal value is 25  
 Hexadecimal value is 15

### 2.2.2 scanf()

scanf() is a predefined function in "stdio.h" header file. It is used for taking input from the user.

scanf() function is used to read character, string, numeric data from keyboard

Syntax#

scanf("format specifiers",&value1,&value2,...);



Format specifier	Type of value
%d	Integer
%f	Float
%lf	Double
%c	Single character
%s	String
%u	Unsigned int
%ld	Long int
%lf	Long double

Example#

```
#include<stdio.h>
#include<conio.h>
main()
{
    char ch;
    char siteName[40];
    float rsFloat;
    int rollno;
    double meterDbl;

    clrscr();

    printf("\nEnter character =>");
    scanf("%c",&ch);

    printf("\nEnter site Name =>");
    scanf("%s",&siteName);

    printf("\nEnter float value =>");
    scanf("%f",&rsFloat);
```





```
printf("\nEnter roll no =>");
scanf("%d",&rollno);

printf("\nEnter Double value =>");
scanf("%lf",&meterDbl);

printf("\nCharacter is %c \n", ch);
printf("String is %s \n", siteName);
printf("Float value is %f \n", rsFloat);
printf("Integer value is %d \n", rollno);
printf("Double value is %lf \n", meterDbl);

getch();
}
```

### Output:

```
Enter character =>a
Enter site Name =>www.shyamsir.com
Enter float value =>23.44567
Enter roll no =>21
Enter Double value =>12.888345

Character is a
String is www.shyamsir.com
Float value is 23.445669
Integer value is 21
Double value is 12.888345
```

### 2.2.3 getch()

getch() is a inbuilt function which is stored in "conio.h" header file.

getch() takes a character input but does not display the character which is entered.

Example#

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

void main()
{
    clrscr();
    char a;
```





```
printf("Enter value of a: ");
a=getch();
printf("a=%d",a);
getch();
}
```

### Output

Enter value of a:  
a=c

#### 2.2.4 getchar()

getchar() is a inbuilt function which is stored in "stdio.h" header file and it does the same function as getch() the difference between both of them is that getchar() gives the output of the entered character while getch() doesn't.

Example#

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

void main()
{
    clrscr();
    char a;
    printf("Enter value of a: ");
    a=getchar();
    printf("a=%d",a);
    getch();
}
```

### Output

Enter value of a: c  
a=c

#### 2.2.5 putchar()

putchar() is an in built function which is used to print a character on the console screen.

Example#

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

void main()
{
    clrscr();
```





```
char a;  
printf("Enter value of a: ");  
a=getchar();  
putchar(a);  
getch();  
}
```

### Output

Enter value of a:c  
c

## 2.3 Header File

A header file in C programming language is a file with .h extension which contains a set of common function declarations and macro definitions which can be shared across multiple program files.

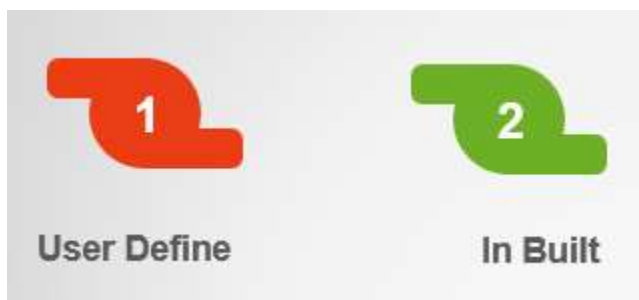
There are 32 numbers of header file.

Each header file contains information (or declarations) for a particular group of functions.

Example#

stdio.h header file contains standard Input and Output functions.  
ctype.h header file contains character handling functions  
string.h header file contains string handling functions.

There are two types of header files



### User Define Header file

The files that the programmer writes and store with .h extension means user define

### Inbuilt Header file

The files that comes with our compiler means in built and extension with .h

### #include







We can include header file in our program by including it with the C preprocessing directive `#include`

`#include` Preprocessor Directives is used to include both system header files and user defined header files in C Program.

Syntax#

```
#include<headfileName.h>
or
#include "headerfileName.h"
```

The `#include` directive works by directing the C preprocessor to scan the specified file as input before continuing with the rest of the current source file.

Including a header file in a c program is equivalent to copying the content of the header file in your program.

Some of the header file of C:

stdio.h	string.h	math.h	stdlib.h
floats.h	conio.h	time.h	limits.h
graphic.h	ctype.h	malloc.h	calloc.h
sound.h			

### 2.3.1 Steps to create our own header file

1. Open a text editor and type a functions definition, like we define a new function in C program.

