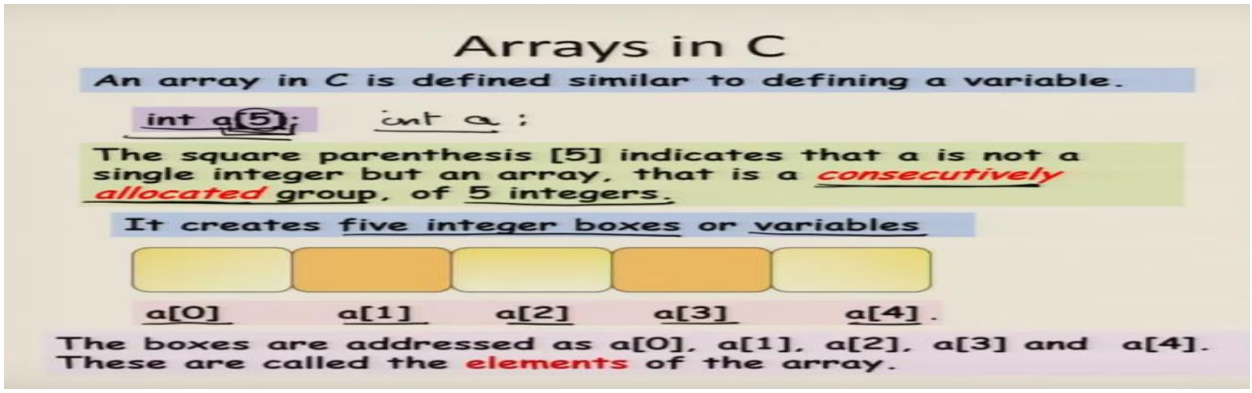
**Unit 4**

**Array (Searching Sorting Matrix )**

An Array is the collection of similar type of data in to a single variable.
A variable can hold multiple values of same type by using array.
Array is the collection of homogeneous data
In c programming if a student wants to store his five subject marks he has two options

1 Declare five different variables and store the marks in all the variables like int sub1=90 , int sub2 = 50 ,int sub3= 45 ,int sub4 = 35 ,int sub5=67
2 Declare an array of integer type that hold five subject marks like
int sub [5] = {90,50,45,35,67}

so there is a single variable sub that hold five different value .In option 1 if number of variable is increased then it is very difficult to declare and remember the name of variable for a programmer. To solve such type of problem we used Array.




**Advantages of Array:**

**1** It is used to represent multiple data items of same type by using only single name.
2 It can be used to implement other data structures like linked lists, stacks, queues etc.
3 2D arrays are used to represent matrices.
4 Arrays are very useful when you are working with sequences of the same kind of data
5 Arrays use reference type
6 . We can easily access each element of array. Not necessity to declare two many variables
7 . Not necessity to declare two many variables.
8. Array elements are stored in continuous memory location.
There are two types of arrays in C,

- Single Dimensional Array.
- Multi Dimensional Array.

The number of dimension of an array is determined by the number of indexes.

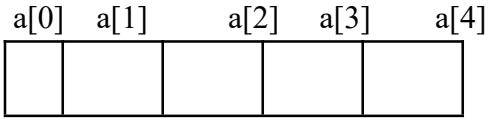
If there is only a single index then it is called one dimension array or Liner array and if there are two index one roe and one column it is called two dimensional array or Matrix.

**Declaration of one Dimensional Array**

data_type array_name[array_size];
For example:
int age[5];

Here, the name of array is *age*. The size of array is 5,i.e., there are 5 items(elements) of array *age*. All element in an array are of the same type .

char name[10];
double p[10];




the first element is numbered 0 and so on. if the size of an array is 10 ,the first index is 0 and second index is 1 and so on the last index is 9 .

In general $n^{th}$ element has the index (n-1).

Here, the size of array *age* is 5 times the size of int because there are 5 elements.

Suppose, the starting addres of age[0] is 2120 and the size of int be 2 bytes. Then, the next address (address of a[1]) will be 2122 , address of a[2] will be 2124 and so on.

**Initialization of one-dimensional array**

```
int age[5]={2,4,34,52,17};
int age[]={2,4,34,52,17};
```

| age[0] | age[1] | age[2] | age[3] | age[4] |
|---|---|---|---|---|
| 2 | 4 | 34 | 52 | 17 |

```
printf("%d",age[2]);     it display 34
printf("%d",age[0]);     it display 2
printf("%d",age[4]);     it display 17
```

```
void main()
{
int marks[5] ={ 35,55,43,67,82};
int i,sum=0;
printf("the element of the array is");
for(i=0;i<5;i++)
{
printf("Marks[%d]=%d\n", i, marks[i])
sum=sum+marks[i];
}
printf("Total=%d"\n,sum);
}
```

