Expression using Operators 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

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

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 **It takes 1 if both operands has value 1.**	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operan **The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1.**	(A | B) = 61, i.e., 0011 1101
^	Binary XOR **1 if the corresponding bits of two operands are opposite**	(A ^ B) = 49, i.e., 0011 0001
~	Binary Ones Complement 'flipping' bits- 0 changed to 1and 1 changed to 0	(~A) = -60, i.e,. 1100 0100
<<	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

Misc Operators

Operator	Description	Example
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
sizeof()	Returns the size of a variable.	int a; sizeof(a), where a is integer, will return 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
&	Returns the address of a variable.	&a; returns the actual address of the variable a .(OxFFA)
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y

Operators Precedence in C

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.

Table showing highest precedence to lowest precedence

Category	Operator	Associativity
Postfix	( ) [ ] -> . ++ - -	Left to right
Unary	Unary +,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

Expression

Expression is a combination of variables(like a,b,m,n..), constants(3,2,1) and operators(+,/*).

Eg : c+d

x/y+b+a*a*a

3.14 *r *r

Algebraic Expression	C Expression
ab-c	a*b-c
(m+n)(k+j)	(m+n)*(k+j)
(ab/c)	a*b/c
$3x^2+2x+1$	3*x^2+2*x+1

Example Program

#include<stdio.h>

Program

```
int main( )
{
int x=2,y=3,result;
result=x*5+y*7;
printf("result =:%d",result);
return 0;
}
```

Expression evaluation

result=x*5 + y*7;
result=2*5 + 3*7;
result=2*5 + 3*7;
result=10 + 3*7;
result=10 + 21;
result=31;

Example program –find greatest of 3 numbers

Example of logical(&& logical AND) and relational operators(>)

```
#include<stdio.h>
int main()
{
   int num1,num2,num3;

   printf("\nEnter value of a, b and c:");

   scanf("%d %d %d",&a,&b,&c);

   if((a>b)&&(a>c))
      printf("\n %d is greatest",a);
   else if(b>c)
      printf("\n %d is greatest",b ");
   else
      printf("\n %d is greatest",c);
   return 0;
}
```

Example program –find odd or even number

Example of Arithmetic(% mod) and relational operators(==)

```
#include<stdio.h>
int main()
{    int num,result;
   if(num%2==0)
   printf("even number \n");
   else
   printf("odd number \n");
   return 0;
```

Bitwise XOR

```
#include
<stdio.h
>int
main()
{
    int a = 12, b =
    25;
    printf("Output =
    %d", a^b);return
    0;
}
```

Output = 21

Explanation

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bitwise XOR Operation of 12 and 25
  00001100
  00011001

  00010101  = 21 (In decimal)
```

Bitwise complement 1's compliment

```
#include
<stdio.h
>int
main()
{
  printf("complement =
  %d\n",~35);return 0;
}
```

OutPut:

complement = 220

Explanation

```
35 = 00100011 (In Binary)

Bitwise complement Operation of 35
~ 00100011

  11011100  = 220 (In decimal)
```

Bitwise AND and OR operator

```
#include
<stdio.h
>int
main()
{
 int a = 12, b = 25;
 printf("OutputAND =
 %d", a&b);
 printf("OutputOR =
 %d", a|b); return 0;
}
```

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)
Bitwise AND Operation of 12 and 25
  00001100
& 00011001

  00001000  = 8 (In decimal)
Bitwise OR Operation of 12 and 25
  00001100
| 00011001

  00011101  = 29 (In decimal)
```

```
OutputAND = 8
OutputOR = 29
```

ARRAYS

<Arrays – Initialization – Declaration – One dimensional and Two-dimensional arrays.>

1. ONE DIMENTIONAL ARRAY
2. TWO DIMENTIONAL ARRAY
3. STRING ARRAYS (ONE DIMENTIONAL ARRAY and TWO DIMENTIONAL ARRAY)
4. MULTIDIMENTIONAL ARRAYS

An **array** is a collection of similar data items, accessed using a common name. The collection of element can all be integers or be all decimal value or be all characters or be all strings.

- A one-dimensional **array** is like a list
- A two dimensional **array** is like a table
- The **C** language places no limits on the number of dimensions in an **array**

ONE DIMENTIONAL ARRAY

Array Declaration:

To declare an array in C, a programmer specifies the type of the elements and the number of elements required. The **arraySize** must be an integer constant greater than zero and **datatype** can be any valid C data type.

Syntax1:

```
datatype arrayName[ arraySize ];
```

Example-1

```
int number[20];
int marks[44];
float salary[10];
double value[25];
```

```
int n=25;
double x[n], y[n]; //array declaration
```

```
int n;
Scanf("%d",&n);//get size int x[n]; //array declaration
```

```
#include<stdio.h
> #define N 100
int main( )
{
int marks[N];//array dec
....
return 0;
}
```

```
#include<stdio.h>
int main( )
{
int N=10,M=20;
int marks[N*M];//array dec
....
return 0;
```

Syntax-2

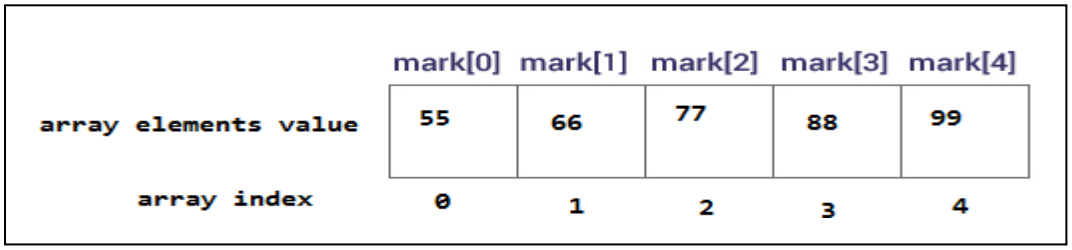
```
<storage class> datatype arrayName[ arraySize ];
```

Example: static int marks[20];

Array intitialization:

Example-1 `int mark[5] = {55, 66, 77, 88, 99};`

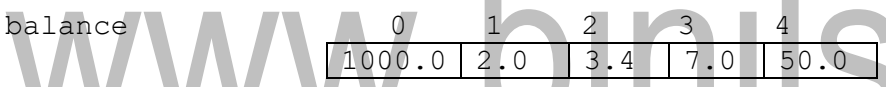
```
mark[0] = 55
mark[1] = 66
mark[2] = 77
mark[3] = 88
mark[4] = 99
```



Example-2 `double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};`

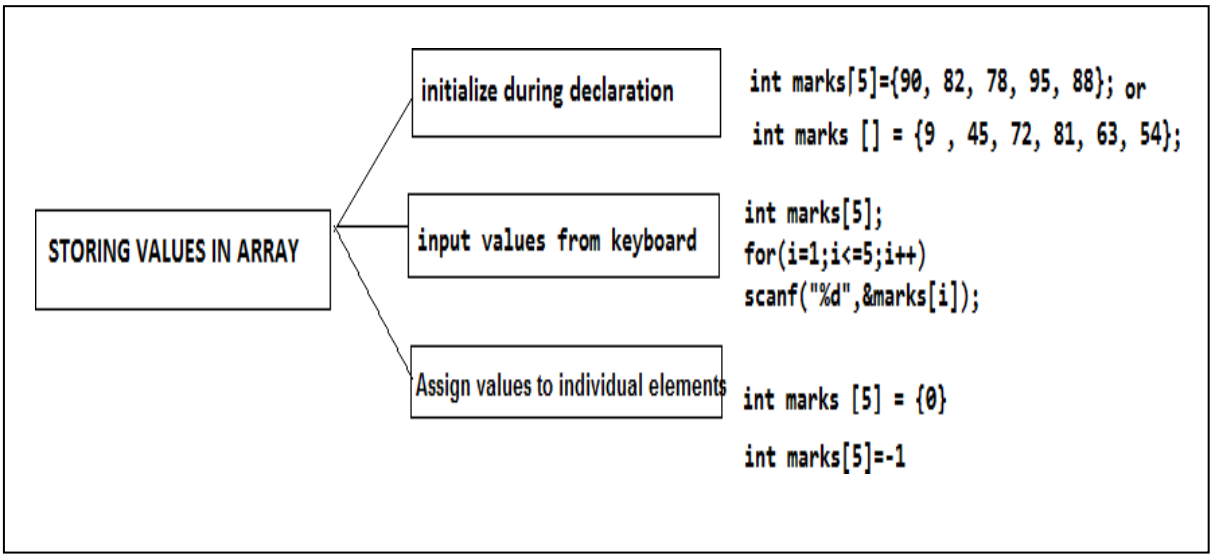
it means....

```
balance[0] = 1000.0;
balance[1] = 2.0;
balance[2] = 3.4;
balance[3] = 7.0;
balance[4] = 50.0;
```

Automatic sizing
int arr[] = {3,1,5,7,9};

Here, the C compiler will deduce the size of the array automatically based on the number of elements. Array size is deduced to be 5



OPERATIONS ON ARRAYS

- **Traversing an array**
- **Inserting an element in an array**
- **Searching an element in an array**
- **Deleting an element from an array**
- **Merging two arrays**
- **Sorting an array in ascending or descending order**

Working with one dimensional array

STORE and DISPLAY VALUES IN AN ARRAY (traversing an array)

```
#include<stdio.h>
int main( )
{
int k,array[10];//array declaration
printf("Enter the array elements:");
for(k=0;k<5;k++)
{
scanf("%d ",&array[i]); // storing values in array
}
printf("\n Display the array elements:");
for(k=0;k<5;k++)
{
printf("%d \n",array[i]);//displaying values of array
}
return 0;
}
```

Output:

```
Enter the array elements
2
4
3
1
8
Display the array elements
2
4
3
1
8
```

FIND SUM AND AVERAGE OF N NUMBERS

```
#include<stdio.h>
int main( )
{
int k,n,sum=0;array[10];//array declaration
float avg;
printf("\n Enter the array size:");
scanf("%d",&n);
printf("\n Enter the array elements:");
for(k=0;k<n;k++)
{
scanf("%d ",&array[i]); // storing values in array
}
for(k=0;k<n;k++)
{
sum=sum+array[i]; //sum of array elements
}
avg=sum/n;
printf("\n sum=%d and avg=%f ",sum,avg);
}
return 0;
}
```

Output:

```
Enter the array size: 6
Enter the array elements
9
2
4
3
1
8

sum=27 and avg=4.50000
```

REVERSE OF ARRAY ELEMENTS

```
#include<stdio.h>
int main( )
{
int k,n,array[10];//array declaration
printf("\n Enter the array size:");
scanf("%d",&n);
printf("\n Enter the array elements:");
for(k=0;k<n;k++)
```

Output:

```
Enter the array elements
2
4
3
1
8
Display the array elements
8
1
3
4
2
```

```
{
scanf("%d ",&array[i]); // storing values in array
}
printf("\n array elements in reverse order:");
for(k=n-1;k>=0;k--)
{
printf("%d \n",array[i]);//displaying values of array
}
return 0;
}
```

Write a program to print the position of the smallest number of *n* numbers using arrays.

```
#include <stdio.h>
int main()
{
int i, n, arr[20], small, pos;
printf("\n Enter the number of elements in the array : ");
scanf("%d", &n);
printf("\n Enter the elements : ");
for(i=0;i<n;i++)
scanf("%d",&arr[i]);
small = arr[0]
for(i=1;i<n;i++)
{
if(arr[i]<small)
{
small = arr[i];
pos = i;
}
}
printf("\n The smallest element is : %d", small);
printf("\n The position of the smallest element in the array is:
%d", pos);
return 0;
}
```

Output

```
Enter the number of elements in the array : 5
Enter the elements : 7 6 5 14 3
The smallest element is : 3
The position of the smallest element in the
array is : 4
```


Program example-1 Printing binary equivalent of a decimal number using array

Logic→ Here the remainders of the integer division of a decimal number by 2 are stored as consecutive array elements.The division procedure is repeated until the number becomes 0.

```
#include <stdio.h>
int main()
{
int bi[20],i,m,num,rem;
printf("\n Enter the decimal Integer");
scanf("%d",&n);
m=n;
for(i=0;i>n;i++)
{
rem=num%2;
```

Output:
Enter the decimal Integer: 12
Binary equivalent of 12 is: 1100

```
bi[i]=rem;
num=num/2;
}
printf("\n Binary equivalent of %d is: \t",m);
for(i--;i>=0;i--)
printf("%d",a[i]);
return 0;
}
```

Program example- Fibonacci series using an array

Logic→ In **Fibonacci series** each element is the sum of the previous two elements. This program stores the series in an array

```
#include <stdio.h>
int main()
{
int fib[15]; // array declaration
int i;
fib[0] = 0; // first array element value=0
fi b[1] = 1;// second array element value=1
for(i = 2; i < 15; i++)
{
 fib[i] = fib[i-1] + fib[i-2];
}
printf("\n Display the fibonacci elements:");
for(i = 0; i < 15; i++)
{
printf("%d\n", fi b[i]);
}
return 0;
}
```

```
Display the fibonacci elements
        0
        1
        1
        2
        3
        5
        8
        13
        21
        34
        55
        89
        144
        233
        377
```


Example- Inserting an Element in an Array

If an element has to be inserted at the end of an existing array, then the task of insertion is quite simple. We just have to add 1 to the upper_ bound and assign the value. Here, we assume that the memory space allocated for the array is still available. For example, if an array is declared to contain 10 elements, but currently it has only 8 elements, then obviously there is space to accommodate two more elements. But if it already has 10 elements, then we will not be ableto add another element to it.

Program to insert a number at a given location in an array

```
#include <stdio.h>
int main()
{
int i, n, num, pos, arr[10];
clrscr();
printf("\n Enter the number of elements in the array : ");
scanf("%d", &n);
for(i=0;i<n;i++)
{
printf("\n arr[%d] = ", i);
scanf("%d", &arr[i]);
}
printf("\n Enter the number to be inserted : ");
scanf("%d", &num);
printf("\n Enter the position at which the number has to be added: ");
```

```
for(i=n–1;i>=pos;i—)
arr[i+1] = arr[i];
arr[pos] = num;
n = n+1;
printf("\n The array after insertion of %d is :num ");
for(i=0;i<n;i++)
printf("\n arr[%d] = %d", i, arr[i]);
getch();
return 0;
}
```

Output

```
Enter the number of elements in the array : 5
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 4
arr[4] = 5
Enter the number to be inserted : 0
Enter the position at which the number has to be added : 3
The array after insertion of 0 is :
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 0
arr[4] = 4
arr[5] = 5
```


3.Deleting an Element from an Array

Algorithm to delete an element from the middle of an array

Step 1: [INITIALIZATION] SET I = POS
Step 2: Repeat Steps 3 and 4 while I <= N – 1
Step 3: SET A[I] = A[I + 1]
Step 4: SET I = I + 1
[END OF LOOP]
Step 5: SET N = N – 1
Step 6: EXIT

Write a program to delete a number from a given location in an array.

```
#include <stdio.h>
int main()
{
int i, n, pos, arr[10];
printf("\n Enter the number of elements in the array : ");
scanf("%d", &n);
for(i=0;i<n;i++)
{
printf("\n arr[%d] = ", i);
scanf("%d", &arr[i]);
}
printf("\nEnter the position from which the number has to be deleted : ");
```

```
for(i=pos; i<n–1;i++)
arr[i] = arr[i+1];
n– –;
printf("\n The array after deletion is : ");
for(i=0;i<n;i++)
printf("\n arr[%d] = %d", i, arr[i]);
getch();
return 0;
}
```

Output

```
Enter the number of elements in the array :
5
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 4
arr[4] = 5
Enter the position from which the number
has to be deleted : 3
The array after deletion is :
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 5
```

TWO DIMENTIONAL ARRAY

Two dimentional arrays stores data in tabular column format represented as rows and columns

Array Declaration:

datatype arrayname[size][size];