```
int Add(int a,int b)
{
    return a+b;
}

int Sub(int a,int b)
{
    return a-b;
}
```

2. Save this file with .h extension. Lets assume we saved this file as myCalc.h.
3. Copy myCalc.h header file to the same directory where other inbuilt header files are stored. (Usually in c:\etc\bin)
4. To Include your new header file in a c program used `#include` preprocessor directive. `#include "myCalc.h"`
5. Create new file and add "myCalc.h" header file.

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





```
#include<conio.h>
#include "myCalc.h"

main()
{
int sumans,subans;

clrscr();

sumans=Add(5,2);
subans=Sub(5,2);
printf("\nAdd = %d",sumans);
printf("\nSub = %d",subans);

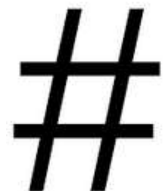
getch();
}
```

### Output

Add = 7  
Sub = 3

## 2.4 PreProcessor

The C preprocessor is a macro preprocessor that transforms your program before it is compiled. Preprocessor is just a text substitution tool and it instructs the compiler to do required pre-processing before the actual compilation.



All preprocessing directives begins with a # symbol. For example,

```
#include<stdio.h>
#define PI 3.14
#ifdef TURBOC
    #define INT_SIZE 16
#else
    #define INT_SIZE 32
#endif
```

Some of the common uses of C preprocessor are:

Processor	Meaning
#define	Substitutes a preprocessor macro.
#include	Inserts a particular header from another file.
#undef	Undefines a preprocessor macro.
#ifdef	Returns true if this macro is defined.
#ifndef	Returns true if this macro is not defined.
#if	Tests if a compile time condition is true.





#else	The alternative for #if.
#elif	#else and #if in one statement.
#endif	Ends preprocessor conditional.
#error	Prints error message on stderr.
#pragma	Issues special commands to the compiler, using a standardized method.

### 2.4.1 #define Preprocessor

The #define directive takes two forms:



### 2.4.2 Defining a Constant using #define

Syntax#

#define token [value]

Example#

```
#define PI 3.14
#define N 200
```

It has no type. It is a simple text substitution. The text 3.14 or 200 will be dropped in place wherever PI and N appears as a token.

### 2.4.3 Creating a macro function using #define

Syntax#

#define token(<arg> [, <arg>s ... ]) statement

Example#

```
#define MAX(a, b) ((a) > (b) ? (a) : (b))
#define SQR(a) (a*a)
```

It has no type. Macro can be type-neutral means it works with any data type. It's inlined directly into the code, so there isn't any function call overhead.

Example# #define preprocessor

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





```
#include<conio.h>

#define PI 3.14
#define Max(a,b)(a>b?a:b)

main()
{
float r=10;
clrscr();

printf("\nArea of Rect = %.2f",PI*r*r);

printf("\nMax = %d",Max(5,2));

getch();
}
```

**Output**

Area of Rect = 314.00

Max = 5





## Decision Making Structure

### 2.5 Control statement

We can control the order of execution of the program with the help of control statement, based on logic and values.

In the sequential flow of control, statements are executed line by line but in the case of conditional program the order of execution is based on data values and conditional logic.

Conditional statements cause variable flow of execution of the same program based on certain condition to be true or false, everytime when a program is executed

The control statements can be classified into three groups: Decision making statements , Looping statements and Jumping statements.



#### 2.5.1 Decision making statement

Decision making statement are also known as Selection statement.

There are two ways to take decision,



##### 2.5.1.1 if

- In some situation, we would like to execute some logic when a condition is TRUE, and some other logic when the condition is FALSE and this can be done by using if statement.
- The condition's result comes from the comparison of two values.
- Condition is an expression that is made up by using relational operators (>, <, >=, <=, ==, etc.) and operand and logical operators (&& , || ,etc) if required, and it returns true or false.

##### 2.5.1.1.1 Type of if statements

- There are 4 ways to write if statements as below given image.





#### 2.5.1.1.1.1 If

- In some situation we would like to execute some logic when a condition is TRUE.
- The condition usually results from comparison of two values.
- If the condition is TRUE then the control between the if brackets.

Syntax#

```
if(condition)
{
    Logic..
}
```



Example# To check whether number is 5

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a;
    clrscr();
    printf("\n Enter value of a=");
    scanf("%d",&a);
    if(a==5)
    {
        printf("\n No. is 5");
    }
    getch();
}
```

**Output:**








Enter value of a=5





No. is 5

### **2.5.1.1.1.2 If...else**

-  In some situation we would like to execute some logic when a condition is TRUE, and some other logic when the condition is FALSE.
-  An if statement consists of a Boolean expression.
-  The condition usually results from a comparison of two values.
-  If the condition is TRUE then the control goes to between if and else block, that is the program will execute the code between if and else statements, else program will execute the code after else.
-  We can write only if block. The else if and else clauses are both optional.
-  We can have any number of elseif statements or none.
-  We can also write nested if-else block.