It is the static implementation of array.

```
Output: Marks[0]=35
        Marks[1]=55
        Marks[2]=43
        Marks[3]=67
        Marks[4]= 82
        Total= 282
```

```
#include <stdio.h>
void main()
{
int a[10]          // Array Declaration
int i ,
printf("Enter the element of an array");
for(i=0;i<10;i++)
{
scanf("%d",&a[i]);
}
printf("The element of the Array is");
for(i=0;i<10;i++)
{
printf("%d",a[i]);
}
```

This loop is used for input 10 elements using scanf() function

This loop is used for output display array elements

**Write a program to display the reverse of an array**

```
#include<stdio.h>
void main()
  {
   int a[50],i,n;
   printf("Enter the size of an array: ");
   scanf("%d",&n);
   printf("Enter the elements of the array: ");
   for(i=0;i<n;i++)
   {
       scanf("%d",&a[i]);
   } printf(" the reverse of an array is");
     for(i=n-1;i>=0;i--)
   {
Printf("%d",a[i]);
  } }
```

**Write a C program to Display Max Element of an array**

```
#include<stdio.h>
 void  main()  {
int  arr[100];     int  i ,n ,max;
printf("Enter total number of elements for Array");
 scanf("%d",&n);
for(i=0;i<n;++i)      {
scanf("%f",&arr[i]);     }
max=arr[0];
for(i=1;i<n;++i)       {
if(max]<arr[i])
 max=arr[i];    }
 printf("Largest element = %d",max);
 }
```

Write a program in C to copy the elements of one array into another array.
Write a program in C to count a total number of duplicate elements in an array.
Write a program in C to find the maximum and minimum element in an array.
Write a program in C to separate odd and even integers in separate arrays.
Write a program in C to sort elements of the array in descending order.
Write a program in C to delete an element at desired position from an array.
Write a program in C to find the second largest element in an array.
Write a program in C to find the number occurring odd number of times in an array.

## Searching

Searching is used to search any element from the given list.
There are two searching techniques.
One is linear search and another one is binary search.
Liner search is also called sequential search. and binary search is also called ordered search

## Liner search

- In Linear Search the list is searched sequentially and the position is returned if the key element to be searched is available in the list.
- The search in Linear Search starts at the beginning of an array and move to the end, testing for a match at each item.
- All the elements preceding the search element are traversed before the search element is traversed. i.e. if the element to be searched is in position 10, all elements form 1-9 are checked before 10

## Binary Search

- It can only be used for sorted arrays, but it's fast as compared to linear search.
- This search algorithm works on the principle of divide and conquer. For this algorithm to work properly, the data collection should be in a sorted form.

- When the list is sorted we can use the binary search technique to find items on the list. In this procedure, the entire list is divided into two sub-lists. If the item is found in the middle position, it returns the location, otherwise jumps to either left or right sub-list and do the same process again until finding the item or exceed the range.
-

**Linear search A go**

- Take the n element of an array a[i]
- Take the search element s
- for i from o to n
- if(a[i]==s) then display element found on position i+1 ; break;
- if(i=n ) display element not found

**Binary Search Algo**

Take the n element of an array a[i]
Take the search element search
beg =0 and end=n-1
mid=(beg+end)/2
While ( beg<end)
{
If(a[mid]<search) then beg=mid+1
else If(a[mid]==search) then found and display the position &break the loop
else
end=mid-1
mid=(beg+end)/2;
}
if(beg>end ) display element not in the list

**Linear Search Program**

```
include <stdio.h>
Void  main()
{
 int array[100], search, i, n;
 printf("Enter number of elements in array\n");
 scanf("%d", &n);
  for (i = 0; i < n; i++)
{
 scanf("%d", &array[i]);
   }
 printf("Enter a number to search\n");
 scanf("%d", &search);
 for (i = 0; i < n; i++)
 {
  if (array[i] == search)
  {
   printf("%d is present at
location %d.\n", search, i+1);
    break;
  }
 }
 if (i == n)
  printf("%d isn't present in the array.\n", search);
}
```

**Binary Search Program:**

```
#include <stdio.h>
Void  main()
{
 int  i, beg, end, middle, n,   search, a[100];
 printf("Enter number of elements\n");
scanf("%d", &n);
   for (i = 0;i < n; i++)
 {       scanf("%d", &a[i]);
    }
 printf("Enter value to find\n");
scanf("%d", &search);
beg = 0;   end =n-1;
 middle = (beg+end)/2;
   while (beg <= end)
   {
     if (a[middle] < search)
       beg= middle + 1;
   else if (a[middle] == search) {
    printf("%d found at
   location %d.\n", search, middle+1);
       break;
     }
     else
       end = middle - 1;
     middle = (beg +end)/2;
    }
    if (beg > end)
     printf("Not found! %d isn't present in the
   list.\n", search);  }
```