Array Initialization:

int a[2][2]={ {1,4 },{2,3}}

int b[2][2]={1,4,2,3}

1 4
2 3

float[ ][ ]={12.3, 45.2,19.3,23.4}

12.3 45.2

19.3 23.4

Accessing two-dimensional Arrays

Program-sample two dimnentional array


```
#include <stdio.h>
int main()
{
int i,j;
int a[3][2] = {{4,7},{1,0},{6,2}};
for(i = 0; i < 3; i++)
{
for(j = 0; j < 2; j++)
{
printf("%d", a[i][j]);
}
printf("\n");
}
return 0;
}
```

Row-1	4	7
Row -2	1	0
Row-3	6	2

The above array actually 'looks' like this

1	2	3	4	5	6	7	8	9
Row 0			Row 1			Row 2		

WORKING WITH TWO-DIMENSIONAL ARRAYS

Transpose of a matrix

Example program:-Transpose of a matrix

Transpose of A is $AT=(aji)$, where i is the row number and j is the column number.

$$A=\begin{pmatrix}5 & 2 & 3\\ 4 & 7 & 1\\ 8 & 9 & 9\end{pmatrix} \qquad A^T=\begin{pmatrix}5 & 4 & 8\\ 2 & 7 & 9\\ 3 & 1 & 9\end{pmatrix}$$

Program

```
#include <stdio.h>

int main()
{
   int a[10][10], transpose[10][10], r, c, i, j;
   printf("Enter rows and columns of matrix: ");
   scanf("%d %d", &r, &c);

   // getting elements of the matrix
   printf("\nEnter elements of matrix:\n");
   for(i=0; i<r; ++i)
      for(j=0; j<c; ++j)
      {
         printf("Enter element a%d%d: ",i+1, j+1);
         scanf("%d", &a[i][j]);
      }

   // Displaying the matrix a[][] */
   printf("\n Entered Matrix: \n");
   for(i=0; i<r; ++i)
      for(j=0; j<c; ++j)
      {
         printf("%d ", a[i][j]);
         if (j == c-1)
            printf("\n\n");
      }

   // Finding the transpose of matrix a
   for(i=0; i<r; ++i)
      for(j=0; j<c; ++j)
      {
         transpose[j][i] = a[i][j];
      }

   // Displaying the transpose of matrix a
   printf("\nTranspose of Matrix:\n");
   for(i=0; i<c; ++i)
      for(j=0; j<r; ++j)
      {
         printf("%d ",transpose[i][j]);
         if(j==r-1)
            printf("\n\n");
      }

   return 0;
}
```

Sample output

```
Enter rows and columns of matrix: 2
3

Enter element of matrix:
Enter element a11: 2
Enter element a12: 3
Enter element a13: 4
Enter element a21: 5
Enter element a22: 6
Enter element a23: 4

Entered Matrix:
2 3 4

5 6 4


Transpose of Matrix:
2   5

3   6

4   4
```

Program -2(Transpose)

```
#include <stdio.h>

void main()
{
    int array[10][10];
  int i, j, m, n;

   printf("Enter the order of the matrix \n");
 scanf("%d %d", &m, &n);
 printf("Enter the coefiicients of the matrix\n");
 for (i = 0; i < m; ++i)
{
        for (j = 0; j < n; ++j)
    {
            scanf("%d", &array[i][j]);
        }
    }
    printf("The given matrix is \n");
    for (i = 0; i < m; ++i)
    {
        for (j = 0; j < n; ++j)
        {
            printf(" %d", array[i][j]);
        }
        printf("\n");
    }
    printf("Transpose of matrix is \n");
    for (j = 0; j < n; ++j)
    {
        for (i = 0; i < m; ++i)
        {
            printf(" %d", array[i][j]);
        }
        printf("\n");
    }
```



```
$ cc pgm85.c
$ a.out
Enter the order of the matrix
3 3
Enter the coefiicients of the matrix
3 7 9
2 7 5
6 3 4
The given matrix is
 3 7 9
 2 7 5
 6 3 4
Transpose of matrix is
 3 2 6
 7 7 3
 9 5 4
```

Matrix addition and subtraction

Addition If *A* and *B* above are matrices of the same type

$$A+B=\begin{pmatrix}1 & 2 & 3\\ 1 & 0 & 2\end{pmatrix}+\begin{pmatrix}2 & 1 & 2\\ 1 & 0 & 3\end{pmatrix}=\begin{pmatrix}3 & 3 & 5\\ 2 & 0 & 5\end{pmatrix}$$

Subtraction If *A* and *B* are matrices of the same type, then

$$A-B=\begin{pmatrix}1 & 2 & 3\\ 1 & 0 & 2\end{pmatrix}-\begin{pmatrix}2 & 1 & 2\\ 1 & 0 & 3\end{pmatrix}=\begin{pmatrix}-1 & 1 & 1\\ 0 & 0 & -1\end{pmatrix}$$

Program to Add Two Matrices

```
#include <stdio.h>
int main(){
   int r, c, a[100][100], b[100][100], sum[100][100], i, j;

   printf("Enter number of rows (between 1 and 100): ");
   scanf("%d", &r);
   printf("Enter number of columns (between 1 and 100): ");
   scanf("%d", &c);
   printf("\nEnter elements of 1st matrix:\n");
   for(i=0; i<r; ++i)
      for(j=0; j<c; ++j)
      {
         printf("Enter element a%d%d: ",i+1,j+1);
         scanf("%d",&a[i][j]);
      }
   printf("Enter elements of 2nd matrix:\n");
   for(i=0; i<r; ++i)
      for(j=0; j<c; ++j)
      {
         printf("Enter element a%d%d: ",i+1, j+1);
         scanf("%d", &b[i][j]);
      }
   // Adding Two matrices
   for(i=0;i<r;++i)
      for(j=0;j<c;++j)
      {
         sum[i][j]=a[i][j]+b[i][j];
      }
  // Displaying the result
   printf("\nSum of two matrix is: \n\n");
   for(i=0;i<r;++i)
      for(j=0;j<c;++j)
      {
         printf("%d ",sum[i][j]);
         if(j==c-1)
         {
            printf("\n\n");
         }
      }
      return 0;
}
```

Output

```
Enter number of rows (between
1 and 100): 2
Enter number of columns
(between 1 and 100): 3

Enter elements of 1st matrix:
Enter element a11: 2
Enter element a12: 3
Enter element a13: 4
Enter element a21: 5
Enter element a22: 2
Enter element a23: 3
Enter elements of 2nd matrix:
Enter element a11: -4
Enter element a12: 5
Enter element a13: 3
Enter element a21: 5
Enter element a22: 6
Enter element a23: 3

Sum of two matrix is:

-2   8   7

10   8   6
```

Matrix multiplication

Matrix multiplication for two 2 × 2 matrices.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} e & f \\ g & h \end{pmatrix}=\begin{pmatrix} (ae+bg) & (af+bh) \\ (ce+dg) & (ef+dh) \end{pmatrix}$$

Finding norm of a matrix

The norm of a matrix is defi ned as the square root of the sum of the squares of the elements of a matrix.

```
#include <stdio.h>
#include <math.h>
#defi ne row 10
#defi ne col 10
int main()
{
fl oat mat[row][col], s;
int i,j,r,c;
printf("\n Input number of rows:");
scanf("%d", &r);
printf("\n Input number of cols:");
scanf("%d", &c);
for(i = 0 ; i< r; i++)
{
for(j = 0 ;j<c; j++)
{
scanf("%f", &mat[i][j]);
}
}
printf("\n Entered 2D array is as follows:\n");
for(i = 0; i < r; i++)
{
for(j = 0; j < c; j++)
{
printf("%f", mat[i][j]);
}
printf("\n");
}
s = 0.0;
for(i = 0; i < r; i++)
{
for(j = 0; j < c; j++)
{
s += mat[i][j] * mat[i][j];
}
}
printf("\n Norm of above matrix is: %f", sqrt(s));
return 0;
}
```

C Program to read a matrix and find sum, product of all elements of two dimensional (matrix) array

```
include <stdio.h>
#define MAXROW     10
#define MAXCOL     10
int main()
{
        int matrix[MAXROW][MAXCOL];
        int i,j,r,c;
        int sum,product;

        printf("Enter number of Rows :");
        scanf("%d",&r);
        printf("Enter number of Cols :");
        scanf("%d",&c);

        printf("\nEnter matrix elements :\n");
        for(i=0;i< r;i++)
        {
                for(j=0;j< c;j++)
                {
                        printf("Enter element [%d,%d] : ",i+1
                        scanf("%d",&matrix[i][j]);
                }
        }
        sum=0;
        product=1;

        for(i=0;i< r;i++)
        {
                for(j=0;j< c;j++)
                {
                        sum+=matrix[i][j];
                        product*= matrix[i][j];
        }       }
printf("\nSUM of all elements : %d \nProduct of all elements :%d",sum,product);
        return 0;
}
```

```
Enter number of Rows :3
Enter number of Cols :3

Enter matrix elements :
Enter element [1,1] : 1
Enter element [1,2] : 1
Enter element [1,3] : 1
Enter element [2,1] : 2
Enter element [2,2] : 2
Enter element [2,3] : 2
Enter element [3,1] : 3
Enter element [3,2] : 3
Enter element [3,3] : 3

SUM of all elements : 18
Product of all elements :216
```


Find the sum of diagonal elements of a matrix

```
#include < stdio.h >
int main()
{
        int a[10][10],i,j,sum=0,r,c;
        clrscr();
        printf("\n Enter the number of rows and column ");
        scanf("%d%d",&r,&c);
        printf("\nEnter the %dX%d matrix",r,c);
        for(i=0;i < r;i++)
        {
                for(j=0;j < c;j++)
                {
                        scanf("%d",&a[i][j]);
                }//for
        }//for

        for(i=0;i < r;i++)
        { for(j=0;j < c;j++)
                { if(i==j)
                        {
                                sum+=a[i][j];
                        }
                }//for
```

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

Sum of diagonal=13
```

```
        }//for
        printf("\nThe sum of diagonal elements is %d",sum); return 0;
}//main
```

Sum of rows and columns

```
#include <stdio.h>
 void main ()
 {
int array[10][10];
int i, j, m, n, sum = 0;
printf("Enter the order of the matrix\n");
scanf("%d %d", &m, &n);
printf("Enter the co-efficients of the matrix\n");
for (i = 0; i < m; ++i)
      {
         for (j = 0; j < n; ++j)
          {
             scanf("%d", &array[i][j]);
          }
       }
         for (i = 0; i < m; ++i)
       {
          for (j = 0; j < n; ++j)
          {
             sum = sum + array[i][j] ;
          }
           printf("Sum of the %d row is = %d\n", i, sum);
          sum = 0;
          }
       sum = 0;
       for (j = 0; j < n; ++j)
          {
           for (i = 0; i < m; ++i)
          {
             sum = sum + array[i][j];
          }
           printf("Sum of the %d column is = %d\n", j, sum);
          sum = 0;
       }
    }
```

Output

```
Enter the order of the matrix
2 2
Enter the co-efficients of the matrix
23 45

80 97
Sum of the 0 row is = 68
Sum of the 1 row is = 177
Sum of the 0 column is = 103
Sum of the 1 column is = 142
```

C Program to do the Sum of the Main & Opposite Diagonal Elements of a MxN Matrix

```
#include <stdio.h>
void main ()
    {           static int array[10][10];
       int i, j, m, n, a = 0, sum = 0;
         printf("Enetr the order of the matix \n");
       scanf("%d %d", &m, &n);
         if (m == n )
       {
          printf("Enter the co-efficients of the matrix\n");
         for (i = 0; i < m; ++i)
         {
            for (j = 0; j < n; ++j)
            {
               scanf("%d", &array[i][j]);
            }
         }
           printf("The given matrix is \n");
         for (i = 0; i < m; ++i)
         {
            for (j = 0; j < n; ++j)
            {
               printf(" %d", array[i][j]);
            }
            printf("\n");
         }

         for (i = 0; i < m; ++i)
         {
            sum = sum + array[i][i];
            a = a + array[i][m - i - 1];
         }
       printf("\nThe sum of the main diagonal elements is = %d\n", sum);
       printf("The sum of the off diagonal elements is = %d\n", a);
         }
        else
         printf("The given order is not square matrix\n");
      }
```

```
Enter the order of the matix
2 2
Enter the co-efficients of the matrix
40 30
38 90
The given matrix is
 40 30
 38 90

The sum of the main diagonal elements is
= 130
The sum of the off diagonal elements is
= 68
```


C Program to Find the Frequency of Odd & Even Numbers in the given Matrix

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

          static int array[10][10];
          int i, j, m, n, even = 0, odd = 0;

          printf("Enter the order ofthe matrix \n");
          scanf("%d %d", &m, &n);

          printf("Enter the coefficients of matrix \n");
          for (i = 0; i < m; ++i)
      {
           for (j = 0; j < n; ++j)
           {
              scanf("%d", &array[i][j]);
              if ((array[i][j] % 2) == 0)
              {
                 ++even;
```

```
                }
                else
                   ++odd;
              }

            }

            printf("The
            given matrix is
            \n");for (i = 0; i
            < m; ++i)
        {
           for (j = 0; j < n; ++j)
           {
              printf(" %d", array[i][j]);
           }
           printf("\n");
        }
printf("\n The frequency of occurrence of odd number = %d
\n", odd);printf("The frequency of occurrence of even number =
%d\n", even);

    }
```



```
Enter the
order of the
matrix3 3
Enter the
coefficients of
matrix34 36 39
23 57 98
12 39 49
The
 giv
 en
 ma
 trix
 is
 34
 36
 39
 23 57 98
 12 39 49
```

The frequency of occurrence of odd number = 5The frequency of occurrence

Compile and Link C Program

There are three basic phases occurred when we execute any C program.

- Preprocessing
- Compiling
- (assembler)
- Linking

Preprocessing Phase :A C pre-processor is a program that accepts C code with preprocessing statements and produces a pure form of C code that contains no preprocessing statements (like #include).

Compilation Phase: The C compiler accepts a preprocessed output file from the preprocessor and produces a special file called an object file. Object file contains machinecode generated from the program.

Linking Phase: The link phase is implemented by the linker. The linker is a process that accepts as input object files and libraries to produce the final executable program.

Compiling and Linking a C program is a multi-stage process.

The process can be split into four separate stages: **Preprocessing**, **compilation**, **assembly**, **and linking**.


```
/*
 * "Hello, World!": A classic.
 */

#include

<stdio.h>int

main(void)
{
        puts("Hello,
        World!");return 0;
```

Preprocessing

The first stage of compilation is called preprocessing. In this stage, lines starting with a # character are interpreted by the *preprocessor* as *preprocessor commands*. These commands form a simple macro language with its own syntax and semantics.

reduce repetition in source code
invoke inline files
define macros
joining continued lines (lines ending with a \)
removes comments.

To print the results of the preprocessing stage, pass the -E option to cc:

```
gcc -E hello_world.c
```

[lines omitted for brevity]

```
extern int_vsnprintf_chk (char * restrict, size_t,
      int, size_t, const char * restrict, va_list);
# 493 "/usr/include/stdio.h" 2 3 4
# 2 "hello_world.c" 2

int main(void) {
 puts("Hello, World!");
 return 0;
}
```

Compilation

In this stage, the preprocessed code is translated to *assembly instructions* These form an intermediate readable language.

Some compilers also supports the use of an integrated assembler, in which the compilation stage generates *machine code* directly, avoiding the overhead of generating the intermediate assembly instructions and invoking the assembler. *object code is directly produced by compiler.*

to view the result of the compilation stage, pass the -S option to cc:

```
gcc -S hello_world.c
```

This will create a file named hello_world.s

```
.section    TEXT,__text,regular,pure_instructions
    .macosx_version_min 10, 10
    .globl  _main
    .align  4, 0x90
_main:                                  ## @main
    .cfi_startproc
## BB#0:
    pushq   %rbp
Ltmp0:
    .cfi_def_cfa_offset 16
Ltmp1:
    .cfi_offset %rbp, -16
    movq    %rsp, %rbp
Ltmp2:
    .cfi_def_cfa_register %rbp
    subq    $16, %rsp
    leaq    L_.str(%rip), %rdi
    movl    $0,  -4(%rbp)
    callq   _puts
    xorl    %ecx, %ecx
    movl    %eax, -8(%rbp)          ## 4-byte Spill
```

```
    movl    %ecx, %eax
    addq    $16, %rsp
    p
    o
    p
    q
            %rbpretq
    .cfi_endproc

    .section    TEXT,_cstring,cstring_literals
L_.str:                                 ## @.str
    .asciz  "Hello, World!"
```

Assembly

During the assembly stage, an assembler is used to translate the assembly instructions tomachine code, or *object code*.
--The output consists of actual instructions to be run by the target processor.