Syntax#

```
if(condition)
{
    Logic..
}
else
{
    Logic..
}
```

Example# To check whether a number is positive or negative

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a;
    clrscr();
    printf("\n Enter the value of a=");
    scanf("%d",&a);

    if(a>0)
    {
        printf("\n No is pos");
    }
    else
    {
        printf("\n No is neg");
    }
}
```





```
    getch();  
}
```


**Output:**


Enter the value of a=4

No is pos


### 2.5.1.1.1.3 Nested If...else

 If within If is known as a Nested If...else

 In some cases we would like to execute some more condition when a condition is TRUE, and some other condition logic when the condition is FALSE.

 The most general way of doing this is by using the else if variant on then if statement.

 This works by cascading several comparisons.

 As soon as one of these gives a true result, the following statement or block is executed, and no further comparisons are performed.

Syntax#

```
if(condition)  
{  
    if(condition)  
    {  
    }  
}  
else  
{  
    if(condition)  
    {  
    }  
    else  
    {  
    }  
}  
}
```

Example# To find the maximum number out of three numbers

```
#include<stdio.h>  
#include<conio.h>  
  
void main()  
{  
    int a,b,c;  
    clrscr();
```







```
printf("\n Enter value of a=");
scanf("%d",&a);
printf("\n Enter value of b=");
scanf("%d",&b);
printf("\n Enter value of c=");
scanf("%d",&c);

if(a>b)
{
    if(a>c)
    {
        printf("\n%d is max",a);
    }
    else
    {
        printf("\n%d is max",c);
    }
}
else
{
    if(b>c)
    {
        printf("\n%d is max",b);
    }
    else
    {
        printf("\n%d is max",c);
    }
}
getch();
}
```

**Output:**


Enter value of a=4

Enter value of b=3

Enter value of c=6

6 is max

#### 2.5.1.1.1.4 if...else if...else

 There are cases when we would like to execute some more condition when a condition is TRUE, and some other condition logic when the condition is FALSE.

 We can write multiple else if condition with if statement.





Syntax#

```
if(condition)
{

}
else if(condition)
{

}
else
{

}
}
```

Example# To check whether a number is odd even or zero

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a;
    clrscr();
    printf("\n Enter the value of a=");
    scanf("%d",&a);
    if(a%2==0)
    {
        printf("No is even");
    }
    else if(a%2!=0)
    {
        printf("No is odd");
    }
    else
    {
        printf("No is zero");
    }
    getch();
}
```

**Output:**

Enter the value of a=6  
No is even

Example# To find total and grade of a student

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





```
void main()
{
    int a,b,c,t;
    clrscr();
    printf("\nEnter marks in science=>");
    scanf("%d",&a);
    printf("\nEnter marks in maths=>");
    scanf("%d",&b);
    printf("\nEnter marks in computer=>");
    scanf("%d",&c);
    printf("\nTotal=%d",a+b+c);
    t=a+b+c;
    if(t<=100)
    {
        printf("\nC grade");
    }
    else if(t<=200)
    {
        printf("\nB grade");
    }
    else
    {
        printf("\nA grade");
    }
    getch();
}
```

**Output:**

Enter marks in science=>60

Enter marks in maths=>66

Enter marks in computer=>75

Total=201

A grade

### Properties of If..else Statement

- It's not compulsory to write else block.
- We cannot write multiple if or else. Usually we have to write { } braces.
- We cannot write condition with else.
- We can use logical and relational operator in condition expression. if(a>b) , if(a>b && a>c)