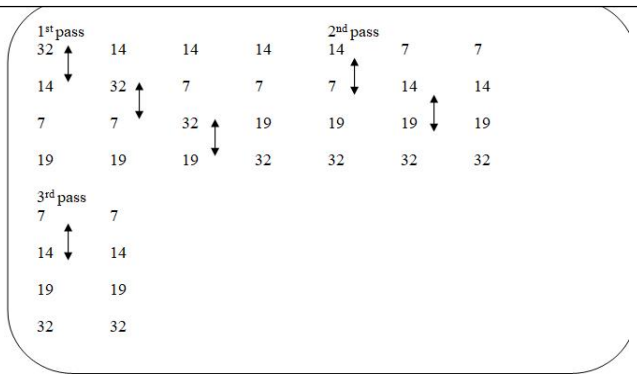
# Sorting

Sorting is a process to arrange the data in an order. Increasing order or Decreasing order.
Sorting is categorized as internal sorting and external sorting. Internal sorting means we arranging the number within the array only in the computer primary memory External Sorting means sorting the numbers from the external file by reading it from external memory.

- **Types of sorting**
- **Bubble Sort**
- **Insertion Sort**
- **Selection Sort**

## Bubble Sort

It is also called exchange sort .
In bubble sort one element compare with its adjacent element if 1st element is greater to its adjacent element than they swap the position and this process continue for all the elements.
In first pass the greater element is set on last position of the array. so if there in n element in an array than total number of passes is n-1.




**Bubble Sort Algo**

- Take the n element of an array a[i]
- for i from 1 to N
- for j from 0 to N – i
- if (a[j] > a[j + 1])
- {
- temp= a[j];
- a[j]=a[j+1];
- a[j+1]=temp;
- }
- Display the sorted array

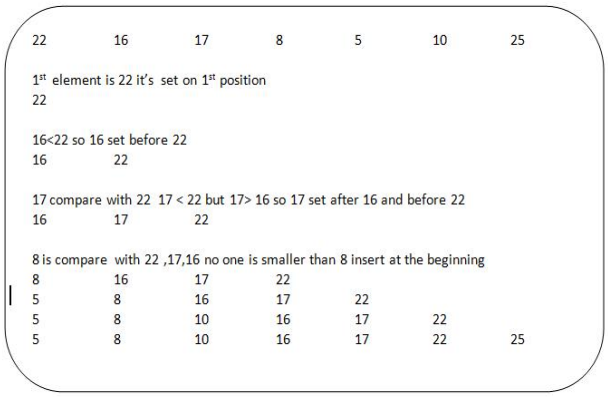
**Bubble Sort Program**

```
    void main()
    {
    int arr[50],temp,i,j,n;
    clrscr();
    printf("\nEnter any Value less Than 50");
    scanf("%d",&n);
    printf("\n\tEnter The Values into ARRAY ");
    for(i=0;i<n;i++)
    {
    scanf("%d",&arr[i]);
}
    for(i=1;i<n;i++)
    {
    for(j=0;j<n-i;j++)
    {
    if(arr[j] >arr[j+1])
    {
    temp=arr[j];
    arr[j]=arr[j+1];
    arr[j+1]=temp;
    } } }
    printf("\n\n The  Sorted Series is : ");
    for(i=0;i<n;i++) {
    printf("\n %d",arr[i]);  }   }
    } }
```

Output: Enter any Value less Than 50 : 10
Enter The Values into ARRAY: 10 7 19 14 32 6 12 8 5 4
The sorted series is : 4 5 6 7 8 10 12 14 19 32

## Insertation Sort

- In the insertion sort the element that is inserted into the array is set its own specific position.
- An insertion sort algo scans array from a[0] to a[n] ;
- inserting each element into its proper position in the previously sorted subarray. If a[3] comes and a[0] ,a[1] and a[2] are already into the sorted list than a[3] compare with a[2] ,a[1] and a[0] till the value of a[3] is smaller than a[2],a[1]and a[0] if value of a[3] is smaller than a[2] but grater than a[1] than a[3] is inserted after a[1] and before a[2].




**Insertion sort algo**

- Take the n element of an array a[i]
- for i= 1 to n
- key = A[i]
- j = i – 1
- while (j >= 0 and A[j] > key)
- {
- A[j + 1] = A[j]
- j = j – 1
- }
- A[j + 1] = key