Input to assembler :cc -c
hello_world.c Output of
assembler phase: hello-worls.o
The contents of this file is in a binary format and can be viewed using hexdump or odcommands:

gcc -C hello_world.c

Linking

The object code generated in the assembly stage is composed of machine instructions that theprocessor understands but some pieces of the program are out of order or missing. To produce an executable program, the existing pieces have to be rearranged and the missing ones filled in. This process is called linking.

The result of this stage is the final executable program(.exe)

Finally to run

a.out hello_world.c

Commands to compile and execute C program

Save the program file using .c

extention To compile:

gcc filename.c

To run the program

./a.out

Constants or Literals and Variables

A constant is a value or an identifier whose value cannot be altered in a program.

For example: 1, 2.5, "C programming is easy", ‘apple’ etc.

We can define constants in a C program in the following ways.

1. **By “const” keyword**
2. **By “#define” preprocessor directive**

Syntax1: **const type constant_name**;
Eg 1: **const double PI = 3.14** //variable *PI* is a constant ,3.14 cannot be changed

Eg-1

```
#include<stdio.h>
main()
{
  const int SIDE = 10;
  int area;
  area = SIDE*SIDE;
  printf("The area of the square %d is: %d sq. units" , SIDE, area);
}
```

Output

The area of the square 10 is: 100 sq. Units

Syntax 2: **#define variable value**
Eg-2 **#define PI 3.14**

Constants can be classified into broad categories

C constant

Primary Constants
-integer constant
-floating point constant
-character Constant
-String Constant
-Backslash Constant

1. Integer constants

An integer constant is a numeric constant (associated with number) without any fractional or exponential part. There are three types of integer constants in C programming:

- **decimal constant(base 10)**
- **octal constant(base 8)**
- **hexadecimal constant(base 16)**

For example:
Decimal constants: 1,0, -9, 22 etc
Octal constants: 021, 077, 033 etc
Hexadecimal constants: 0x7f, 0x2a, 0x521 etc

In C programming, octal constant starts with a 0 and hexadecimal constant starts with a 0x.

55 /*int constant */
55l /*unsigned int constant*/
55 ul /*unsigned long constant*/

Rules for defining integer constants:

- An integer constant must have at least one digit.
- It must not have a decimal point.
- It can either be positive or negative.
- No commas or blanks are allowed within an integer constant.
- If no sign precedes an integer constant, it is assumed to be positive.
- The allowable range for integer constants is -32768 to 32767.

2. Floating-point constants

A floating point constant is a numeric constant that has either a fractional form or an exponent form(decimal point). For example**:**

-2.0
0.0000234
-0.22E-5

6.333 –correct

633E-4L-

633E---illegal..incomplete exponent
633f—illegal..no decimal or exponent
.e633—illegal..missing integer

Rules for defining floating point(real) constants:

- A real constant must have at least one digit
- It must have a decimal point
- The mantissa part and exponential part should be separated by a letter e/E
- The mantissa part must have a positive or negative sign. The default sign of mantissa part is positive.
- No commas or blanks are allowed within a real constant.

3. Character constants

A character constant is a constant which uses single quotation around characters.

For example:

'a'
'6',
'=',
'F'

Rules for defining character constants

- A character constant is a single alphabet, a single digit or a single special symbol enclosed within single quotes.
- The maximum length of a character constant is 1 character.

4. String constants

String constants are the constants which are enclosed in a pair of double-quote marks. For example:

"good" //string constant
"" //null string constant
" " //string constant of six white space
"x" //string constant having single character.

String constants are enclosed within double quotes.

5. Backslash Character Constants in C:

- There are some characters which have special meaning in C language.
- They should be preceded by backslash symbol.(\)

Backslash character	Meaning
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\”	Double quote
\’	Single quote
\\	Backslash
\v	Vertical tab
\a	Alert or bell
\?	Question mark
\N	Octal constant (N is an octal constant)
\XN	Hexadecimal constant (N – hex.dcml cnst)

Example Program Using Const Keyword

```
#include<stdio.h>
int main( )
{
int const BASE ,HEIGHT;
float area;
char NEWLINE=’\n’;
area=0.5*BASE*HEIGHT
printf(“The Area of Triangle is:”);
printf(“%c”,NEWLINE);
printf(“%f”,area);
return 0;
}
```

```
OUTPU
T 10 20
The Area of Triangle is:

100
```

Using # Define

```
#include<stdio.h>
#define BASE 100
#define HEIGHT 100
#define NEWLINE ‘\n’
int main( )
{
float area;
area=0.5*BASE*HEIGHT
printf(“The Area of Triangle is:”);
printf(“%c”,NEWLINE);
printf(“%f”,area);
return 0;
```

```
OUTPU
T 10 20
The Area of Triangle is:

100
```

Example program using const keyword in C:

```
#include <stdio.h>
void main()
{
const int height = 100; /*int constant*/
const float number = 3.14; /*Real constant*/
const char letter = 'A'; /*char constant*/
const char letter_sequence[10] = "ABC"; /*string constant*/
const char backslash_char = '\?'; /*special char cnst*/
printf("value of height :%d \n", height );
printf("value of number : %f \n", number );
printf("value of letter : %c \n", letter );
printf("value of letter_sequence : %s \n", letter_sequence);
printf("value of backslash_char : %c \n", backslash_char);
}
```

Output:

```
value of height : 100
value of number : 3.140000
value of letter : A
value of letter_sequence : ABC
value of backslash_char : ?
```

2. Example program using #define preprocessor directive in C:

```
#include <stdio.h>
#define height 100
#define number 3.14
#define letter 'A'
#define letter_sequence "ABC"
#define backslash_char '\?'
void main()
{
printf("value of height : %d \n", height );
printf("value of number : %f \n", number );
printf("value of letter : %c \n", letter );
printf("value of letter_sequence : %s \n",letter_sequence);
printf("value of backslash_char : %c \n",backslash_char);
}
```

Output:

```
value of height : 100
value of number : 3.140000
value of letter : A
value of letter_sequence :
ABC value of
backslash_char : ?
```

Difference between variable and constants

The **difference between variables** and **constants** is that **variables** can change their value at any time but **constants** can never change their value.

Variable	**Constant variable**
int a =10; **a++;** **printf("%d",a);** **----------** **o/p:= 11**	**const int a =10;** **a++;** **printf("%d",a);** **----------** **o/p:= 10**

Variable is the name of memory location which holds the data. Unlike constant, variables are changeable, value of a variable can be changed during execution of a program. A programmer must chose a meaningful variable name.

Variables are used for holding data values so that they can be utilized for various computations in a program.A variable must be declaed and then used for coputation work in program./A variable is an identifier used for storing and holding some data(value).

All variables have three important attributes.

1.A ***data type:*** ***Like*** int, double, float. Once defined,the type of a C variable cannot be changed.
2.A ***name*** of the variable.
3.A ***value*** that can be changed by assigning a new value to the variable. The kind of values a variable can assume depends on its type.
Eg : for variable int salary,it can only take integer values can only take integer values like 65000 and not 6500.0

Rules For Constructing Variables

1. A variable name can be a combination of alphabets, numbers and special character underscore(_).
2. The first character in the variable name must be an alphabet.
3. No commas or blank spaces are available are allowed within a variable name.
4. No special symbol other than an underscore is allowed.
5.Upper and Lower case names are treated as different, as C is case sensitive, so it is suggested to keep the variable names in lower case.

Declaring and Initializing a variable:=

→Declaration of a variable must be done before it is used for any computation in the program.
→Declaration tells the compiler what the variable name is.
→ Declaration tells what type of data the variable will hold.
→Until the variable is not defined/or/declared compiler will not allocate memory space to the variables.
→ A variable can also be declared outside main() function.
→A variable can also be declared in other program and declared using extern keyword.

```
int yearly_salary;
float monthly_salary;
int a;
double x;
int ECE1111;
```

Initializing a variable:=
Initializing a variable means to provide a value to variable

```
int yearly salary=5,00,000
float monthly_salary= 41666.66
```

Difference between identifier and variable

Identifier	Variable
Indentifier is the name given to a variable,function etc.	While variable is used to name a memory location which stores data
An identifier can be a variable ,but not all identifiers are variables	All variables names are identifiers

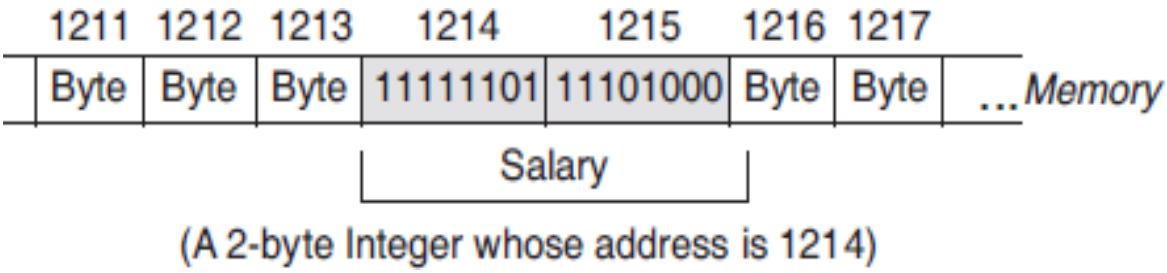
Example : void average()
{
}
	Example: int average

Variables are a way of reserving memory to hold some data and assign names to them so thatwe don't have to remember the numbers like REG46735 or memory address like FFFFoxFF and instead we can use the memory location by simply referring to the variable.

Every variable is mapped to a unique

[[[[[[note A computer memory is made up of registers and cells. It accesses data in a collection of bits, typically 8 bits, 16 bit, 32 bit or 64 bit. A computer memory holds information inthe form of binary digits 0 and 1 (bits).**]]]]]]**


```
int salary = 65000;
```



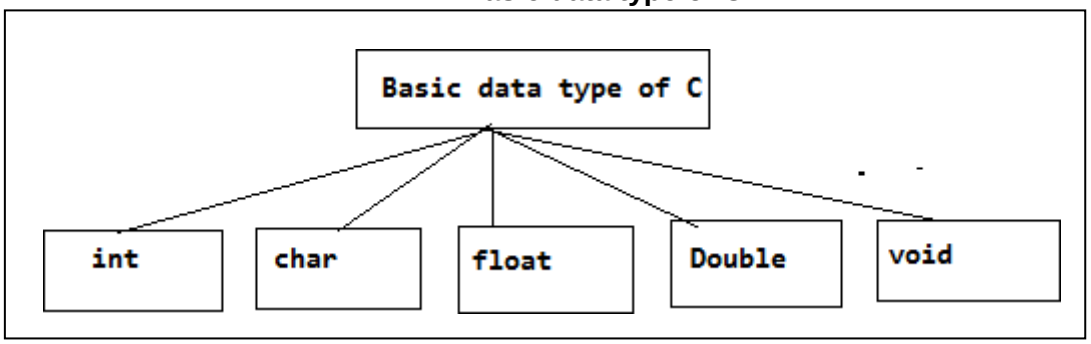
DATA TYPES IN C

→The data type, of a variable determines a set of values that a variable might take and a set of operations that can be applied to those values.

→Data type refer to the type and size of data associated with the variable and functions.

→Data types can be broadly classified as shown in Figure

Basic data type of C



	Data Type	Size in Bytes	Range	Format-Specifier
int	int	2	-32768 to +32767	%d
	short signed int (or) signed int	2	32768 to +32767	%d
	short unsigned int (or) unsigned int	2	0 to 65535	%u
	long signed int (or) long int	4	-2147483648 to 2147483647	%ld
	long unsigned int	4	0 to 4294967295	%lu
char	char or signed char	1	-128 to 127	%c
	unsigned char	1	0 to 255	%c
	float Allows 6 digits after decimal point.	4	$-3.4e^{-38}$ to $+3.4e^{38}$	%f
	double Allows 15 digits after decimal point.	8	$-1.7e^{-308}$ to $+1.7e^{308}$	%lf
	long double Allows 15 digits after decimal point.	10	$-1.7e^{-4932}$ to $1.7e^{4932}$	%LF

/*Program*/

```
#include<stdio.h>
int main()
{
char a;
unsigned char b;
int i;
unsigned int j;
long int k;
unsigned long int m;
float x;
double y
long double z;

printf("\n char and unsigned char");
scanf("%c %c",&a,&b) //get char and unsigned char value
printf("%c %c",a,b) //display char and unsigned char value

printf("\n int unsigned int");
scanf("%d %u",&i,&j) //get int unsigned int value
printf("%d %u",i,j) //display int unsigned int value

printf("\n long int unsigned long int");
scanf("%ld %lu",&i,&j) //get long int and long unsigned int value
printf("%ld %lu",i,j) //display int unsigned int value

printf("\n float,double and long double");
scanf("%f %lf %Lf",&i,&j) //get float,double and long double value
printf("%f %lf %Lf",i,j) //display float,double and long double value

return 0;
}
```

The specifiers and qualifiers for the data types can be broadly classified into three types

- ***Size specifiers***— short and long
- ***Sign specifiers***— signed and unsigned
- ***Type qualifiers***— const, volatile and restrict.

Size qualifiers alter the size of the basic data types. There are two such qualifiers that can be used with the data type int; these are short and long.

short, when placed in front of the data type int declaration, tells the C compiler that the particular variable being declared is used to store fairly small integer values. **Long** specifies it is a very big integer value.Long integers require twice the memory of than small ints.

Table: Sizes (bytes) of short int ,int,long int

	16-bit Machine (size in bytes)	16-bit Machine (size in bytes)	16-bit Machine (size in bytes)
short int	2	2	2
int	2	4	4
long int	4	4	8

Table:Size and range of *long long* type (64-bit machine)

Data type	Size (in bytes)	Range
long long int	8	-9, 223, 372, 036, 854, 775, 808 to +9, 223, 372, 036, 854, 775, 808
unsigned long int or unsigned long	4	0 to + 4, 294, 967, 295
unsigned long long int or unsigned long long	8	0 to + 18, 446, 744, 073,709, 551, 615

Sign specifiers: for example fot int data type out of 2bytes(2*8=16bits) of its size the highest bit(the sixtheenth bit) is used to store the sign of the integer value. The bit is 1 if number is negative and 0 if the number is positive.

Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Bit 8	Bit 9	Bit 10	Bit 11	Bit 12	Bit 13	Bit 14	Bit 15	Bit 16
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Sign of number (1 for –ve and 0 for +ve0

Type qualifiers : There are two type qualifiers, const and volatile;

Eg: **const float pi = 3.14156;** // specifies that the variable pi can never be changed by the Program.