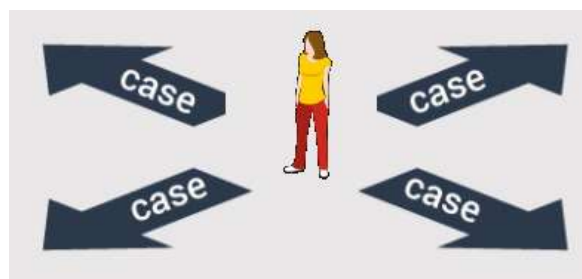


### 2.5.1.2 Switch-case

- When you are comparing the same expression to several different values it gets complicated with if...else if...else so we can use the switch-case statements as an alternative to the if-else statements.
- It executes one of several groups of statements depending on the value of an expression.
- If-else block executes different condition or expression in each statement on the other hand the switch statement evaluates a single expression and compares it for every comparison.
- If any case statement is satisfied, then the logic of that case will be executed and then the control will come out of switch block. This one will not happen with if-else block.
- Remember that switch block use break statement at the end of each case. C simply requires that we leave the block before it ends.
- If-else block are time consuming then switch case block.
- A switch statement allows a variable to be tested for equality against a list of values.
- Each value is called a case, and the variable being switched on is checked for each switch case.

Syntax#

```
switch(testexpression)
{
case expression1:
    Logic..
    break;
case expression2:
    Logic..
    break;
case expression N:
    Logic..
    break;
default:
    break;
}
```



Example# Switch case demo

```
#include<stdio.h>
#include<conio.h>
void main()
```





```
{
    int a;
    clrscr();
    printf("\nEnter option=>");
    scanf("%d",&a);
    switch(a)
    {
        case 1:
        {
            printf("\n jan");
            break;
        }
        case 2:
        {
            printf("\n feb");
            break;
        }
        case 3:
        {
            printf("\n March");
            break;
        }
        case 4:
        {
            printf("\n April");
            break;
        }
        case 5:
        {
            printf("\n May");
            break;
        }
        default:
        {
            printf("\nWrong opt");
        }
    }
    getch();
}
```

**Output:**

Enter option=>4

April

Example# Add, subtract, multiply or divide based on user choice using switch statement





```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b;
    char op;
    clrscr();
    printf("\n Enter no1:");
    scanf("%d",&a);
    printf("\n Enter no2:");
    scanf("%d",&b);
    fflush(stdin);
    printf("\n Enter option A,M,S,D=>");
    scanf("%c",&op);

    switch(op)
    {
        case 'A':
        {
            printf("\n Add=%d",a+b);
            break;
        }
        case 'M':
        {
            printf("\n Multi=%d",a*b);
            break;
        }
        case 'S':
        {
            printf("\n Sub=%d",a-b);
            break;
        }
        case 'D':
        {
            printf("\n Div=%d",a/b);
            break;
        }
        default:
        {
            printf("Wrong option");
        }
    }
    getch();
}
```

**Output:**

Enter no1:4





Enter no2:6

Enter option A,M,S,D=>A

Add=10

#### 2.5.1.2.1 Properties of Switch statements

- We don't use those expressions to evaluate switch case, which may return floating point values or strings.
- The case label values must be unique.
- The case label must end with a colon(:)
- The break is used to break out of the case statements. Break is a keyword that breaks out of the code block, usually surrounded by braces, which it is in.
- It isn't necessary to use break after each block, but if you do not use it, then all the consecutive block of codes will get executed after the matching block.
- The default case is optional, but it is wise to include it as it handles any unexpected cases. Usually default case is executed when none of the mentioned cases matches the switch expression.



#### 2.5.1.3 Difference between if and switch

If	Switch
Which statement will be executed depend upon the output of the expression inside if statement.	Which statement will be executed is decided by user.
if-else statement test for equality as well as for logical expression.	switch statement test only for equality.
if statements can evaluate float conditions.	switch statements cannot evaluate float conditions.
If the condition inside if statements is false, then by default the else statement is executed if created.	If the condition inside switch statements does not match with any of cases, for that instance the default statements is executed if created

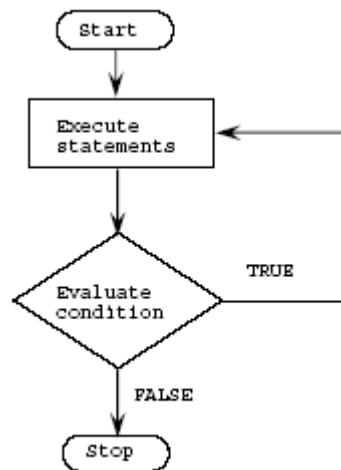




## 2.5.2 Loop Control Structure

Iteration statements are also known as Looping statements. Looping statements are used when a group of statements is to be executed repeatedly, till a condition is TRUE or until a condition is FALSE.

The following illustration shows a loop structure that runs a set of statements until a condition becomes true.



### Entry controlled loop and Exit controlled loop

In entry controlled loop, where test condition is checked before entering the loop body, known as Entry Controlled Loop.