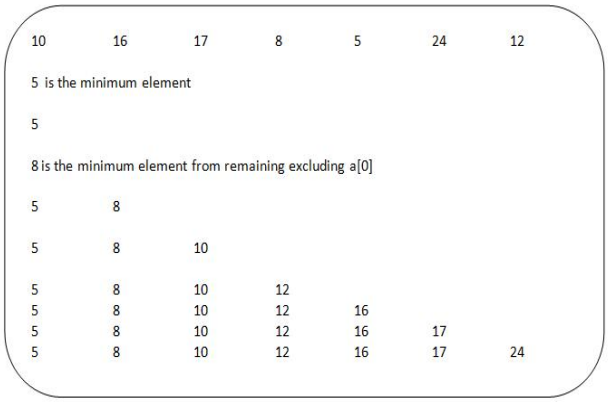
**Insertion Sort Program**

```
#include<stdio.h>
    Void  main()  {
    int i, j, n, key, arr[25];
    printf("enter no of elements in array ");
     scanf("%d",&n);
    printf("Enter %d elements: “);
    for(i=0;i<n;i++)  {
    scanf("%d",&arr[i]);  }
    for(i=1;i<n;i++)
    {
    key =arr[i];
    j=i-1;
    while (j >= 0 and A[j] > key)
    {
    A[j + 1] = A[j]
     j = j – 1
    }
     A[j + 1] = key;
      }
    printf("Order of Sorted elements:
    ");
    for(i=0;i<n;i++)
    printf(" %d“,arr[i]);
     }
```

## Selection sort

Select the minimum element from the array and set the minimum element from first to last position.
if there are n element in the array first find the minimum element from the array and set the min element on a[0] position.

Again find the minimum element from the array excluding a[0] and set that element on a[1] position. Again find the minimum element from the remaining array excluding a[0] and a[1] and set the minimum element on a[2] position continue the same process for n-1 times.




**Selection Sort Algorithm**

- Take the n element of an array a[i]
- for (i = 0; to (n - 1)
- pos= i;
- for (j = i + 1; to j < n)
  if (ar[pos} > arr[j])
  pos = j;
- if (pos != i)
  {
  temp = arr[i];
  arr[i] = arr[pos];
  arr[pos] = temp;

Display the sorted array

**Selection Sort Program**

```
#include <stdio.h>
Void  main()
{
int  arr[50], n,i, j, pos,temp;
  printf("Enter size of array ");
  scanf("%d", &n);
 printf("Enter  elements ", n);
  for (i = 0; i < n; i++)
    scanf("%d", &array[i]);
  for (i = 0; i < (n - 1); i++)
{
    pos= i;
for (j = i + 1; j < n; j++)
{
  if (ar[pos} > arr[j])
    pos = j;
  }
  if (pos != i)
  {
   temp = arr[i];
   arr[i] = arr[pos];
   arr[pos] = temp;
  }
}
printf("Sorted list is “);
for (i = 0; i < n; i++)
  printf("%d\n", arr[i]);
```

# 2D Array Matrix

If the number of indexes is more than one then such array are called multi dimensional array .Thus ,we can have a two dimensional array ,three dimensional array and so on

The dimension is determined by the number of pairs of square brackets placed after the array name.

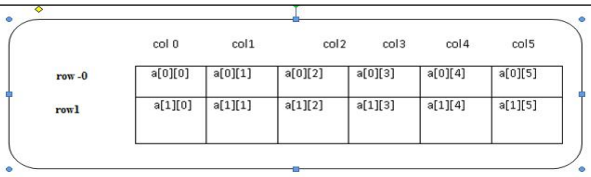
array [ ] → one dimensional array

- array [ ][ ] → two dimensional array
- array [ ][ ][ ] → Three dimensional array

**Declaration of two Dimensional Array**

- To declare a two-dimensional integer array of size x, y you would write something as follows:
- type arrayName [ Rows ][ Columns ];
- int arr[2][6];

This array has 2 rows and 6 columns .The total element of two dimension array is the multiply of row and column .means if array is a[2][6] then total no of element is 2*6 = 12




**Initialization of Multidimensional Arrays**

In C, two dimensional arrays can be initialized in different number of ways.

```
int c[2][3]={{1,3,0}, {7,5,9}};
          OR
int c[][3]={{1,3,0}, {7,5,9}};
          OR
int c[2][3]={1,3,0,7,5,9};
```

```
C[0][0] =  1        C[0][1] =  3        C[0][2] =  0
C[1][0] =  7        C[1][1] =  5        C[1][2] =  9
```

```
void main ()
{
 /* an array with   rows and 2 columns*/
 int a[4][2] = { {0,0}, {1,2}, {2,4}, {3,6}};
 int i, j;
 /* output each array element's value */
 for ( i = 0; i < 5; i++ )
 {
   for ( j = 0; j < 2; j++ )
   {
     printf("a[%d][%d] = %d\n", i,j, a[i][j] );
   }
 }
```

output:
```
a[0][0]: 0
a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
```