Table:Size and range in (16-bit machines)

Data type	Size (in bits) note:[1byte=8bits]	Range
char	8	–128 to 127
int	16	–32768 to 32767
float	32	1.17549×10^{-38} to 3.40282×10^{38}
double	64	2.22507×10^{-308} to 1.79769×10^{308}
Void	8	valueless

Table:Size and range of (32-bit machine)

Data type	Size (in bits) note:[1byte=8bits]	Range
char	8	–128 to 127
int	32	–2147483648 to 2147483647
float	32	1.17549×10^{-38} to 3.40282×10^{38}
double	64	2.22507×10^{-308} to 1.79769×10^{308}
Void	8	valueless

Data Type	Size (bits)	Range
char	8	–128 to 127
unsigned char	8	0 to 255
signed char	8	–128 to 127
int	16	–32768 to 32767
unsigned int	16	0 to 65535
signed int	16	–32768 to 32767
short int	16	–32768 to 32767
unsigned short int	16	0 to 65535
signed short int	16	–32768 to 32767
long int	32	–2147483648 to 2147483647
unsigned long int	32	0 to 4294967295
signed long int	32	–2147483648 to 2147483647
float	32	3.4E–38 to 3.4E+38
double	64	1.7E–308 to 1.7E+308
long double	80	3.4E–4932 to 1.1E+4932

Download Binils Android App in Playstore Download Photoplex

Decision Making and Branching

Conditional Branching	Conditional Branching
→if statement →nested if statement →if ..else statement →nested if else statement	→break →continue →goto

These include conditional type branching and unconditional type branching.

if statement

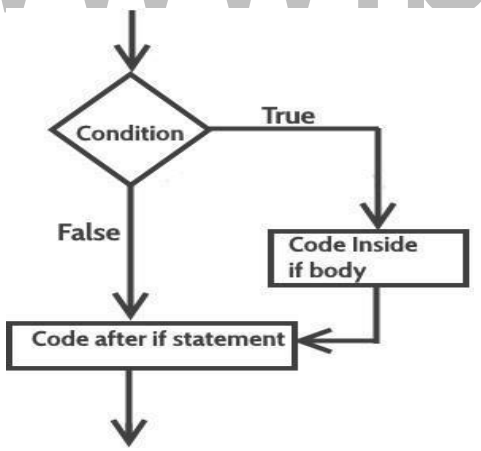
It takes the following form

```
if(test-expression)
```

It allows the computer to evaluate the expression first and them depending on whether the value of the expression is "true" or "false", it transfer the control to a particular statement. This point of program has two paths to flow, one for the true and the other for the false condition.

```
if(test-expression)
{
statement-block
}
statement-x;
```

The statement-block may be a single statement or group of statements. If the test expression is true, the statement-block will be executed; otherwise the statement-block will be skipped and the execution will jump to the statement-x. But when is condition true both the statement-block and the statement-x are executed in sequence.



Eg-2

```
if (code = = 1)
{
salary = salary + 500;
}
printf("%d",salary);
```

Eg: Example program: C Program to check equivalence of two numbers using if statement

```
#include<stdio.h>
void main()
{
int m,n;
clrscr();
printf(" \n enter two numbers:");
scanf(" %d %d", &m, &n);
if(m-n= = 0)
{
printf(" \n two numbers are equal");
}
getch();
}
```

o/p

4 4

two numbers are equal

Nested if

The syntax for a nested if statement is as follows

```
if( cond 1)
{
            /* Executes boolean expression when cond 1 is true */
   if(cond 2) {
              /* Executes when the boolean expression 2 is true */
   }
}
```

Example:

```
#include <stdio.h>
 int main ()
{
   int a = 100;
   int b = 200;
       if( a == 100 ) {
                          /* if condition is true then check the following */
      if( b == 200 ) {
                          /* if condition is true then print the following */

         printf("Value of a is 100 and b is 200\n" );
      }
   }
      printf("Exact value of a is : %d\n", a );
   printf("Exact value of b is : %d\n", b );
    return 0;
}
```

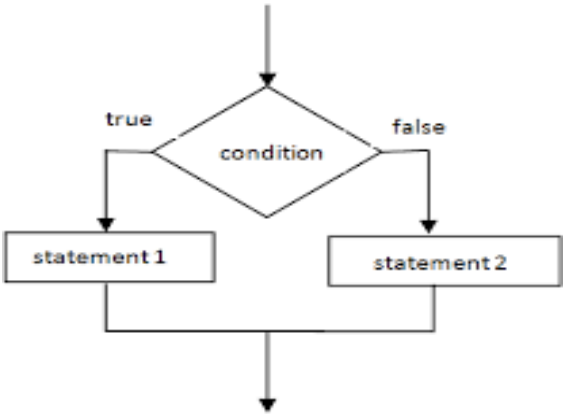

The if-else statement

The if-else statement is an extension of the simple if statement. The general form is If the test- expression is true, then true-block statements immediately following if statement are executed otherwise the false-block statements are executed.

```
if(test-expression)
{true-block statements
}
else
{
false-block statements
}statement-x
```

Example: C program to find largest of two numbers

```
#include<stdio.h>
int main()
{
int m,n,large;
printf(" \n enter two
numbers:"); scanf(" %d %d",
&m, &n); if(m>n)
large=
m; else
large=n
;
printf(" \n large number is = %d",
large); return 0;
}
```



Nested if-else statement

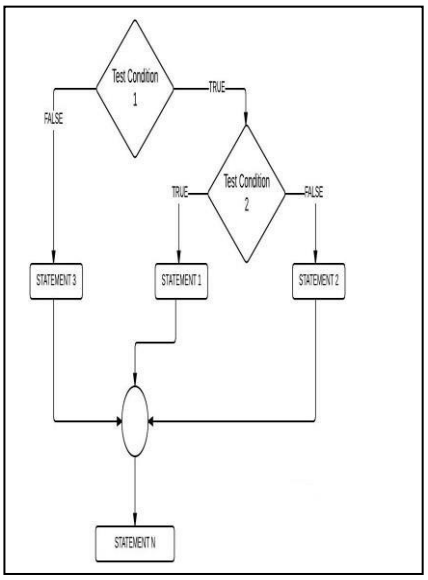
Nested if construct is also known as **if-else-if** construct.

Syntax-1

```
if(test-condition-1)
{
   (stmts)
else
   {
        if(condition 2)
       {
         Statement-1;
       }
    else
      {
        statement-2;
      }
}
}
   statement-x
```

Syntax-2

```
If(test-condition-1)
 {
    if(test-condition-2)
   {
            statement-1;
   }
    else
      {
         statement-2;
      }
}
else
{
statement-3;
}
   statement-x
```



If the test-condition-1 is false, the statement-3 will be executed; other wise it continues the second test. If the condition-2 is true, the statement-2 will be evaluated and then the control is transferred to the statement-x.

Example: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;
}
```

The switch statement:

When many conditions are to be checked then using nested if...else is very difficult, confusing and cumbersome.So C has another useful built in decision making statement known as switch.

This statement can be used as multiway decision statement.The switch statement tests the value of a given variable or expression against a list of case values and when a match is found, a block of statements associated with that case is executed.

Eg-1

```
int i = 1;
switch(i)
{
case 1:
        printf("A");
         break;
case 2:
         printf("B");
         break;
case 3:
         printf("C");
         break;
default:
 }
```

```
switch( code)
{
case 1:
        stmts1;
         break;
case 2:
         stmts2;
        break;
case 3:
         stmts3
         break;
 default:
         stmtsx
}
```

Example2: C program to find largest of two numbers

```
#include<stdio.h>
void main ()
{
float basic , da ,
salary ; int code ;
char
name[25];
da=0.0;
printf("Enter employee
name\n");
scanf("%[^\n]",name);
printf("Enter basic salary\n");
scanf("%f",&basic);
printf("Enter code of the
Employee\n"); scanf("%d",&code);
switch (code)
{case 1:
da = basic * 0.10;
break;
case 2:
da = basic * 0.15;
break;
case 3:
da = basic * 0.20;
break; default :
da = 0;
} salary = basic + da; printf("Employee name
is\n");printf("%s\n",name); printf ("DA is %f and Total salary is
=%f\n",da, salary);
getch();
}
```

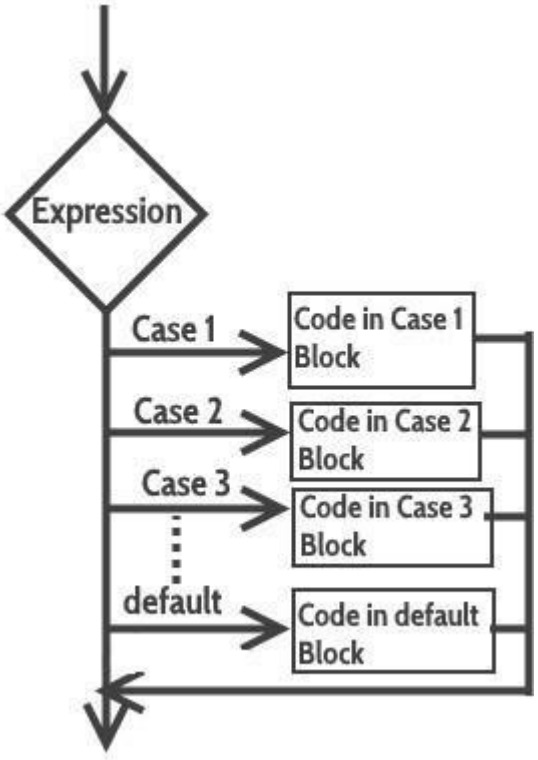
For case 1,da=10% of basic salary. For case 2, da=15% of basic salary. For case 3, da=20% of basic salary.
For default case >3 da is not given.(da=0)

```
o/p

Enter name of
employee: Kartiyani
Enter Basic salary
5000

Enter code of
employee 1

Employee name is
Kartiyani
DA is 500 and total salary is 5500
```



Rules for using `switch` statement

1. The expression (after switch keyword) must yield an **integer** value
2. The case **label** values must be unique.
3. The case label must end with a colon(**:**)

Difference between `switch` and `if`

- `if` statements can evaluate `float` conditions. `switch` statements cannot evaluate `float` conditions.
- `if` statement can evaluate relational operators. `switch` statement cannot evaluate relational operators i.e they are not allowed in `switch` statement.

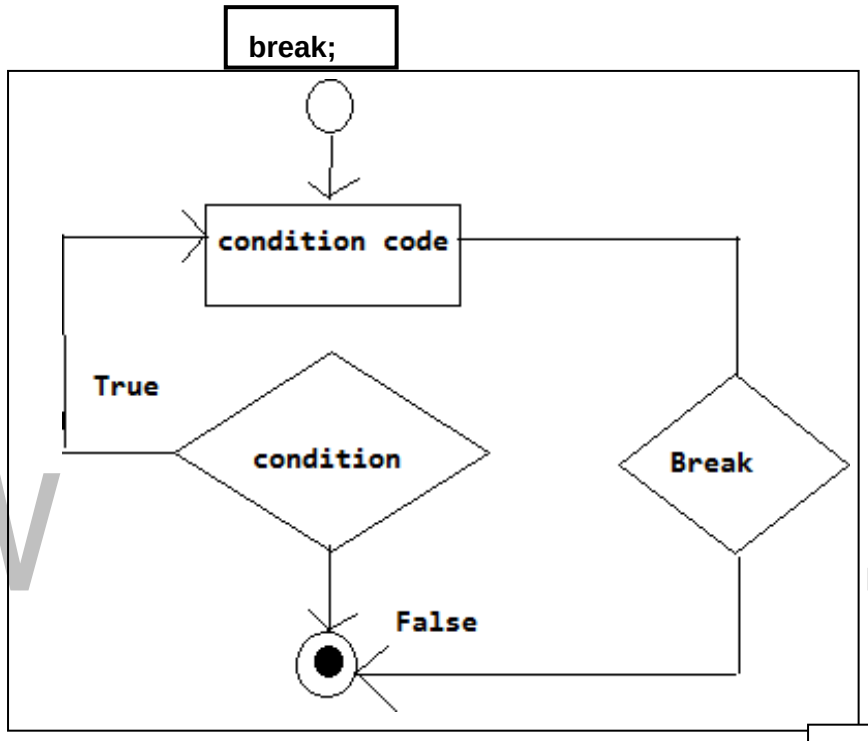
BREAK

The break statement in C programming has the following two usages −

- **When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.**
- **It can be used to terminate a case in the switch statement**

BREAK is a keyword that allows us to jump out of a loop instantly, without waiting to get back to the conditional test.

The syntax for a break statement in C is as follows −



Example program

```
#include <stdio.h>
int main ()
{
    int a = 10;
  while( a < 20 )
    {

       printf("value of a: %d\n", a);
       a++;
     if( a > 15)
      {
       break;
      }
   }
  return 0;
}
```

Output

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
```

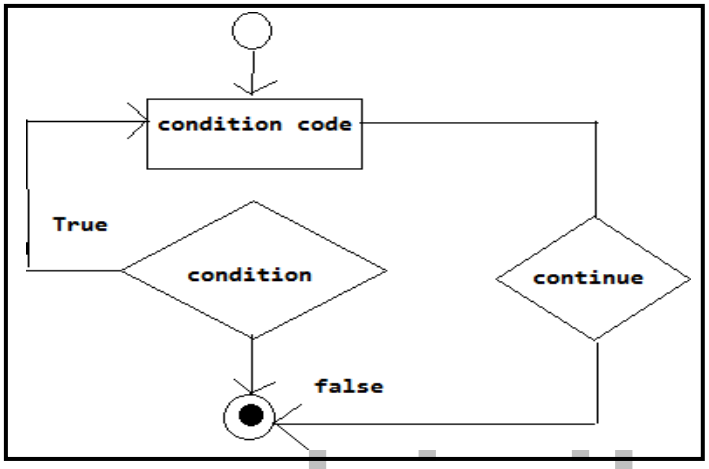
Continue

The continue statement in C programming works somewhat like the break statement. Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between.

For the for loop, continue statement causes the conditional test and increment portions of the loop to execute. For the while and do...while loops, continue statement causes the program control to pass to the conditional tests.

Syntax:

```
continue;
```



Example program

```
#include <stdio.h>
 int main ()
 {
      int a = 10;
    do {
         if( a == 15)
         {
           /* skip the iteration */
         a = a + 1;
         continue;
         }
       printf("value of a: %d\n", a);
       a++;
       } while( a < 20 );
   return 0;
}
```

Output

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Break	Continue
The `break` statement can be used in both `switch` and loop (`for`, `while`, `do while`) statements.	**The `continue` statement can appear only in loops. You will get an error if this appears in switch statement.**
A `break` causes the `switch` or loop statements to terminate the moment it is executed. Loop or `switch` ends abruptly when break is encountered.	**A `continue` doesn't terminate the loop, it causes the loop to go to the next iteration. The `continue` statement is used to skip statements in the loop that appear after the `continue`.**
The `break` statement can be used in both `switch` and loop statements.	**The `continue` statement can appear only in loops. You will get an error if this appears in switch statement.**
When a `break` statement is encountered, it terminates the block and gets the control out of the `switch` or loop.	**When a `continue` statement is encountered, it gets the control to the next iteration of the loop.**

GOTO

GOTO STATEMENT

'C' supports goto statement to branch unconditionally from one point to another in the program.

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

NOTE − Use of **goto** statement is highly discouraged in any programming language because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify. Any program that uses a goto can be rewritten to avoid them.