Example# for and while loops are known as entry controlled loop

While in exit controlled loop the body of loop will be executed first and at the end the test condition is checked, if condition is satisfied than body of loop will be executed again.

Example# do while loop is known as exit controlled loop





### 2.5.2.1 For loop

- The for loop executes a block of statements a specified number of times.
- It is the most commonly and most popular loop used in any programming language.
- For loop is an entry controlled loop i.e. the condition is checked before entering into the loop. So, if the condition is false for the first time, the statements inside for loop may not be executed at all.



Syntax#

```
for(initialization; condition; increment/decrement)
{
    Logic....
}
```

#### Step 1: Initialization

Evaluates the initialization code means from where to start.

The variable initialization allows us to declare a variable and gives it a starting value.

The initialization statement is executed only once.

This part is optional and may be absent.

Example#

```
for(i=0 ; condition; increment/decrement)
for(i=20 ; condition; increment/decrement)
for(i=strlen(name) ; condition; increment/decrement)
for( ; condition; increment/decrement) // initialization is absent
```

Keep in mind that a semicolon separates each of these sections.



#### Step 2: Condition

The condition expression is evaluated. If the test expression is false, for loop is terminated at that time. But if the test expression is true, logic inside the body of for loop is executed and the update expression is updated.

The condition is checked after the execution of increment/decrement statement.

This process repeats until the test expression is false.

Usually condition is combination of Relational operator.

Example#



```
for(initialization ; i <= X; increment/decrement)
```

```
for(initialization ; i > X; increment/decrement)
```

```
for(initialization ; i < strlen(name); increment/decrement)
```

### **Step 3: Increment/Decrement**

Then it evaluates the increment/decrement condition and again follows from step 2. When the condition expression becomes false, it exits the loop.

The variable increment/decrement section is the easiest way for a for loop to handle changing of the variable. It is possible to do things like `x++`, `x = x + 10`, `x--`

Example#

```
for(initialization ; condition ; i=i+1)
```

```
for(initialization ; condition ; i++)
```

```
for(initialization ; condition ; i--)
```

```
for(initialization ; condition ; i=i+10)
```

Example# for loop demo

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i,n;
    clrscr();
    printf("\n Enter a number=");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        printf("\n%d",i);
    }
    getch();
}
```

#### **Output:**

Enter a number=4

1  
2  
3  
4

Example# To print table

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i,n;
```





```
clrscr();
printf("\nEnter a number=");
scanf("%d",&n);
for(i=1;i<=10;i++)
{
    printf("\n %d*%d=%d",n,i,n*i);
}
getch();
}
```

**Output:**

Enter a number=4

```
4*1=4
4*2=8
4*3=12
4*4=16
4*5=20
4*6=24
4*7=28
4*8=32
4*9=36
4*10=40
```

Example# To print odd and even numbers till N

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i,n;
    clrscr();
    printf("\n Enter a number=");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        if(i%2==0)
        {
            printf("\n % is even",i);
        }
        else
        {
            printf("\n %d is odd=%d",i);
        }
    }
    getch();
}
```





### Output:

Enter a number=7

1 is odd  
2 is even  
3 is odd  
4 is even  
5 is odd  
6 is even  
7 is odd

### Some good examples of for loop

<pre>for(i=2; i&lt;=n/2; i++) {     logic... }</pre>	<pre>for(i=1; i&lt;=20; i=i+2) {     logic... }</pre>
<pre>for(i=20; i&gt;=1; i--) {     logic... }</pre>	<pre>for(i=no1; i&lt;=no2; i++) {     logic... }</pre>
<pre>for(i=0 ; i&lt;strlen(name); i++) {     logic... }</pre>	<pre>for(i=1, j=n; i&lt;=n/2; i++, j--) { }</pre>
<pre>for(i=1; i&lt;=n; i++) {     for(j=1; j&lt;=n; j++)     {         Logic..     } }</pre>	<pre>for(i=1; i&lt;=n; i++) {     for(j=1; j&lt;=n; j++)     {         for(k=1; k&lt;=n; k++)         {             Logic...         }     } }</pre>

### Infinite for loop

There may be a condition in a for loop which is always true. In such case, the loop will run infinite times.

<pre>for (i=0; i&gt;0; i++) {     printf("infinite loop"); }</pre>	<pre>for (i=0; ; i++) {     printf("infinite loop"); }</pre>	<pre>for (;;) {     printf("infinite loop"); }</pre>
--	--	--





We can use break to come out of an Infinite loop.