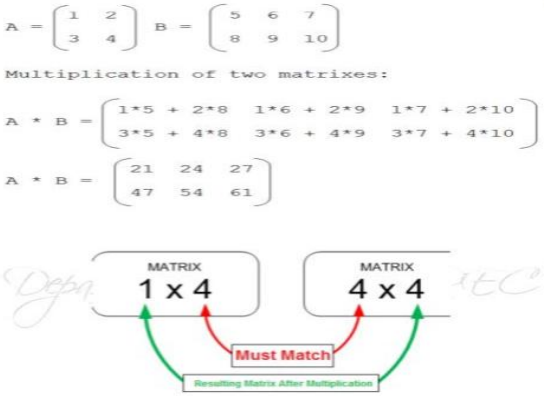
**Write a program to find the sum of the element a matrix**

```
void main() {
int a[10][10], r, c, i, j ,sum=0;
printf("Enter rows & column of matrix: ");
scanf("%d %d", &r, &c);
 printf("\nEnter elements of matrix:\n");
 for(i=0; i<r; i++)
{
for(j=0; j<c; j++)
 {
 scanf("%d",&a[i][j]);
 } }
 printf("\nThe Entered Matrix: \n");
for(i=0; i<r; i++)
{
 for(j=0; j<c; j++)
{
printf("%d  ",a[i][j]);
}
printf("\n");
   }
 printf("\nSum of the Matrix element
is:\n");
  for(i=0; i<c; i++)  {
  for(j=0; j<r; j++)   {
  sum=sum+a[i][j];
}   }
printf("%d",sum);   }
```

**Matrix Transpose**

```
void  main()
{
     int a[3][3], trans[3][3],I,j;
    printf("\nEnter elements of matrix:\n");
    for(i=0; i<3; i++)    {
    for(j=0; j<3; j++)  {
    scanf("%d",&a[i][j]);
    }}
    for(i=0; i<3; i++)    {
    for(j=0; j<3; j++)    {
     trans[i][j]=a[j][i];
} }
  printf("\nTranspose of Matrix:\n");
  for(i=0; i<3; i++)   {
   for(j=0; j<r; ++j)    {
 printf("%d  ",trans[i][j]);   }
       printf("\n\n");    }
```

**Matrix Multiplication**




**Matrix Multiplication**

```
#include<stdio.h>
Void  main()
 {
int a[10][10],b[10][10],mul[10][10],r,c,i,j,k;
system("cls");
printf("enter the number of row=");
scanf("%d",&r);
printf("enter the number of column=");
scanf("%d",&c);
printf("enter the first matrix elemen");
for(i=0;i<r;i++)    {
for(j=0;j<c;j++)    {
scanf("%d",&a[i][j]);    }    }

printf("enter the second matrix element");
for(i=0;i<r;i++)   {
    for(j=0;j<c;j++)    {
    scanf("%d",&b[i][j]);    }    }



    printf("multiply of the matrix=\n");
    for(i=0;i<r;i++)     {
    for(j=0;j<c;j++)     {
    mul[i][j]=0;
    for(k=0;k<c;k++)     {
    mul[i][j]=mul[i][j]+a[i][k]*b[k][j];
    }    }    }
    //for printing result
    for(i=0;i<r;i++)
    {
    for(j=0;j<c;j++)
```

**Q Write a program to perform the multiplication of two matrix of 5*6 and 6*3 if possible.**

Solution

Input for 1st matrix elements loop I move 0 to < 5 and j move 0 to <6

Input for 2nd matrix elements loop I move 0 to < 6 and j move 0 to <3

For  matrix multiplication loop

Loop i move  0 to <5

Loop j move 0 to <3

Loop k move 0 to <6

All other process are same

**Bubble sort Program to pass an array as a parameter in function**

```
#include <stdio.h>
void bubble_sort(int [], int);
    void  main()
    {
     int array[100], n, i;
     printf("Enter number of elements\n");
     scanf("%ld", &n);
     printf("Enter %ld integers\n", n);
     for (i = 0; i < n; i++)
       scanf("%ld", &array[c]);
     bubble_sort(array, n);
     printf("Sorted list in ascending order:\n");
     for (i = 0;i < n; i++)
        printf("%ld\n", array[i]);
    }
```

```
{
printf("%d\t",mul[i][j]);
}
printf("\n");
}    }
```

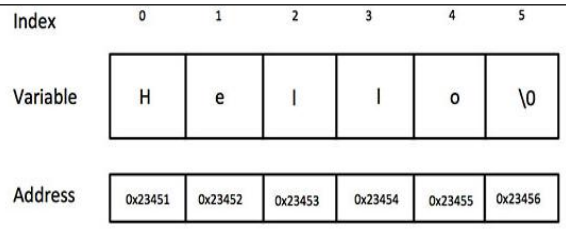
```
void bubble_sort(int list[], int n)
{
 long i, j, temp;
      for (i = 0 ; i < n - 1; i++)
      {
        for (j = 0 ; j < n - i - 1; j++)
     {
         if (list[d] > list[d+1]) {
          /* Swapping */
          temp        = list[j];
          list[j]   = list[j+1];
          list[j+1] = temp;
     } } } }
```

Write a program in C for addition of two Matrices of same size.
Write a program in C to find sum of right diagonals of a matrix.
Write a program in C to find the sum of left diagonals of a matrix.
Write a program in C to find sum of rows an columns of a Matrix.
Write a program in C to print or display the lower triangular of a given matrix.
Write a program in C to accept a matrix and determine whether it is a sparse matrix.
Write a program in C to accept two matrices and check whether they are equal