Syntax

The syntax for a **goto** statement in C is as follows −

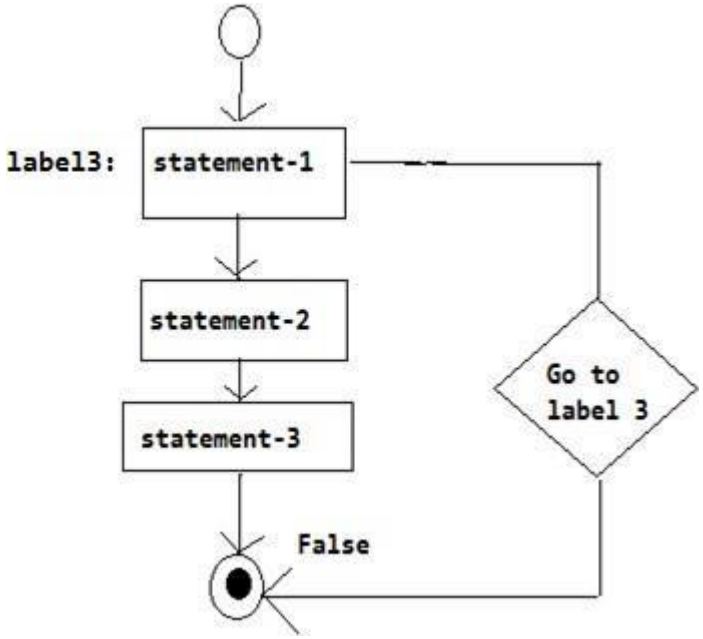
```
goto label;
..
.
label: statement;
```

Or

```
label: statement;
...

...

goto label;
```



Output

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Example program

```
#include <stdio.h>
 int main ()
{
   int a = 10;
     ABCL:do
   {
         if( a == 15)
         {
         a = a + 1;
         goto ABCL;
         }
        printf("value of a: %d\n", a);
        a++;
   }while( a < 20 );
    return 0;
}
```

program to print 'n' natural number

```
#include<stdio.h>
void main( )
{
     int n,i=1;
     clrscr();
     printf("enter number");
     scanf("%d\t",n);
     printf("natural numbers from 1 to %d", n);
     lb: printf("%d\t",i);
         i++;
     if(i<=n)
         goto lb;
     getch();
 }
```

LINEAR SEARCH

Searching an element within an array an array of n elements, where each Consider element is a key (e.g., a number). The articular key(number) in the array. task is to

The simplest method is a sequential search or linear search. The idea is to simply search the array, element by element, from the beginning until the key is found or the end of the list is reached. If found, the corresponding position in the array is printed; otherwise, a message will have to be displayed that the key(number) is not found. Now, the implementation of the program will be

Program:

```
#include
<stdio.h>
#include
<stdlib.h>      int
main()
{
int n,i,key, FOUND=0, a[30]; // array
declaration printf("\n How many
numbers:"); scanf("%d",&n); // array size
printf("\n Enter the array elements:
\n");for(i=0 ; i<n; i++)
{
scanf("%d", &a[i]);
}
printf("\n Enter the key to be
searched: ");scanf("%d",&key);

for(i=0 ; i<n; i++) // searching an element in an
arrayif(a[i] == key)
{
printf("\n Found at
%d",i);FOUND=1;
}
if(FOUND = = 0)
printf("\n NOT
FOUND...");return 0;
}
```

Output

```
How many numbers: 6
Enter the array elements:
21
33
46
52
27
Enter the key to be searched: 5
Found at 3
```


Decision Mak

Conditional Branching
→if statement
→nested if statement
→if ..else statement
→nested if else statement

These include conditional type branching

if statement

It takes the following form

```
if(test-expression)
```

It allows the computer to evaluate the ex the value of the expression is "true" or statement. This point of program has two the false condition.

```
if(test-expression)
{
statement-block
}
statement-x;
```

The statement-block may be a single s expression is true, the statement-block w will be skipped and the execution will jum both the statement-block and the stateme

Eg: equ

BINARY SEARCHING

In ***Binary searching*** the drawbacks of sequential search can be

eliminatedThe binary search halves the size of the list to search

in each Iteration

Logic : Binary search can be explained simply by the analogy of searching for a page in a book. Suppose a reader is searching for page 90 in a book of 150 pages. The reader would first open the book at random towards the latter half of the book. If the page number is less than 90, the reader would open at a page to the right; if it is greater than 90, the reader would open at a page to the left, repeating the process till page 90 was found.

Binary search requires sorted data (in ascending order) to operate on.

In binary search, the following procedure is implemented.

- Look at the middle element of the list.
- If it is the value being searched, then the job is done.
- If the value that is being searched is smaller than the middle element, then continuewith the bottom half of the list.
- If the value that is being searched is larger than the middle element, then continuewith the top half of the list.

Eg:-Depiction of binary search algorithm (the number to be searched is greater than midvalue)

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Data	23	27	29	32	34	41	46	47	49	52	55	68	71	74	77	78
1st iteration	L						M									H
2nd iteration								L				M				H
3rd iteration								L	M	H						

Algorithm*:* The algorithm determines the position of T in the LIST.

1. START
2. PRINT “ENTER THE NO. OF ELEMENTS IN THE ARRAY”
3. INPUT

N4. I=0

5. PRINT “ENTER ARRAY ELEMENT”
6. INPUT

LIST(I) 7.

I=I+1

8. IF I<N THEN GOTO STEP 5
9. PRINT “ENTER THE ELEMENT TO SEARCH”
10. INPUT T
11. HIGH = N - 1
12. LOW = 0

13. FOUND = 0
14. MID = (HIGH + LOW)/ 2
15. IF T = LIST [MID]
        FOUND = 1
    ELSE IF T < LIST[MID]
        HIGH = MID-1
    ELSE
    LOW = MID+1
16. IF (FOUND =0) and (HIGH > = LOW) THEN GOTO STEP 14

17. IF FOUND =0 THEN PRINT “NOT FOUND”
18. ELSE PRINT “FOUND AT”, MID.
19. STOP

The C program for this algorithm is as follows:

```
#inc
lude
<std
io.h
>
#inc
lude
<std
lib.h
> int
mai
n()
{
int a[30],n,i,t,low,mid,high,found=0;
printf(“\n Enter the number of
elements in the array:”);
scanf(“%d”,&n);
printf(“\n Enter the
elements of the array:”);
for(i=0 ; i< n; i++)
scanf(“%d”, &a[i]);
printf(“\n Enter the
element to search :”);
scanf(“%d”,&t);
low = 0; high = n - 1;
while(high >= low)
{
mid = (low + high) / 2;if(a[mid] == t)
{
found = 1;break;
}else if (t < a[mid])high = mid - 1;
 else
low = mid + 1;
}
if(found==0)
printf(“\n NOT FOUND”);else
printf(“\n FOUND AT %d”,mid);return 0;
}
```

Output

Enter the number of elements in the array: 9
Enter the number of elements in the array 9
Enter the elements of the array:
Enter the elements of the array:

1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
Enter the element to search: 7	Enter the element to search: 7
FOUND AT 6	NOT FOUND

LOOPING STATEMENTS

Loop constructs supports repeated execution of statements when a condition match.

If the loop Test Condition is true, then the loop is executed, the sequence of statements to be executed is kept inside the curly braces { } is known as the **Loop body**. After every execution of the loop body, **condition** is verified, and if it is found to be **true** the loop body is executed again. When the condition check returns **false**, the loop body is not executed, and execution breaks out of the loop.

Types of Loop

There are 3 types of Loop in C language, namely:

1. while loop
2. for loop
3. do while loop

while loop

Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

```
while(condition)
{
   statements;
}
```

Example-- Example: Program to print first 10 natural numbers

```
#include<stdio.h>
void main( )
{
  int x;
  x =
  1;
  while(x <= 10)
  {
    printf("%d\t", x);
     x++;
  }
}
```

Output: 1 2 3 4 5 6 7 8 9 10

do while loop

In some situations it is necessary to execute body of the loop before testing the condition. Such situations can be handled with the help of do-while loop. do statement executes the body of the loop first and at the end, the condition is checked using while statement. It means that the body of the loop will be executed at least once, even though the starting condition inside while is initialized to be false.

General syntax

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

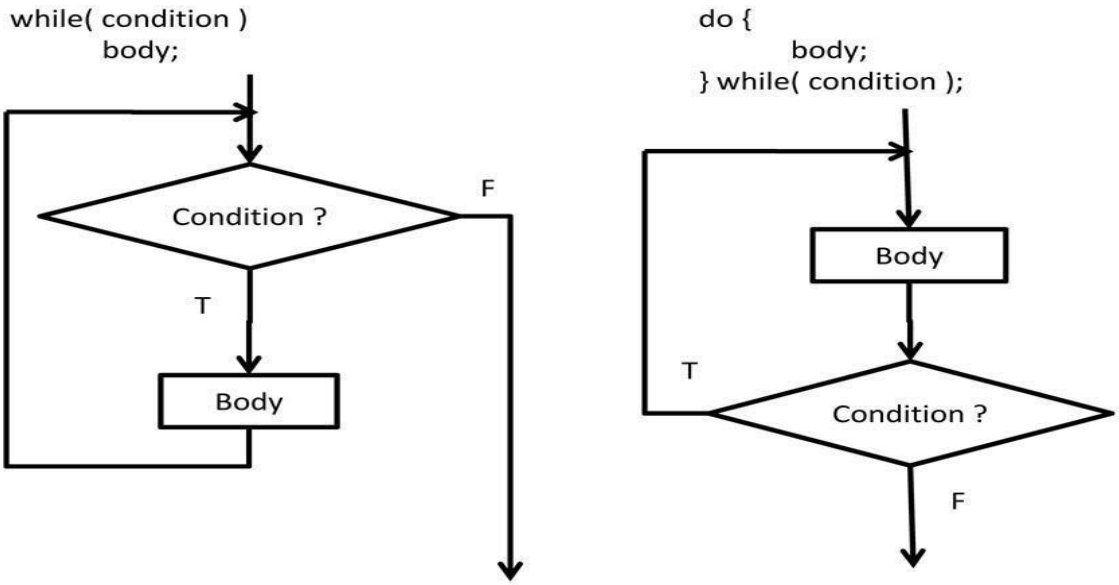
Example: Program to print first 10 multiples of 5.

```
#include<stdio.h>
void main()
{
   int a, i;
   a = 5;
   i = 1;
   do
   {
      printf("%d\t", a*i);
      i++;
   }
   while(i <= 12);
}
```


Output

5 10 15 20 25 30 35 40 45 50 55 60

While versus Do-While Loops



Jumping Out of Loops

Sometimes, while executing a loop, it becomes necessary to skip a part of the loop or to leave the loop

1) break statement

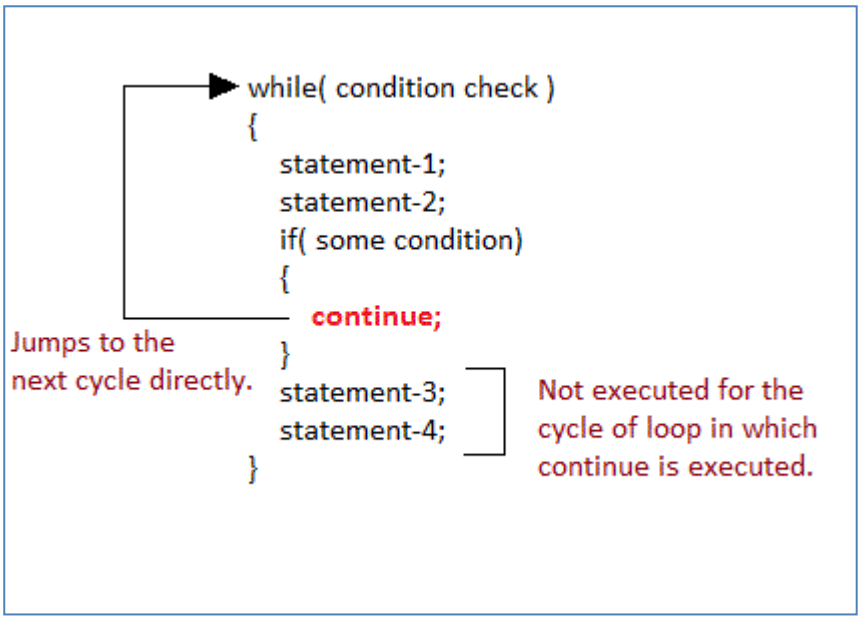
When break statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop.

```
while( condition check )
{
   statement-1;
   statement-2;
   if( some condition)
   {
      break;
   }
   statement-3;
   statement-4;
}
```

Jumps out of the loop, no matter how many cycles are left, loop is exited.

2) continue statement

It causes the control to go directly to the test-condition and then continue the loop process. On encountering continue, cursor leave the current cycle of loop, and starts with the next cycle.



For Loop

for loop is used to execute a set of statements repeatedly until a particular condition is satisfied. We can say it is an **open ended loop.**. General format is,


```
for(initialization; condition; increment/decrement)
{
    statement-block;
}
```

In for loop we have exactly two semicolons, one after initialization and second after the condition. In this loop we can have more than one initialization or increment/decrement, separated using comma operator. But it can have only one **condition**.

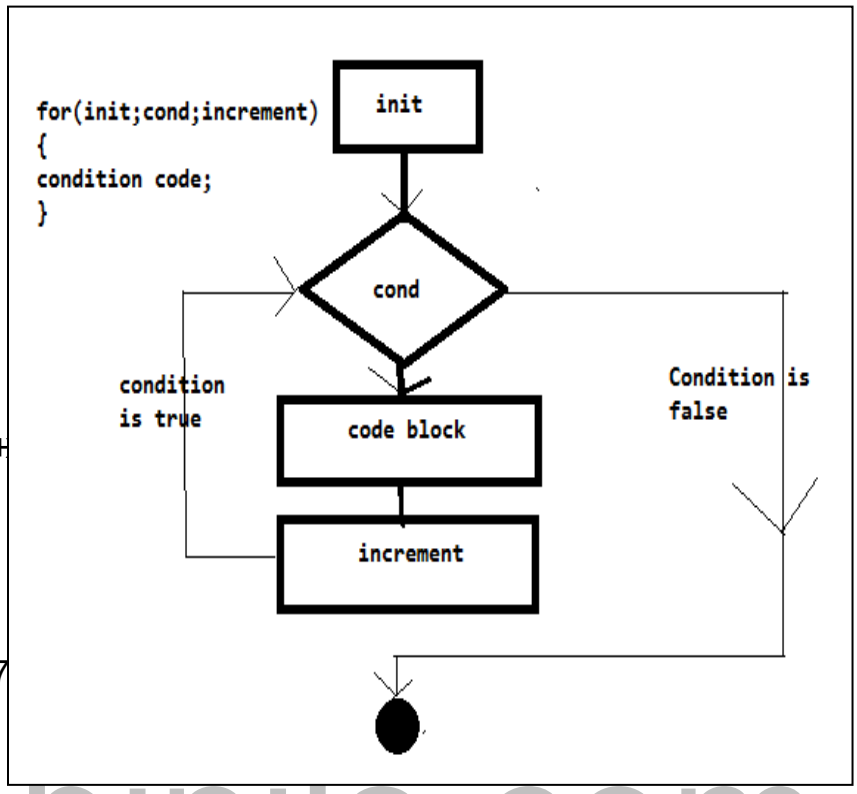
The for loop is executed as follows:

1. It first evaluates the initialization code.
2. Then it checks the condition expression.
3. If it is **true**, it executes the for-loop body.
4. Then it evaluate the increment/decrement condition and again follows from step 2.
5. When the condition expression becomes **false**, it exits the loop.

Example: Program to print first 10 natural numbers

```
#inclu
d
e<stdi
o
.h>voi
d
main(
)
{
   int x;
   for(x = 1; x <= 10;+
   {
      printf("%d\t", x);
   }
}
```

Output: 1 2 3 4 5 6 7



Nested for loop

We can also have nested for loops, i.e one for loop inside another for loop.Basic syntax is,

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


Example: Program to print half Pyramic of numbers

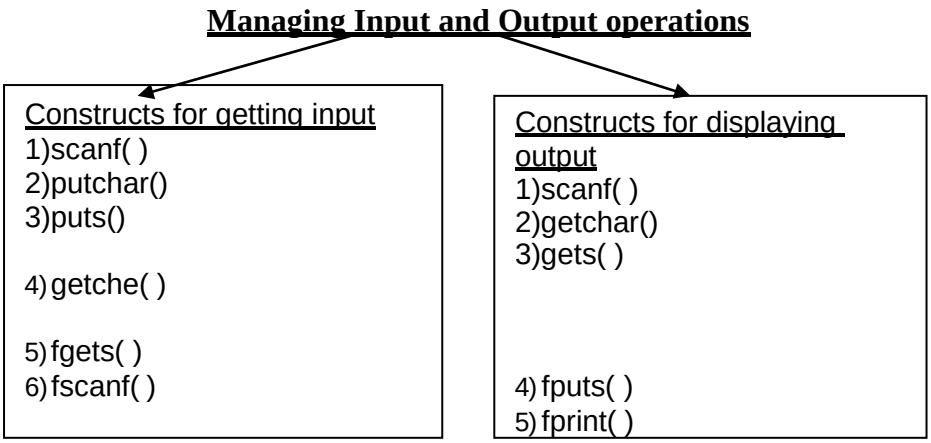
```
#include<stdio.h>
void main( )
{
   int i, j;
   for(i = 1; i < 5; i++) /* first for loop */
   {       printf("\n");
                  /* second for loop
```

Output

```
1
2 1
3 2 1
4 3 2 1
5 4 3 2 1
```

```
        inside the first */for(j = i; j > 0;
        j--)

{
            printf("%d", j);
        }
    }
}
```



C Input and Output

Input means to provide the program with some data to be used in the program

Output means to display data on screen or write the data to a printer or a file.