```
for (i=0; i>0; i++)
{
    if(Some Condition)
    {
        break;
    }
}
```

### 2.5.2.2 While loop

- While loop is an entry controlled loop i.e. the condition is checked before entering into the loop. So if the condition is false for the first time, the statements inside while loop may not be executed at all.
- While loop keeps executing till the condition against which it tests remain true.
- The While statement always checks the condition before it begins the loop.
- In While loop programmer have to maintain or keep track of increment or decrement value.



An entry controls loop checks the condition first and then enters the loop body. **So for...and while..both are known as a entry controls loop.**

Syntax#

```
while(condition)
{
    logic....
}
```

Step 1: condition

The while loop evaluates the conditional expression first, If the test expression is false, while loop is terminated at a time. But if the test expression is true , logic inside the body of while loop is executed and the update expression is updated.

When the test expression is false, the while loop is terminated. The loop iterates while the condition is true.

Some good Examples

<pre>i=20; while(i&gt;50) {     logic...     i--; }</pre>	<pre>i=1; n=5; while(i&gt;=n) {     logic...     i++; }</pre>	<pre>no=123; while(no&gt;0) {     logic... }</pre>
---	---	--

Example# To print 1 to N





```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i,n;
    clrscr();
    printf("Enter a number=");
    scanf("%d",&n);
    i=1;
    while(i<=n)
    {
        printf("\n%d",i);
        i++;
    }
    getch();
}
```

**Output:**

```
Enter a number=5
1
2
3
4
5
```

Example# To print factorial of a number

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i,f=1,num;
    clrscr();
    printf("Enter a number=");
    scanf("%d",&num);
    i=1;
    while(i<=num)
    {
        f=f*i;
        i++;
    }
    printf("\nFactorial of %d is %d",num,f);
    getch();
}
```

**Output:**

```
Enter a number=5
```





### Factorial of 5 is 120

Example# Enter no and print reverse of it.

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

main()
{
    int no;
    int ans=0,x;
    clrscr();

    printf("\nEnter no =>");
    scanf("%d",&no);

    while(no>0)
    {
        x=no%10;
        ans=ans*10+x;
        no=no/10;
    }

    printf("\nReverse No = %d",ans);

    getch();
}
```

#### Output

Enter no =>123

Reverse No = 321

### Infinite while loop

There may be a condition in a while loop which is always true. In such case, the loop will run infinite times.

```
while (1)
{
    printf("This is an infinite loop");
}
```

We can use break to come out of an Infinite loop.

```
while (1)
{
    if(Some Condition)
    {
```





```

        break;
    }
}

```

### 2.5.2.3 Do while loop

- Do-while loop is an exit controlled loop i.e. the condition is checked at the end of loop.
- In do while loop, statement/logic is given before the condition, so statement or code will be executed at least one time.
- The body of do...while loop is executed once, before checking the test expression.
- In order to exit a do-while loop either the condition must be false or we should use break statement.
- Do..while loop is known as an exit controlled loop. An exit controlled loop checks the condition at the end of the loop. Hence even if the condition is false the loop will be executed at least once.



Syntax#

```

do
{
    logic...;
}while(upto condition);

```

Some good Examples

<pre> do {     Logic.. }while(op!=4); </pre>	<pre> do {     Logic.. }while(no&gt;4); </pre>
<pre> do {     <b>Loops...</b> }while(op!='s'); </pre>	

**Example# To print from 1 to N using do...while loop**

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int i,n;
    clrscr();

```







```
printf("Enter a number=");  
scanf("%d",&n);  
i=1;  
do  
{  
    printf("\n%d",i);  
    i++;  
}while(i<=n);  
getch();  
}
```

**Output:**

Enter a number=5

1  
2  
3  
4  
5

Example# Menu program for addition, square and cube

```
#include<stdio.h>  
#include<conio.h>  
void main()  
{  
    int a,b;  
    char op;  
    do  
    { clrscr();  
        printf("'a' for addition \n's' for square \n'c' for cube \n'e' for exit");  
        printf("\nEnter your choice: ");  
        scanf("%c",&op);  
        switch(op)  
        {  
            case 'a':  
            {  
                printf("Enter number 1: ");  
                scanf("%d",&a);  
                printf("Enter number 2: ");  
                scanf("%d",&b);  
                printf("Addition:%d",a+b);  
                getch();  
                break;  
            }  
            case 's':  
            {  
                printf("Enter a number: ");  
                scanf("%d",&a);
```





```

        printf("Square:%d",a*a);
        getch();
        break;
    }
    case 'c':
    {
        printf("Enter a number: ");
        scanf("%d",&a);
        printf("Cube:%d",a*a*a);
        getch();
        break;
    }
    case 'e':
    {
        break;
    }
    default:
    {
        printf("Enter a valid option!!");
    }
}
}while(op!='e');
}