## String

Strings are actually one-dimensional array of characters terminated by a null character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a null.
The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
If you follow the rule of array initialization then you can write the above statement as follows −
char greeting[] = "Hello";




```
Void  main ()
 {
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
printf("Greeting message: %s\n", greeting );
}
```

Greeting message: Hello

<table>
<tr>
<td>String I/O functions<br><br>String I/O<br>1) printf and scanf<br>2) puts and gets<br>Syntax of above functions - Assume string as str1<br><br>printf("%s", str1);<br>puts(str1); --%s not require here.<br>scanf("%s", &str1);<br>gets(str1); --%s not require</td>
<td>Read & write Strings in C using Printf() and Scanf() functions<br>#include <stdio.h><br>Void main() {<br>char nickname[20];<br>printf("Enter your Nick name:");<br>scanf("%s", nickname);<br>printf("%s",nickname<br>}<br>Output:<br>Enter your Nick name: Neha Negam<br>It display only Neha</td>
</tr>
<tr>
<td>Read & Write Strings in C using gets() and puts() functions<br>#include <stdio.h><br>void main() {<br>char nickname[20];<br>puts("Enter your Nick name:");<br>gets(nickname);<br>puts(nickname);<br>}<br>Output:<br>Enter your Nick name: Neha Negam<br>It display only Neha Negam</td>
<td>when scanf() is used to read string input it stops reading when it encounters whitespace, newline or End Of File.<br>when gets() is used to read input it stops reading input when it encounters newline or End Of File.It does not stop reading the input on encountering whitespace as it considers whitespace as a string.<br>SCANF() is used to read input of any datatype BUT GETS ()is used only for string input.</td>
</tr>
</table>

**String Header file functions #include<string.h>**

- **strcpy(s1, s2);**

Copies string s2 into string s1.

- **strcat(s1, s2);**

Concatenates (attach) string s2 onto the end of string s1.

- **strlen(s1);**
  Returns the length of string s1.
- **strrev(s1);**
  Returns the reverse of string s1.
- **strcmp(s1, s2);**
  Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
- **strchr(s1, ch);**
  Returns a pointer to the first occurrence of character ch in string s1.
- **strstr(s1, s2);**
  Returns a pointer to the first occurrence of string s2 in string s1.

```
include <stdio.h>
#include <string.h>
void main () {
char str1[12] = "Hello";
char str2[12] = "World";
char str3[12];
 int len ;
/* total lenghth */
 len = strlen(str1);
printf(“ %d\n", len );
 /* copy str1 into str3 */
 strcpy(str3, str1);
printf("%s\n", str3 );
 /* concatenates str1 and str2 */
strcat( str2, str2);
printf("%s\n", str2 );
/* Reverse of str1 after concatenation */
Strrev(str1)
Puts(str1);
  }
```

**WAP to Check string is palindrome or not**

```
#include <stdio.h>
#include <string.h>
Void  main()
{
  char a[100], b[100];
  printf("Enter a string to check if it's a
palindrome\n");
  gets(a);
  strcpy(b, a);  // Copying input string
  strrev(b);  // Reversing the string
  if (strcmp(a, b) == 0)  // Comparing input
string with the reverse string
    printf("The string is a palindrome.\n");
  else
    printf("The string isn't a palindrome.\n");
  }
```

**C program for palindrome without using string functions**

```
#include <stdio.h>
Void  main()
{
char text[100];
int begin, middle, end, length = 0;
gets(text);
while ( text[length] != '\0' )
length++;
end = length - 1;
middle = length/2;
for( begin = 0; begin < middle ;
 begin++ )
{
if ( text[begin] != text[end] )
{
printf("Not a palindrome.\n");
break;
}
end--;
}
if( begin == middle )
printf("Palindrome.\n");
}
```

**Write a program to reverse a string without function**

```
#include <stdio.h>
Void  main()
{
  char s[1000], r[1000];
  int begin, end, len = 0;
  printf("Input a string\n");
  gets(s);
 // Calculating string length
  while (s[len] != '\0') {
    len++; }
  end = len - 1;
  for (begin = 0; begin < len; begin++) {
    r[begin] = s[end];
    end--;
  }
  r[begin] = '\0';
  printf("%s\n", r);
 }
```

**Write a program to sort the names of 10 Students**

```
#include <stdio.h>
#include <string.h>
void main()
{
 char name[10][8], temp[8];
int i, j, n;

printf("Enter the value of n \n");
scanf("%d", &n);
printf("Enter %d names n \n", n);
 for (i = 0; i < n; i++)
{
scanf("%s", name[i]);
}
for (i = 0; i < n - 1 ; i++)
{
for (j = i + 1; j < n; j++)
{
if (strcmp(name[i], name[j]) > 0)
{
strcpy(temp, name[i]);
strcpy(name[i], name[j]);
strcpy(name[j], temp);
} } }
printf("Input NamestSorted names\n");
for (i = 0; i < n; i++)
{
printf("t%s\n",, name[i]);
}
}
```