1. Singlecharacterinputandoutput[getchar() andputchar()]

- **input- getchar()**
- **output- putchar()**

The **int getchar(void)** function reads the next available character from the screen and returns it as an integer. This function reads only single character at a time.

The **int putchar(int c)** function puts the passed character on the screen and returns the same character. This function puts only single character at a time.

program

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

   int c;

   printf( "Enter a value :");
   c = getchar( );

   printf( "\nYou entered: ");
   putchar( c );

   return 0;
}
```

```
output
$./a.out
Enter a value : this is DS class
You entered: t
```

2. Stringinputandoutput[gets()andputs()

- **Input--- gets (str)**
- **Output---puts (str)**

The **gets( )** function reads a line from **stdin** into the buffer pointed to by **s** until either a terminating newline or EOF (End of File).

The **puts( )** function writes the string 's' and 'a' trailing newline to **stdout**.

Program

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

   char str[100];

   printf( "Enter a value :");
   gets( str );

   printf( "\nYou entered: ");
   puts( str );

   return 0;
}
```

Output

```
$./a.out
Enter a value : this is DS class
You entered: this is DS class
```


3.FormattedInput[scanf( )] and FormattedOutput[printf( )]

Specifier Meaning
%c – Print a character
%d – Print a Integer
%i – Print a Integer
%u-- Unsigned int
%ld-- Long int
%e – Print float value in exponential form.
%f – Print float value
%g – Print using %e or %f whichever is smaller
%lf --Double
%lf-- Long double
%o – Print octal value
%s – Print a string
%x – Print a hexadecimal integer (Unsigned) using lower case a – f
%X – Print a hexadecimal integer (Unsigned) using upper case A – F
%a – Print a unsigned integer.
%p – Print a pointer value
%hx – hex short

scanf()

scanf() is a predefined function in "stdio.h" header file. It can be used to read the input value from the keyword.

Syntax of `scanf()` function

1. **& ampersand** symbol is the address operator specifying the address of the variable
2. **control string** holds the format of the data
3. **variable1, variable2**, ... are the names of the variables that will hold the input value.

```
scanf("control string", &variable1, &variable2, ...);
```

```
int a;
float b;
scanf("%d%f",&a,&b);
```

Example

```
double d;
char c;
long int l;
scanf("%c%lf%ld",&c&d&l);
```

- **Printf**
- Printf is a predefined function in "stdio.h" header file, by using this function, we can print thedata or user defined message on console or monitor. While working with printf(), it can take any number of arguments but first argument must be within the double cotes (" ") and every argument should separated with comma (,) Within the double cotes, whatever we pass, it prints same, if any format specifies are there, then value is copied in that place.

Program

```
#include <stdio.h>      //This is needed to run printf() function.
int main()
{
    printf("C Programming");  //displays the content inside quotation
    return 0;
}
```

```
Output
C Programming
```

Program(integer and float)

```
#include <stdio.h>
#include <conio.h>
void main();
{
int a;
float b;
clrscr();
printf("Enter any two numbers: ");
scanf("%d %f",&a,&b);
printf("%d %f \n",a,b);
getch();
}
```

```
Output: Enter any two numbers:10 3.5
10
3.5
```

Program

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

    int integer  =  9876;
    float decimal = 987.6543;

    printf("4 digit integer right justified to 6 column: %6d\n", integer);

    printf("4 digit integer right justified to 3 column: %3d\n", integer);

    printf("Floating point number rounded to 2 digits: %.2f\n",decimal);

    printf("Floating point number rounded to 0 digits: %.f\n",987.6543);

    printf("Floating point number in exponential form: %e\n",987.6543);

    return 0;
}
```

FILE INPUT and OUTPUT

4. File string input and output using fgets( )and fputs()

The fgets() function

The fgets() function is used to read string(array of characters) from the file.

Syntax

```
fgets(char str[],int n,FILE *fp);
```

The fgets() function takes three arguments, first is the string read from the file, second is size of string(character array) and third is the file pointer from where the string will be read.

Example

```
File*fp;
Str[80];
fgets(str,80,fp)
```

Example program

```
#include<stdio.h>
    void main()
    {
        FILE *fp;
        char str[80];
        fp = fopen("file.txt","r"); // opens file in read mode (“r”)

        while((fgets(str,80,fp))!=NULL)
            printf("%s",str); //reads content from file
       fclose(fp);
    }
```

Data in file...

```
        C is a general-purpose programming language.
        It is developed by Dennis Ritchie.
```

```
C is a general-purpose programming language.
It is developed by Dennis Ritchie.
```

Output :

The fputs() function

The fputs() function is used to write string(array of characters) to the file.

→ The fputs() function takes two arguments, first is the string to be written to the file and second is the file pointer where the string will be written.

Syntax:

```
fputs(char str[], FILE *fp);
```

```
#
i
{
   FILE *fp;
   fp = fopen("proverb.txt", "w+"); //opening file in write mode
   fputs("Cleanliness is next to godliness.", fp);
   fputs("Better late than never.", fp);
   fputs("The pen is mightier than the sword.", fp);
   fclose(fp);
   return(0);
}
```

Output

```
Cleanliness is next to godliness.
 Better late than never.
The pen is mightier than the sword.
```

4. File string input and output using fgets( )and fputs( )

The fscanf() function

→The fscanf() function is used to read mixed type(characters, strings and integers) form the file.

→The fscanf() function is similar to scanf() function except the first argument which is a file pointer that specifies the file to be read.

Syntax:

```
fscanf(FILE *fp,"format-string",var-list);
```

Example program


```
#include<stdio.h>

        void main()
        {
                FILE *fp;
                char ch;

               int roll;
                char name[25];

                fp = fopen("file.txt","r");
               printf("\n Reading from file...\n");

                while((fscanf(fp,"%d%s",&rollno,&name))!=NULL)
                    printf("\n %d\t%s",rollno,name);//reading data
           fclose(fp);
        }
```

Output :

```
Reading from file...
6666     keith
7777     rose
```

The fprintf() function:

→ The fprintf() function is used to write mixed type(characters, strings and integers) in thefile.

→The fprintf() function is similar to printf() function except the first argument which is a filepointer specifies the filename to be written.

Syntax

```
fprintf(FILE *fp,"format-string",var-list);
```

Example program

```
#include<stdio.h>

        void main()
        {
FILE *fp;introll;
char name[25];
fp = fopen("file.txt","w");
scanf("%d",&roll);
scanf("%s",name);
fprintf(fp,"%d%s%",roll,name);
close(fp);
        }
```

Output

```
6666
john
```

SORTING

C PROGRAM FOR BUBBLE SORT

Bubble sort is a simple sorting algorithm in which each element is compared with adjacent element and swapped if their position is incorrect. It is named as bubble sort because same as like bubbles the lighter elements come up and heavier elements settle down.

It is less efficient as its average and worst case complexity is high, there are many other fast sorting algorithms like quick-sort, heap-sort, etc. Sorting simplifies problem-solving in computer programming.

Step by Step

– First Pass:

(5 1 4 2 8) (1 5 4 2 8), Here, algorithm compares the first two elements, and swaps them.
(1 5 4 2 8) (1 4 5 2 8), Swap since 5 > 4
(1 4 5 2 8) (1 4 2 5 8), Swap since 5 > 2
(1 4 2 5 8) (1 4 2 5 8), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

– **Second Pass:**

(1 4 2 5 8) (1 4 2 5 8)
(1 4 2 5 8) (1 2 4 5 8), Swap since 4 > 2
(1 2 4 5 8) (1 2 4 5 8)
(1 2 4 5 8) (1 2 4 5 8)

Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one whole pass without any swap to know it is sorted.

-Third Pass:

(1 2 4 5 8) (1 2 4 5 8)
(1 2 4 5 8) (1 2 4 5 8)
(1 2 4 5 8) (1 2 4 5 8)
(1 2 4 5 8) (1 2 4 5 8)

Finally, the array is sorted, and the algorithm can terminate.

Program

```
#include <stdio.h>
int main()
{
 int array[100], n, c, d, swap;
  printf("Enter number of elements\n");
  scanf("%d", &n);
   printf("Enter %d integers\n", n);
   for (c = 0; c < n; c++)
   scanf("%d", &array[c]);
   for (c = 0 ; c < n - 1; c++)
   {
    for (d = 0 ; d < n - c - 1; d++)
    {
      if (array[d] > array[d+1]) /* For decreasing order use < */
      {
        swap       = array[d];
        array[d] = array[d+1];
        array[d+1] = swap;
      }
    }
  }
   printf("Sorted list in ascending order:\n");
   for (c = 0; c < n; c++)
     printf("%d\n", array[c]);
   return 0;
}
```

Output

Enter number of elements
5
Enter 5 integers
5 1 4 2 8
Sorted list in ascending order:
1 2 4 5 8

INSERTION SORT

This is an in-place comparison-based sorting algorithm. Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted. An element which is to be 'insert'ed in this sorted sub-list, has to find its appropriate place and then it has to be inserted there. Hence the name, **insertion sort**.

The array is searched sequentially and unsorted items are moved and inserted into the sorted sub-list (in the same array). This algorithm is not suitable for large data sets as its average and worst case complexity are of $O(n^2)$, where **n** is the number of items.

How Insertion Sort Works?

We take an unsorted array for our example.

Insertion sort compares the first two elements.

It finds that both 14 and 33 are already in ascending order. For now, 14 is in sorted sub-list.

Insertion sort moves ahead and compares 33 with 27.

And finds that 33 is not in the correct position.

It swaps 33 with 27. It also checks with all the elements of sorted sub-list. Here we see that the sorted sub-list has only one element 14, and 27 is greater than 14. Hence, the sorted sub-list remains sorted after swapping.

By now we have 14 and 27 in the sorted sub-list. Next, it compares 33 with 10..... and so on.

Algorithm

Now we have a bigger picture of how this sorting technique works, so we can derive simple steps by which we can achieve insertion sort.

```
Step 1 - If it is the first element, it is already sorted. return 1;
Step 2 - Pick next element
Step 3 - Compare with all elements in the sorted sub-list
Step 4 - Shift all the elements in the sorted sub-list that is greater than
the
         value to be sorted
Step 5 - Insert the value
Step 6 - Repeat until list is sorted
```

Program

```
#include<stdio.h>
int main()
{
        int data[100],n,temp,i,j;
        printf("Enter number of terms(should be less than 100): ");
        scanf("%d",&n);
        printf("Enter elements: ");
        for(i=0;i<n;i++)
        {
                scanf("%d",&data[i]);
        }
        for(i=1;i<n;i++)
        {
                temp = data[i];
                j=i-1;
                while(temp<data[j] && j>=0)
/*To sort elements in descending order, change temp<data[j] to temp>data[j]
in above line.*/
                {
                        data[j+1] = data[j];
                        --j;
                }
                data[j+1]=temp;
        }
        printf("In ascending order: ");
        for(i=0; i<n; i++)
                printf("%d\t",data[i]);
    return 0;
}
```



```
Enter number of terms(should be less than
100):5 Enter elements: 33 12 4 26 77
In ascending order: 4 12 26 33 77
```

Ascending order

Step by step descriptive logic to sort array in ascending order.

1. Input size of array and elements in array. Store it in some variable say size and arr.
2. To select each element from array, run an outer loop from 0 to size - 1. The loop structure must look like for(i=0; i<size; i++).
3. Run another inner loop from i + 1 to size - 1 to place currently selected element at its correct position. The loop structure should look like for(j = i + 1; j<size; j++).
4. Inside inner loop to compare currently selected element with subsequent element and swap two array elements if not placed at its correct position.
   Which is if(arr[i] > arr[j]) then swap arr[i] with arr[j].

Program

```
#include <stdio.h>
#define MAX_SIZE 100    // Maximum array size
int main()
{
    int arr[MAX_SIZE];
    int size;
    int i, j, temp;

    printf("Enter size of array: ");
    scanf("%d", &size);

    /* Input elements in array */
    printf("Enter elements in array: ");
    for(i=0; i<size; i++)
    {
        scanf("%d", &arr[i]);
    }

    for(i=0; i<size; i++)
    {
                /* Place currently selected element array[[i]to its correct place*/
        for(j=i+1; j<size; j++)
        {
            /* Swap if currently selected array element is not at its correct position.
             */
            if(arr[i] > arr[j])
            {
                temp     = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
    /* Print the sorted array */
    printf("\nElements of array in ascending order: ");
    for(i=0; i<size; i++)
    {
        printf("%d\t", arr[i]);
    }

    return 0;
}
```

OUTPUT

```
Enter size of array: 10
Enter elements in array: 20 2 10 6 52 31 0 45 79 40
Elements of array in ascending order:
0       2       6       10      20      31      40      45      52      79
```

```
/*C program to sort an one dimensional array in descending order.*/
#inclu
de
<stdio
.h>
#defin
e MAX
100
int
main()
{
    int
    arr[MAX],
    n,i,j;int
    temp;

    printf("Enter total number of
    elements: ");scanf("%d",&n);

    //read array elements
    printf("Enter array
    elements:\n");
    for(i=0;i< n;i++)
    {
        printf("Enter element
        %d: ",i+1);
        scanf("%d",&arr[i]);
    }

    //sort
    array
    for(i=0;i
    < n;i++)
    {
        for(j=i+1;j< n;j++)
        {
            if(arr[i]< arr[j])
            {
                temp
                =arr
                [i];
                arr[
                i]
                =arr
                [j];
                arr[
                j]
                =tem
                p;
            }
        }
    }

    printf("\nArray elements after
    sorting:\n");for(i=0;i< n;i++)
```

```
    {
        printf("%d\n",arr[i]);
    }
    return 0;
}
```

STRINGS:

STRING ARRAY | STRING POINTER

Strings in C are represented by arrays of characters. The end of the string is marked with a special character, the *null character*

- **string.h :** collection of functions for string manipulation.

DECLARATION OF STRINGS

- Strings are declared in a similar manner as arrays

```
char s[5];
```

- Strings can also be declared using pointer

```
char *p;
```

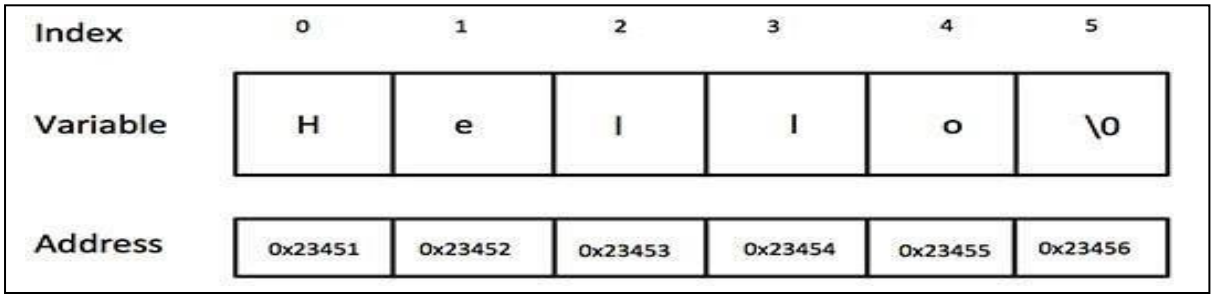
Syntax:

```
datatype string name[size];
```

```
char str[30];
char text[80];
```

STRING INITIALIZATION

char var[ ]="hello";



'\0' (NULL) charater would automatically be inserted at the end of string.