```

**Output:**  
'a' for addiotn  
's' for square  
'c' for cube  
'e' for exit  
Enter your choice: s  
Enter a number: 5  
Square:25

#### 2.5.2.4 Difference between While and Do while

While	Do..while
In While loop the condition is tested first and then the statements are executed if the condition turns out to be true.	In do while the statements are executed for the first time and then the conditions are tested, if the condition turns out to be true then the statements are executed again.
These situations tend to be relatively rare, thus the simple while is more commonly used.	A do while is used for a block of code that must be executed at least once.
while loop do not run in case the condition given is false	A do while loop runs at least once even though the the condition given is false





In a while loop the condition is first tested and if it returns true then it goes in the loop	In a do-while loop the condition is tested at the last.
While loop is entry control loop	do while is exit control loop.
Syntax: while (condition) { Statements; }	Syntax: do { Statements; }while(condition);

### Infinite do while loop

There may be a condition in a do while loop which is always true. In such case, the loop will run infinite times.

```
do
{
    printf("This is an infinite loop");
}while (1);
```

We can use break to come out of an Infinite loop.

```
do
{
    if(Some Condition)
    {
        break;
    }
}while (1);
```

### 2.5.2.5 Nested Loop

Loop within loop is known as a nested loop. The depth of nested loop depends on the complexity of a problem. We can have any number of nested loops as required.

```
#include<stdio.h>
#include<conio.h>
main()
{
    int i,j,no;

    clrscr();

    printf("\nEnter no =>");
    scanf("%d",&no);

    for(i=1;i<=no;i++)
    {
        for(j=1;j<=i;j++)
        {
```





```
        printf("*");
    }
    printf("\n");
}

getch();
}
```

### Output

```
Enter no =>5
*
**
***
****
*****
```

## 2.5.2.6 Counter controlled loops or Definite repetition loop.

### 2.5.2.6.1 Conunter Controlled loop

The type of loops, where the number of the execution is known in advance are termed by the counter controlled loop.

Example#

```
for(i=1; i<=20; i++)
{
    printf("\n%d ",i);
}
```

Here, the loop will be executed exactly 10 times for n = 1,2,3,.....,20.

### 2.5.2.6.2 Sentinel controlled loop

The type of loop where the number of execution of the loop is unknown, is termed by sentinel controlled loop.

Example#

```
do{
    printf("Enter value =>\n");
    scanf("%d", &x);
}while(x > 0);
```

In the above example, the loop will be executed till the entered value of the variable x is not 0 or less then 0. This is a sentinel controlled loop and here the variable x is a sentinel variable.



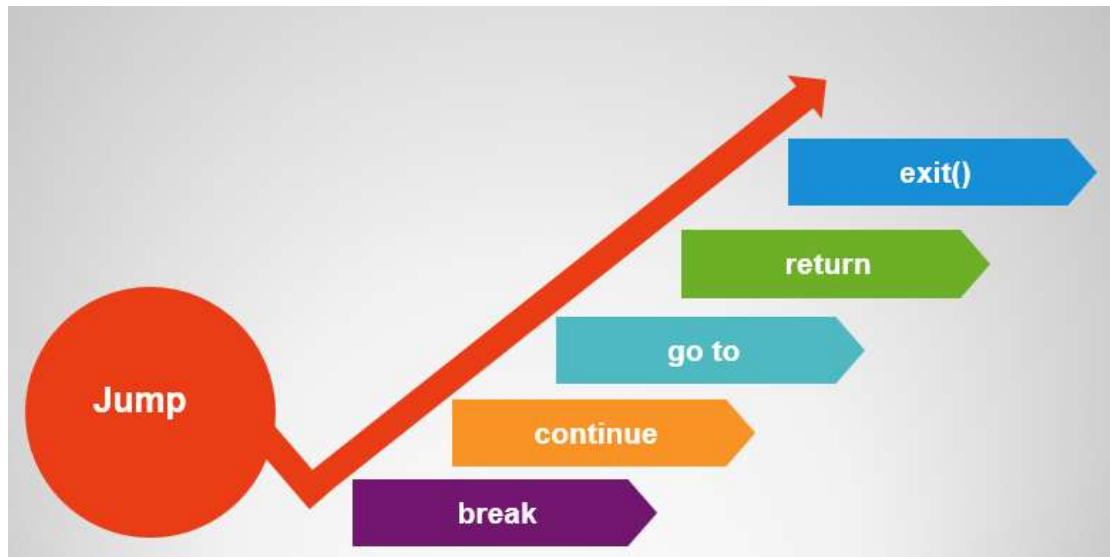


**Dr. Shyam N. Chawda, C Language Tutorial , 78 74 39 11 91**



### 2.5.3 Branching Statement

Branching statements also known as jumping statements. Jumping statements jumps from one statement to another, conditionally or unconditionally.