**String Programs**

Write a C program to convert lowercase string to uppercase.
Write a C program to find total number of alphabets, digits or special character in a string.
Write a C program to count total number of vowels and consonants in a string.
Write a C program to find first occurrence of a character in a given string.
C program to remove all repeated characters in a string

# Structure & Union

## Structure

Structure is a user defined data type. It is used to combine different types of data into a single type. It can have multiple members and structure variables.
Arrays allow to define type of variables that can hold several data items of the same kind.
Similarly structure is another user defined data type available in C that allows to combine data items of different kinds.
Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book –

Title , Author , Subject , Book ID

A structure can store different data types element like double,int,float,char.But array can store only one kind elements only.For eg :- int array can't store double or float type value.

### Defining a Structure

- To define a structure, you must use the **struct** statement. The struct statement defines a new data type, with more than one member. The format of the struct statement is as follows −

```
struct [structure tag]
{
member definition;
member definition; ... member definition;
} [one or more structure variables];
```

The **structure tag** is optional and each member definition is a normal variable definition, such as int i; or float f; or any other valid variable definition. At the end of the structure's definition, before the final semicolon, you can specify one or more structure variables but it is optional. Here is the way you would declare the Book structure

```
struct Books
    {
    char title[50];
    char author[50];
    char subject[100];
    int book_id;
    } book1;
```

**Accessing Structure Members**

To access any member of a structure, we use the **member access operator (.)**. The member access operator is coded as a period between the structure variable name and the structure member that we wish to access. You would use the keyword **struct** to define variables of structure type.
printf( "Book 1 title : %s\n", Book1.title);
printf( "Book 1 author : %s\n", Book1.author); printf( "Book 1 subject : %s\n", Book1.subject);
printf( "Book 1 book_id : %d\n", Book1.book_id);

**Structure with array**

**C Program to Store Information of Students Using Structure**

```
include <stdio.h>
struct student
{
char firstName[50];
 int roll;
float marks;
}

void main()
{
student s[10];
int i;
printf("Enter information of students:\n");
// storing information
 for (i = 0; i < 5; ++i)
{
printf("Enter Roll Number: ");
scanf("%d", s[i].roll);
printf("Enter first name: ");
scanf("%s", s[i].firstName);
printf("Enter marks: ");
 scanf("%f", &s[i].marks); }
 printf("Displaying Information:\n");
}
//displaying information
for (i = 0; i < 5; ++i)
{
printf("\nRoll no: %d\n", s[i].roll);
printf("First name: ");
 puts(s[i].firstName);
printf("Marks: %.1f", s[i].marks);
printf("\n");
```

```
}
 }
```

## Union

A **union** is a special data type available in C that allows to store different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multiple-purpose.

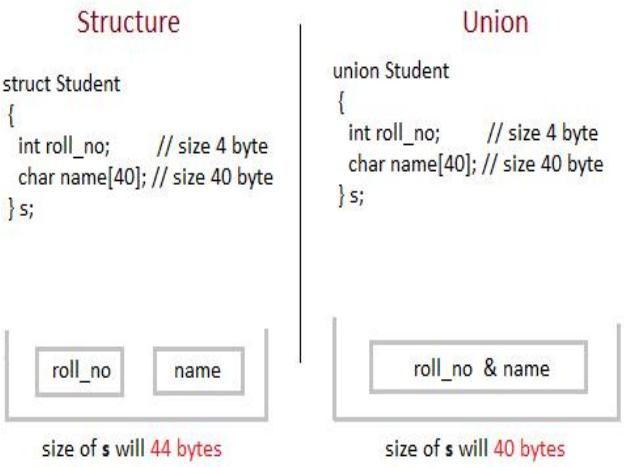
Defining a Union

To define a union, you must use the **union** statement in the same way as you did while defining a structure. The union statement defines a new data type with more than one member for your program. The format of the union statement is as follows −

```
union [union tag]
{
member definition;
 member definition; ...
member definition;
 } [one or more union variables];
```

**Difference between Structure & Union**

| | STRUCTURE | UNION |
|---|---|---|
| Keyword | The keyword **struct** is used to define a structure | The keyword **union** is used to define a union. |
| Size | When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is **greater than or equal to the sum of sizes of its members.** | when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of **union is equal to the size of largest member.** |
| Memory | Each member within a structure is assigned unique storage area of location. | Memory allocated is shared by individual members of union. |
| Value Altering | Altering the value of a member will not affect other members of the structure. | Altering the value of any of the member will alter other member values. |
| Accessing members | Individual member can be accessed at a time. | Only one member can be accessed at a time. |
| Initialization of Members | Several members of a structure can initialize at once. | Only the first member of a union can be initialized. |




Write a C Program to accept different goods with the number, price and date of purchase and display them.