```
char arr[4]={'s','h','b','r','\0'}

char arr[]={'hello', 'good' ,'day',
 'please'} char Str = "abcdefg"
char greeting[] = "welcome";

char greeting[6] = {'H', 'e', 'l', 'l',
'o'}; char *c = "abcd";
char str="100"
 char str="3.4"
 Char
 str="111000"
```

STRING INPUT /OUTPUT

- **printf("%s",str), scanf("%s",str)**
- **gets(str),puts(str)**
- **fgets with stdin and fputs with stdout**
- **fgets( ) and fputs( )…for files**

```
#include
<stdio.h>
#include
<string.h>      int
main()
{
 char nickname[20];
 scanf("%s",
 nickname);
 printf("%s",nicknam
```

Eg **char s[1000] ;**
fgets(s,1000,stdi
n);

```
#include
<stdio.h>
#include
<string.h>      int
main()
{
     char
   nickname[20];
   gets(nickname);
   puts(nickname);
   return 0;
}
```

```
#include
<stdio.h>
#include
<string.h>      int
main()
{
 char nickname[20];
fgets(nickname,20,stdin
);
fputs(nickname,stdout);
   return 0;
}
```

Example program

```
#include
<stdio.h>       int
main()
 {
char name[10];
 printf("Who are you?
 ");
 fgets(name,10,stdin)
 ;
printf("Glad to meet you, %s
\n",name); return(0);
}
```

String input and output using fscanf() and fprintf()

C program has three I/O streams.

- stdin,
- stdout, and
- stderr

--The input stream is called standard-input (stdin); the usual output stream is called standard-output (stdout); and the side stream of output characters for errors is called standard error (stderr).

--Internally they represent file descriptors 0, 1, and 2 respectively.
--Calls to fprinf() and fscanf() differ significantly from calls to printf() and scanf().
--fprintf() sends formatted output to a stream and fscanf() scans and formats input from a stream.

Example program

```
#include <stdio.h>
int main()
{
int fi rst, second;
fprintf(stdout,“Enter two ints in this line: ”);
fscanf(stdin,“%d %d”, &fi rst, &second);
fprintf(stdout,“Their sum is: %d.\n”, fi rst + second);
return 0;
}
```

Character Manipulation

Table: Character functions in <ctype.h> where c is the character argument

ialnum(c) Returns a non-zero if c is alphabetic or numeric
isalpha(c) Returns a non-zero if c is alphabetic
scntrl(c) Returns a non-zero if c is a control character
isdigit(c) Returns a non-zero if c is a digit, 0 – 9
isgraph(c) Returns a non-zero if c is a non-blank but printing character
islower(c) Returns a non-zero if c is a lowercase alphabetic character, i.e., a – z
isprint(c) Returns a non-zero if c is printable, non-blanks and white space included
ispunct(c) Returns a non-zero if c is a printable character, but not alpha, numeric, or blank
isspace(c) Returns a non-zero for blanks and these escape sequences: '\f', '\n', '\r', '\t', and '\v'
isupper(c) Returns a non-zero if c is a capital letter, i.e., A – Z
isxdigit(c) Returns a non-zero if c is a hexadecimal character: 0 –9, a – f, or A – F
tolower(c) Returns the lowercase version if c is a capital letter; otherwise returns c
toupper(c) Returns the capital letter version if c is a lowercase character; otherwise returns c

This program counts the number of words in a string

```
#include  <stdio.h>
#include <ctype.h>
int main()
{
char s[30];
int i=0,count=0;
printf("\n enter the string\n");
scanf("%[^\n]",s);
while(s[i]!='\0')
{
while(isspace(s[i]))
i++;
if(s[i]!='\0')
{
++count;
while(!isspace(s[i]) && s[i] != '\0')
i++;
}
}
printf("\n NO. of words in the string is %d:", count);
return 0;
}
```

Output:

```
enter the string
how are you
NO. of words in the string is
3
```

converts a given text into a capital letter using toupper() function

```
#include <stdio.h>
#include <sting.h>
int main()
{
char a[30];
int i=0;
printf("\n enter the string\n");
gs(a);
while(a[i]!='\0')
{
a[i]=toupper(a[i]);
i++;
}
a[i]='\0';
puts(a);
return 0;
}
```

Output:

```
enter the string
how
HOW
```

Disadvantage : C has the weakest character string capability. Strictly speaking, there are no character strings in C, just arrays of single characters.

What character manupilation cannot do

- Assign one to the other: s1 = s2;
- Compare them for collating sequence: s1 < s2
- Concatenate them to form a single longer string: s1 + s2
- Return a string as the result of a function

String Manipulation

A set of standard C library functions that are contained in <string.h> provides the following.

Function	Description
strcpy(s1,s2)	Copies s2 into s1
strncpy(s1,s2,n)	**It copies first n characters of str2 into str1.**
strcat(s1,s2)	Concatenates s2 to s1. That is, it appends the string contained by s2 to the end of the string pointed to by s1. The terminating null character
strncat(s1,s2,n)	**First n characters of str2 is concatenated at the end of str1**
strlen(s1)	Returns the length of s1. That is, it returns the number of characters in the string without the terminating null character.
strcmp(s1,s2)	Returns 0 if s1 and s2 are the same Returns less than 0 if s1<s2 Returns greater than 0 if s1>s2
strncmp(s1,s2,n)	Returns 0 if s1 and s2 are the same for first n characters
strcmpi( )	Same as strcmp() function. But, this function negotiates case. "A" and "a" are treated as same.
strchr(s1,ch)	Returns pointer to first occurrence ch in s1
strrchr(s1,ch)	Returns pointer tolast occurrence ch in s1
strstr(s1,s2)	Returns pointer to first occurrence s2 in s1
strlen( )	function in C gives the length of the given string
strdup( )	function in C duplicates the given string
strlwr( )	function converts a given string into lowercase
strupr( )	function converts a given string into uppercase
strrev( )	function reverses a given string in C language

strcat (str1, str2) #include <stdio.h> #include <string.h> int main( )

```
{
  char source[ ] = "
  APPLE" ; char target[ ]=
  " LIME" ;
  printf ( "\nSource string = %s",
  source ) ; printf ( "\nTarget string =
  %s", target ) ; strcat ( target, source )
  ;
  printf ( "\nTarget string after strcat( ) = %s", target
```

Source string = APPLE
Target string = LIME
Target string after strcat( ) = LIME APPLE

strncat (str1, str2, n) #include <stdio.h> #include <string.h> int main( )

```
{ char source[ ] = "
  APPLEJIUCE" ; char target[
  ]= "LIME" ;
   printf ( "\nSource string = %s", source ) ;
  printf ( "\nTarget string = %s", target ) ;
  strncat ( target, source, 4 ) ;
  printf ( "\nTarget string after strncat( ) = %s",
  target )
.
```

Source string = APPLEJUICE Target string = LIME
Target string after strcat( ) = LIME APPL

strcpy(str2,str1)

```
#include
<stdio.h>
#include
<string.h>     int
main( )
{
  char source[ ] = "fresh2refresh"
  ; char target[20]= "" ;
  printf ( "\n source string = %s", source
  ) ; printf ( "\n target string = %s", target
  ) ; strcpy ( target, source ) ;
    printf ( "\n target string after strcpy( ) = %s", target )
  ; return 0;
}
```

source string = one
target string
target string after strcpy( ) = one

strcpy(str2,str1, n) #include <stdio.h> #include <string.h> int main( )

```
{
  char source[ ] =
  "mindblowing" ; char
  target[20]= "" ;
  printf ( "\n source string = %s", source ) ;
 printf ( "\n target string = %s", target ) ;
 strncpy ( target, source, 5 ) ;
 printf ( "\ntarget string after strcpy( ) = %s",
 target ) ;
```

source string = mindblowing
target string =
target string after strncpy( ) = mindb

strlen( )

```
#include
<stdio.h>
#include
<string.h>        int
main( )
{

int len;
char str[20]="APPLE" ;
 len = strlen(str) ;
printf ( "\string length = %d \n" , len
) ; return 0;
      }
```

string length = 6

strcmp(str1,str2) 0

```
#include
<stdio.h>
#include
<string.h> int
main()
 {

char str1[] = "abcd", str2[] = "abCd", str3[] =
"abcd"; int result;
 result = strcmp(str1, str2);
printf("strcmp(str1, str2) = %d\n",
result); result = strcmp(str1, str3);
printf("strcmp(str1, str3) = %d\n",
result); return 0;
 }
```

strcmp(str1, str2) = 32
strcmp(str1, str3) = 0 (same)

strchr(str,ch)

```
#include
<stdio.h>
#include
<string.h>        int
main ()
{

char string[55] ="This is a string for
testing"; char *p;
p = strchr (string,'i');
printf ("First occurrence of character "i" in
%s " is" %s, string, p);
return 0;
}
```

first occurrence of character "i" in This is a string for testing" is 3

strrchr(str,ch)

```
#include
<stdio.h>
#include
<string.h>
int main ()
{
 char string[55] ="This is a string for testing";
 char *p;
 p = strrchr (string,'i');
printf ("Last occurrence of character "i" in
"%s is", string, p);
return 0;
}
```

Last occurrence of character "i" in This is a string for testing" is 26

strlwr( )

```
#include<stdio.h
>
#include<string.h
>
int main()
{
char str[ ] = "MODIFY This String
To LOwer";
   printf("%s\n",strlwr (str));
   return 0;
}
```

Output:
modify this string to lower

strupr( )

```
#include<stdio.h
>
#include<string.
h> int main()
{
   char str[ ] = "Modify This String To Upper";
   printf("%s\n",strupr(str));
   return 0;
}
```

Output:
MODIFY THIS STRING TO UPPER

strrev( )

```
#include<stdio.h
>
#include<string.h
> int main()
{
  char name[30] = "Hello";
   printf("String before strrev( ): %s\n",name);
   printf("String after strrev( ) :
   %s",strrev(name)); return 0;
}
```

OUTPUT
String before strrev( ) :
Hello String after strrev( ):
olleH

strset( )

```
#include<stdio.h
>
#include<string.
h> int main()
{
  char str[20] = "Test String";
  printf("Original string is : %s", str);
  printf("Test string after strset() :
  %s",strset(str,'#'));
  printf("After string set: %s",str);
  return 0;
}
```

Original string is : Test String
Test string after strset() :
############

Conversion functions

Typecast Function	Description
atoi( )	Converts string to integer
atof( )	Converts string to float
atol( )	Converts string to long
itoa( )	Converts integer to string
ltoa( )	Converts long to string

atoi function

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

int main()
{
   char a[10] = "100";
   int value = atoi(a);
  printf("Value = %d\n",
   value); return 0;
}
```

Output:

atof function

```
#include
<stdio.h>
#include
<stdlib.h> int
main()
{
   char a[10] = "3.14";
   float pi = atof(a);
printf("Value of pi = %f\n",
pi);
   return 0;
}
```

Output:

itoa( )-Converts integer to string

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
 int main()
{
   int a=54325;
   char buffer[20];
itoa(a,buffer,2);
printf("Binary value = %s\n",
 buffer); itoa(a,buffer,10);
 printf("Decimal value = %s\n",
 buffer); itoa(a,buffer,16);
 printf("Hexadecimal value = %s\n",
   buffer); return 0;
}
```

Output:

Binary value = 1101010000110101
Decimal value = 54325
Hexadecimal value = D435

SCANSET

- **This conversion facility allows the programmer to specify the set of characters that are (or are not) acceptable as part of the string.**
- A scanset conversion consists of a list of acceptable characters enclosed within square brackets.

Program-1

```
#include<stdio.h>
int main()
{
char str[50];
printf("Enter a string in lower case:");
scanf("%[a-z]",str);
printf("The string was : %s\n",str);
return 0;
}
```

Output

(a) Enter a string in lower case: hello world
The string was: hello world

(b) Enter a string in lower case: hello, world
The string was: hello
[In the second case, the character, ',' (comma) is not in the specified range.]

(c) Enter a string in lower case: abcd1234
The string was : abcd
[In the third case, the digit 1234 is not in the specified range.]

Program-2

```
#include<stdio.h>
int main()
{
char str[50];
printf("Enter a string in lower case:");
scanf("%[^a-z]",str); printf("The
string was : %s\n",str);return 0;
}
```

Output

Enter a string in lower case: abcd1234
The string was : 1234

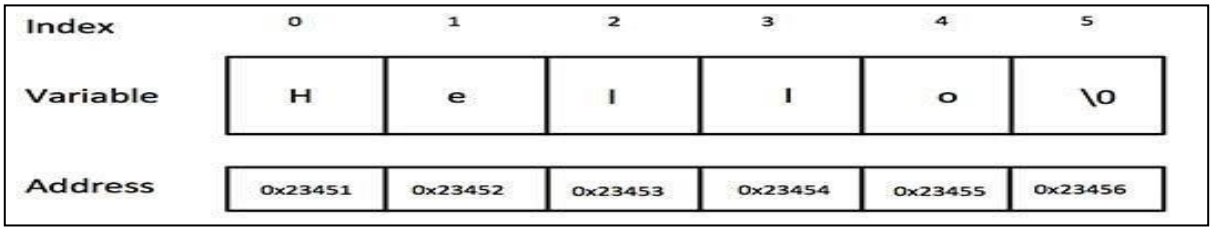
STRING ARRAY [ONE DIMENTIONAL]

CHAR/STRING ARRAY DECLARATION

String array are one-dimensional array of characters terminated by a null character '\0'.

- **Character Array**

  char arr[]={‘s’,'h’,'b',’r'}
  char arr[]={‘hello’, ‘good’ ,‘day’, ‘please’}
  char Str = “abcdefg”
  char greeting[] = "Hello";
  char greeting[6] = {'H', 'e', 'l', 'l', 'o'};



String Array

- **String Array = {“abc”, ”def”,**

“ghi”} **STRING ARRAY [TWO**

DIMENTIONAL]

Declaration of a two-dimensional array of strings.

A two-dimensional array of strings can be declared as follows:

```
<data_type> <string_array_name>[<row_size>][<columns_size>];

char s[5][30];
```

Initialization

Two-dimensional string arrays can be initialized as shown

```
char s[5][10] ={“Cow”,”Goat”,”Ram”,”Dog”,”Cat”};
```

which is equivalent to

s[0]	C	o	w	\0					
S[1]	G	o	a	t	\0				
S[2]	R	a	m	\0					
S[3]	D	o	g	\0					
S[4]	C	a	t	\0					

Example program : Search a character in a string

```
#include<stdio.h>
int main() {
   char str[20], ch;
  int count = 0, i;
     printf("\nEnter a string : ");
 scanf("%s", &str);

   printf("\nEnter the character to be searched : ");
   scanf("%c", &ch);

  for (i = 0; str[i] != '\0'; i++) {
     if (str[i] == ch)
        count++;
   }
    if (count == 0)
     printf("\n Character '%c'is not present", ch);
   else
    printf("\n Character '%c'is present", ch);
   return 0;
}
```

```
Enter a string: apple lime juice

Enter the character to be searched : i

Character i is  present
```

binary search for strings

```
#include <stdio.h>
#include <string.h>
void main()
{
 int i,n,low,high,mid;
 char a[50][50],key[20];
 printf("enter the number of names to be added\n");
 scanf("%d",&n);
 printf("enter the name in ascending order\n");
 for(i=0;i<=n-1;i++)
 {
 scanf("%s",&a[i]);
 }
 printf("\n");
 printf("enter the name to be searched\n");
 scanf("%s",&key);
 low=0;
 high=n-1;
 while(low<=high)
 {
 mid=(low+high)/2;
 if (strcmp(key,a[mid])==0)
 {
 printf("key found at the position %d\n",mid+1);
       exit(0);
 }
 else if(strcmp(key,a[mid])>0)
 {
 high=high;
 low=mid+1;
 }
 else
 {
 low=low;
 high=mid-1;
 }
 }
 printf("name not found\n");
}
```

```
enter the number of names to be added
4
enter the name in ascending order
mango
jackfruit
apple
grapes
enter the name to be searched
oranges
name not found
```

Program to Sort String Characters in string

```
#include <stdio.h>
#include <string.h>
int main (void) {
   char string[] = "simplyeasylearning";
   char temp;

   int i, j;
   int n = strlen(string);

   printf("String before sorting - %s \n", string);

   for (i = 0; i < n-1; i++) {
      for (j = i+1; j < n; j++) {
         if (string[i] > string[j]) {
            temp = string[i];
            string[i] = string[j];
            string[j] = temp;
         }
      }
   }
      printf("String after sorting - %s \n", string);
   return 0;
}
```

OUTPUT

```
String before sorting - simplyeasylearning
String after sorting - aaeegiillmnnprssyy
```

program to sort the names of students.