#### 2.5.3.1 break

- The break statement is used to stop the current execution and comes out of loop.
- In many situations, we need to get out of the loop before the loop execution is complete normally.

Syntax#  
break;

Example# To check whether a number is prime or not.

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

void main()
{
    int n,i,flag=1;
    clrscr();
    printf("Enter a number: ");
    scanf("%d",&n);
    for(i=2;i<=n/2;i++)
    {
        if(n%i==0)
        {
            flag=0;
        }
    }
}
```



```
        break;
    }
}
if(flag==0)
{
    printf("%d is not a prime number.",n);
}
else
{
    printf("%d is a prime number.",n);
}
getch();
}
```

**Output:**

Enter a number: 13  
13 is a prime number.

### 2.5.3.2 Continue

- Continue is similar to break but when break statement is executed it break the loop where as continue skips the rest of the statement and proceeds for next iteration.
- We can use Continue statment with For Loop, While Loop or do-While Loop.

Example# Print the number which is not divisible by 3

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

void main()
{
    int n,i;
    clrscr();
    printf("Enter limit: ");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        if(i%3==0)
        {
            continue;
        }
        printf("%d\n",i);
    }
    getch();
}
```





```
}
```

**Output:**

Enter limit: 20

```
1
2
4
5
7
8
10
11
13
14
16
17
19
20
```

### 2.5.3.3 goto

A goto statement in C programming provides an unconditional jump from the 'goto' to a labeled statement in the same function.

C has a goto statement which permits unstructured jumps to be made.

Note that a goto breaks the normal sequential execution of the program.

Syntax#

```
goto label;
```

```
.
```

```
.
```

```
label: statement;
```

Backward jump

The label: is before statement goto label, a loop will be formed and some statements will be executed repeatedly. Such a jump is called backward jump.

Forward jump

The label: is placed after the goto label; some statements will be skipped and the jump is known as a forward jump.

Example#

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







```
main()
{
int a;
clrscr();

pos:
printf("\nEnter Positive value =>");
scanf("%d",&a);

if(a<0)
{
    printf("\nPlz enter positive value");
    goto pos;
}
else
{
    printf("\nNo = %d and Square = %d",a,a*a);
}

getch();
}
```

**Output:**

Enter Positive value =>-2

Plz enter positive value

Enter Positive value =>-3

Plz enter positive value

Enter Positive value =>4

No = 4 and Square = 16

### 2.5.3.4 return

return statement is used for returning from a function. Whenever return statement is encountered then the control goes out from the function. With the help of return statement, we can also return some value

Example#

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

int add(int x,int y)
{
    return x+y;
```





```
}  
  
void main()  
{  
    int a,b,sum;  
    clrscr();  
    printf("Enter number 1: ");  
    scanf("%d",&a);  
    printf("Enter number 2: ");  
    scanf("%d",&b);  
    sum=add(a,b);  
    printf("Sum=%d",sum);  
    getch();  
}
```

**Output:**

```
Enter number 1: 4  
Enter number 2: 5  
Sum=9
```

### 2.5.3.5 Exit() function

exit() function terminates the calling process immediately.

We need to add "stdlib.h" to use exit() function.

Syntax#

```
void exit(int status)
```

Example#

```
exit(0)
```

The difference between break and exit() function is ,that break is a keyword, which causes an immediate exit from the switch or loop (for, while or do), while exit() is a standard library function, which terminates program execution when it is called. break is a reserved word in C; therefore it can't be used as a variable name while exit() can be used as a variable name.

Example#

```
#include<stdio.h>  
#include<conio.h>  
#include<stdlib.h>  
main()  
{  
    int i,j,no;  
  
    clrscr();  
  
    printf("\nEnter no =>");
```





```
scanf("%d",&no);

if(no<0)
{
    printf("\nBye");
    exit(0);
}
else
{
    printf("\nEntered no is positive");
}

getch();
}
```

### Output

Enter no =>-9

Bye