Write a C Program to accept the empcode,empname,empbasic_salary of the employee and computer their gross salary .the gross salary is computed by Basic+DA+HRA .

Write a C Program to read the following information of 100 students Student name , Roll number ,Marks(out of 100)

The program should print the name and roll no of student who have obtain more than 60 marks.

Design a structure for a student (like Q3) and display the name of top ten student according to their marks.

# Asymptotic Notations

Asymptotic Notations are languages that allow us to analyze an algorithm's running time by identifying its behavior as the input size for the algorithm increases. This is also known as an algorithm's growth rate.
Asymptotic Notations are used to represent the Complexilty of any algorithms
The complexity of an algorithm describes the efficiency of the algorithm in terms of the amount of the memory required to process the data and the processing time.
Complexity of an algorithm is analyzed in two perspectives: Time and Space.

**Time Complexity**
It's a function describing the amount of time required to run an algorithm in terms of the size of the input. "Time" can mean the number of memory accesses performed, the number of comparisons between integers, the number of times some inner loop is executed, or some other natural unit related to the amount of real time the algorithm will take.

**Space Complexity**
It's a function describing the amount of memory an algorithm takes in terms of the size of input to the algorithm. We often speak of "extra" memory needed, not counting the memory needed to store the input itself. Again, we use natural (but fixed-length) units to measure this.

Asymptotic analysis refers to computing the running time of any operation in mathematical units of computation.
For example, the running time of one operation is computed as $f(n)$ and may be for another operation it is computed as $g(n^2)$. This means the first operation running time will increase linearly with the increase in **n** and the running time of the second operation will increase exponentially when **n** increases. Similarly, the running time of both operations will be nearly the same if **n** is significantly small.

Usually, the time required by an algorithm falls under three types −

- **Best Case** − Minimum time required for program execution.
- **Average Case** − Average time required for program execution.
- **Worst Case** − Maximum time required for program execution.

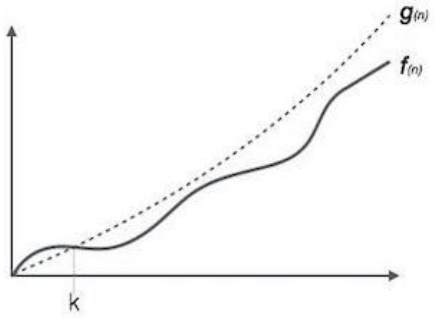
Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.

- O Notation
- Ω Notation
- θ Notation

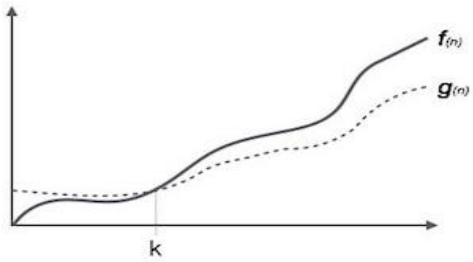
**Big Oh Notation O**
The notation O(n) is the formal way to express the upper bound of an algorithm's running time. It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.

- For example, for a function $f(n)$

- $O(f(n)) = \{ g(n)$ : there exists $c > 0$ and $n_0$ such that $f(n) \leq c.g(n)$ for all $n > n_0$. $\}$



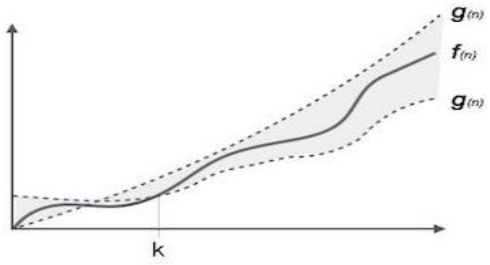

**Omega Notation Ω**




**Theta Notation(Θ)**

The notation Ω(n) is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.
For example, for a function $f(n)$
$\Omega(f(n)) \geq \{ g(n)$ : there exists $c > 0$ and $n_0$ such that $g(n) \leq c.f(n)$ for all $n > n_0$. $\}$

The notation θ(n) is the formal way to express both the lower bound and the upper bound of an algorithm's running time. It is represented as follows −
For example, for a function $f(n)$
$\theta(f(n)) = \{ g(n)$ if and only if $g(n) = O(f(n))$ and $g(n) = \Omega(f(n))$ for all $n > n_0$. $\}$




| Algorithm | Best Time Complexity | Average Time Complexity | Worst Time Complexity |
|---|---|---|---|
| Linear Search | O(1) | O(n) | O(n) |
| Binary Search | O(1) | O(log n) | O(log n) |
| Bubble Sort | O(n) | O(n^2) | O(n^2) |
| Selection Sort | O(n^2) | O(n^2) | O(n^2) |
| Insertion Sort | O(n) | O(n^2) | O(n^2) |