```
#include <stdio.h>
#include <string.h>
int main()
{
char names[5][10], temp[10];
int i, n, j;
clrscr();
printf("\n Enter the number of students : ");
scanf("%d", &n);
for(i=0;i<n;i++)
{
printf("\n Enter the name of student %d : ", i+1);
scanf("%s",&names[i]);
}
for(i=0;i<n;i++)
{
for(j=0;j<n-i-1;j++)
{
if(strcmp(names[j], names[j+1])>0)
{
strcpy(temp, names[j]);
strcpy(names[j], names[j+1]);
strcpy(names[j+1], temp);
}
}
}
printf("\n Names of the students in alphabetical order are : ");
for(i=0;i<n;i++)
 { printf("%s \n",names[i]);
}
return 0;
}
```

Output

```
Enter the number of students : 3
Enter the name of student 1 : Goransh
Enter the name of student 2 : Aditya
Enter the name of student 3 : Sarthak
Names of the students in alphabetical order are :
 Aditya
 Goransh
 Sarthak
```

Length of String Without Using strcat( )

```
#include <stdio.h>
int main()
{
    char s[1000], i;

    printf("Enter a string: ");
    scanf("%s", s);

    for(i = 0; s[i] != '\0'; ++i);

    printf("Length of string: %d", i);
    return 0;
}
```

Output

```
Enter a string: apple
Length of string: 5
```

copy Two Strings Without Using strcpy( )

Output

```
#include <stdio.h>
int main()
{
    char s1[100], s2[100], i;
    printf("Enter string s1: ");
    scanf("%s",s1);
    for(i = 0; s1[i] != '\0'; ++i)
    {
        s2[i] = s1[i];
    }
    s2[i] = '\0';
    printf("String s2: %s", s2);
    return 0;
}
```

```
Enter String s1: apple
String s2: apple
```

program to convert the lower case characters of a string into upper case without using string functions

```
#include <stdio.h>
int main()
{
char str[100], upper_str[100];
int i=0;
clrscr();
printf("\n Enter the string :");
gets(str);
while(str[i] != '\0')
{
if(str[i]>='a' && str[i]<='z')
upper_str[i] = str[i] – 32;
else
upper_str[i] = str[i];
i++;
}
upper_str[i] = '\0';
printf("\n The string converted into upper case is : ");
puts(upper_str);
return 0;
}
```

Output

```
Enter the string : Hello
The string converted into upper case is : HELLO
```

program to compare two strings without using string function

```
#include <stdio.h>
#include <string.h>
int main()
{
char str1[50], str2[50];
int i=0, len1=0, len2=0, same=0;
clrscr();
printf("\n Enter the first string : ");
gets(str1);
printf("\n Enter the second string : ");
gets(str2);
len1 = strlen(str1);
len2 = strlen(str2);
if(len1 == len2)
{
while(i<len1)
{
if(str1[i] == str2[i])
i++;
else break;
}
if(i==len1)
{
same=1;
printf("\n The two strings are equal");
}
}
if(len1!=len2)
printf("\n The two strings are not equal");
if(same == 0)
{
if(str1[i]>str2[i])
printf("\n String 1 is greater than string 2");
else if(str1[i]<str2[i])
printf("\n String 2 is greater than string 1");
}
return 0;
}
```

Write a program to reverse a given string without using string function

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

```
#include <conio.h>
#include <string.h>
int main()
{
char str[100], reverse_str[100], temp;
int i=0, j=0;
clrscr();
printf("\n Enter the string : ");
gets(str);
j = strlen(str)–1;
while(i < j)
{
temp = str[j];
str[j] = str[i];
str[i] = temp;
i++;
j—;
}
printf("\n The reversed string is : ");
puts(str);
getch();
return 0;
}
```

Output
Enter the string: Hi there
The reversed string is: ereht iH

C program to change case from upper to lower and lower to upper without using string function

```
#include <stdio.h>
 int main ()
{
   int i = 0;
   char ch, s[1000];

   printf("Input a string\n");
   gets(s);

   while (s[i] != '\0') {
    ch = s[i];
    if (ch >= 'A' && ch <= 'Z') // convrt to lower case
       s[i] = s[i] + 32;
     else if (ch >= 'a' && ch <= 'z') //convert to upper case
       s[i] = s[i] - 32;
     i++;
   }
   Printf(“\n the string is:”)
   printf("%s\n", s);
   return 0;
}
```

o/p
Input a string
file ABC
the string is
FILEabc

C Program to Count Number of Words in a given Text or Sentence

```
#include <stdio.h>
#include <string.h>
void main()
{
char s[200];
int count = 0, i;
printf("enter the string\n");
scanf("%[^\n]s", s);
for (i = 0;s[i] != '\0';i++)
{ if (s[i] == ' ')
count++; }
printf("number of words in given string are: %d\n", count + 1);
return 0;
}
```

o/p
enter the string
hello how are you friends
number of words in given string are: 5

Palindrome program in C language using built in functions

```
#include <stdio.h>
#include <string.h>
int main()
{
  char a[100], b[100];
   printf("Enter a string to check if it is a palindrome\n");
  gets(a);
   strcpy(b,a);
  strrev(b);
   if (strcmp(a,b) == 0)
     printf("Entered string is a palindrome.\n");
  else
     printf("Entered string isn't a palindrome.\n");
   return 0;
}
```

o/p
Enter a string to check if it is a palindrome
wow
Entered string is a palindrome

Palindrome program in C language without using built in functions

```
#include <stdio.h>
#include <string.h>
int main(){
   char string1[20];
   int i, length;
   int flag = 0;
      printf("Enter a string:");
   scanf("%s", string1);
      length = strlen(string1);
      for(i=0;i < length ;i++){
      if(string1[i] != string1[length-i-1]){
         flag = 1;
         break;
  } }
      if (flag) {
      printf("%s is not a palindrome", string1);
   }
   else {
      printf("%s is a palindrome", string1);
   }
   return 0;
}
```

o/p
Enter a string:wow
wow is not a palindrome

C program to find the frequency of characters in a string

```
#include <stdio.h>
#include <string.h>
int main()
{
  char string[100];
  int c = 0, count[26] = {0}, x;
  printf("Enter  a  string\n");
  gets(string);
   while (string[c] != '\0') {
  /** Considering characters from 'a' to 'z'
only and ignoring others. */
      if (string[c] >= 'a' && string[c] <= 'z') {
        x = string[c] - 'a';
        count[x]++;
     }
      c++;
  }
   for (c = 0; c < 26; c++)
      printf("%c occurs %d times in the string.\n", c + 'a', count[c]);
   return 0; }
```

Enter a
string maple
tree
a occurs 1 times in the string e occurs 3 times in the string l occurs 1 times in the string m occurs 1 times in the string p occurs 1 times in the string r occurs 1 times in the string

C program to swap two strings

```
#include <stdio.h>
#include <string.h>
int main()
{
  char first[100], second[100], temp[100];

  printf("Enter first string\n");
  gets(first);

  printf("Enter second string\n");
  gets(second);

  printf("\nBefore Swapping\n");
  printf("First string: %s\n", first);
  printf("Second string: %s\n\n", second);

  strcpy(temp, first);
  strcpy(first, second);
  strcpy(second, temp);

  printf("After Swapping\n");
  printf("First string: %s\n", first);
  printf("Second string: %s\n", second);

  return 0;
}
```

Write a program to extract a substring from the middle of a given string.

```
#include  <stdio.h>
#include <conio.h>
int main()
{
char str[100], substr[100];
int i=0,  j=0, n, m;
clrscr();
printf("\n Enter the main string : ");
gets(str);
printf("\n Enter the position from which to start the substring: ");
scanf("%d", &m);
printf("\n Enter the length of the substring: ");
scanf("%d", &n);
i=m;
while(str[i] != '\0' && n>0)

substr[j] = str[i];
i++;
j++;
n—;
}
substr[j] = '\0';
printf("\n The substring is : ");
puts(substr);
getch();
return 0;
}
```

Output

```
Enter the main string : Hi there
Enter the position from which to start the substring: 1
Enter the length of the substring: 4
The substring is : i th
```

Write a program to insert a string in the main text.

```
#include <stdio.h>
int main()
{
char text[100], str[20], ins_text[100];
int i=0, j=0, k=0,pos;
clrscr();
printf("\n Enter the main text : ");
gets(text);
printf("\n Enter the string to be inserted : ");
gets(str);
printf("\n Enter the position at which the string has to be inserted: ");
scanf("%d", &pos);
while(text[i]! = '\0')
{
if(i==pos)
{
while(str[k] != '\0')
{
ins_text[j] = str[k];
j++;
k++;
}
}
else
{
ins_text[j] = text[i];
j++;
}
i++;
}
ins_text[j] = '\0';
printf("\n The new string is : ");
puts(ins_text);
getch();
return0;
}
```

Output

```
Enter the main text : newsman
Enter the string to be inserted : paper
Enter the position at which the string has to be inserted: 4
The new string is: newspaperman
```


Write a program to delete a substring from a text.

```
#include <stdio.h>
int main()
{
char text[200], str[20], new_text[200];
int i=0, j=0, found=0, k, n=0, copy_loop=0;
clrscr();
printf("\n Enter the main text : ");
gets(text);
printf("\n Enter the string to be deleted : ");
gets(str);
while(text[i]!='\O')
{
j=O, found=0, k=i;
while(text[k]==str[j] && str[j]!='\0')
{
k++;
j++;
}
if(str[j]=='\O')
copy_loop=k;
new_text[n] = text[copy_loop];
i++;
copy_loop++;
n++;
}
new_str[n]='\O';
printf("\n The new string is : ");
puts(new_str);
return O;
```

Output

```
Enter the main text : Hello, how are you?
Enter the string to be deleted : , how are you?
The new string is : Hello
```

```
}
```

Write a program to replace a pattern with another pattern in the text.

```
#include <stdio.h>
#include <conio.h>
main()
{
char str[200], pat[20], new_str[200], rep_pat[100];
int i=0, j=0, k, n=0, copy_loop=0, rep_index=0;
clrscr();
printf("\n Enter the string : ");
gets(str);
printf("\n Enter the pattern to be replaced: ");
gets(pat);
printf("\n Enter the replacing pattern: ");
gets(rep_pat);
while(str[i]!='\O')
{
j=0,k=i;
while(str[k]==pat[j] && pat[j]!='\0')

{
k++;
j++;
}
if(pat[j]=='\O')
{
copy_loop=k;
while(rep_pat[rep_index] !='\O')
{
new_str[n] = rep_pat[rep_index];
rep_index++;
n++;
}
}
new_str[n] = str[copy_loop];
i++;
copy_loop++;
n++;
}
new_str[n]='\O';
printf("\n The new string is : ");
puts(new_str);
getch();
return O;
}
```

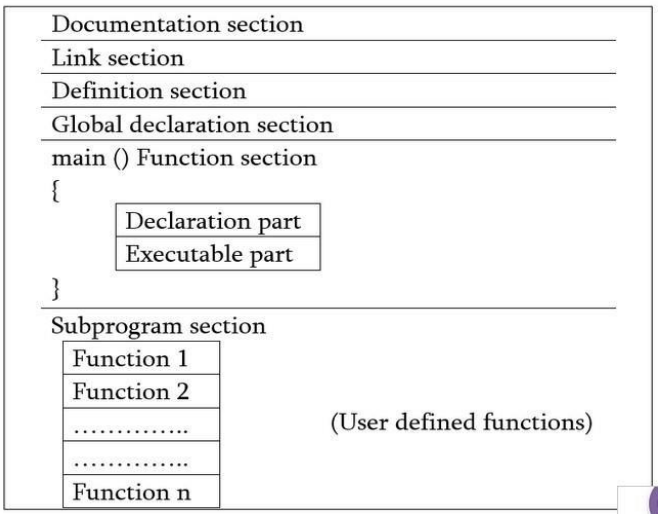
Output

```
Enter the string : How ARE you?
Enter the pattern to be replaced : ARE
Enter the replacing pattern : are
The new string is : How are you?
```

UNIT-1

Structure of a C program – compilation and linking processes – Constants, Variables – Data Types – Expressions using operators in C – Managing Input and Output operations – Decision Making and Branching – Looping statements. Arrays – Initialization – Declaration – One dimensional and Two- dimensional arrays. Strings- String operations – String Arrays. Simple programs- sorting,searching – matrix operations.

1. STRUCTURE OF C PROGRAM



Documentation section: The documentation section consists of a set of comment lines giving the name of the program, the author ,date on which program is written and other details, which the programmer would like to use later.

Link section:
The link section provides instruction to the compiler to link or include the required in-built functions from the system library such as using the #include directive.Eg #include<stdio.h>, #include<string.h>,#include<math.h>.

Definition section:
The definition section defines all symbolic constants using the #define directive(optional). Having the constants being defined here, we can use them elsewhere in code.

```
# define N 100    /* which means N's value is 100*/
# define pi 3.14
```

Global declaration section :
There are some variables that are used in more than one function. i.e common to more than one function. Such variables are called global variables and are declared in the global declaration section that is outside of all the functions.

main () function section :
Every C program must have one main function section. This section contains two parts; declaration part and executable part.

Declaration part : The declaration part declares all the variables used in the executable part.

Executable part : There is at least one or more statements in the executable part designed for some task\executing some logic.

These two parts must appear between the opening and closing braces. The program execution begins at the opening brace and ends at the closing brace. The closing brace of the main function is the logical end of the program. All statements in the declaration and executable part end with a semicolon(;).

Sub program section:
If the program is a multi-function program then the subprogram section contains definition of all the user-defined functions which were declared earlier in the Definition Section. User-defined functions are generally placed immediately after the main () function, although they may appear in any order.

Figure 1.2

```
main()
{
        Statement 1;
        Statement 2;
        .....
        Statement N;
}
Function1()
{
        Statement 1;
        Statement 2;
        Statement N;
}
Function2()
{
        Statement 1;
        Statement 2;
        Statement N;
}
FunctionN()
{
        Statement 1;
        Statement 2;
        Statement N;
}
```

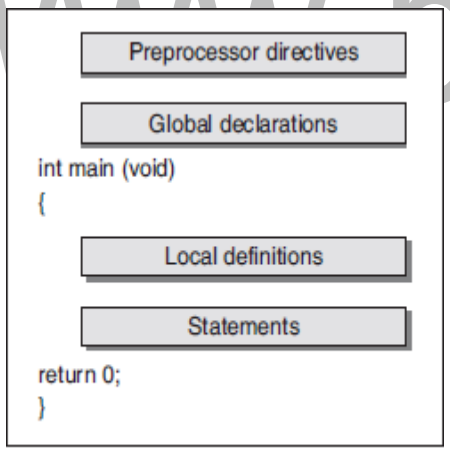



Figure Structure of a C program

PROGRAM STATEMENT

A statement performs an action when a program is executed.

All C program statements are terminated with a semi-colon **(;).**

Declaration statement: the name and type of the data objects needed duringprogram execution.

Example :-int a

Expression statement: is the simplest kind of statement

Example x = a+b*c^4 is an expression

Compound statement: is a sequence of statements that may be treated as asingle statement

Labeled statements: can be used to mark any statement so that control may betransferred to the statement by *switch* statement

Case 1:
labelABC:

Control statement: is a statement whose execution results in a choice beingmade as to which of two or more paths to execute.

Eg:categories of if and if..else

Selection statements: allow a program to select a particular execution pathfrom a set of one or more alternatives. **Eg Switch**

Iteration statements: are used to execute a group of one or more statementsrepeatedly. ***while, for,*** **and** ***do..while*** statements falls under this group.

Jump statements: cause an unconditional jump to some other place in theprogram. ***goto*** statement falls in this group.

EXAMPLE C PROGRAM

```
//sample.c//
#include<stdio.h>
main()
{
printf("welcome to C");
return 0;
}
```