Title: **Arrays and Strings**	**Duration** (in hours):15 hours

Prerequisite:
Usage of Decision control statements(Branching and looping) statements along with derived datatypes in solving real time problems.

TOPIC CONTENT

1. One-dimensional array declaration, initialization, access array elements.
2. Other operations such as traversal, search and sort.
3. Implement Two-dimensional array and its operations.
4. Learn One-dimensional strings, declaration, initialization, manipulation.
5. Write modular C programs on 1D and 2D arrays .
6. Write modular C programs on strings without using built-in functions.

Context Building:

- The problems solved so far had solutions which involve only primitive datatypes(int, float,char,double) .
- There are some problems whose solutions are based on derived data type.
- Sometimes we come across the situation where we need to store more than one value. Variables store only one value at a time.

For ex: To store percentage of 100 students, there are two ways

i. we need to create 100 variables of type float.
ii. we create only one variable of type array to hold all 100 values.

- Arrays are the derived data type in C programming language which can store the primitive type of data such as int, char, double, float, etc.
- Arrays are set of similar data types stored in contigious memory locations.

Introduction

An array is a collection of similar data elements.They generally have the same data types.Instead of declaring variables individually, arrays help to declare them all together.

Definition and Declaration of arrays

- An array is defined in the same way as variables.
- The only difference is that of the size-specifier which tells us the size of the array.
- All the elements of an array are accessed with the help of an index.
- They have continuous memory location.

Types of Arrays

C language supports 3 types of Arrays

- One dimensional Array
- Multidimensinal Array (Two dimensional Array)

One dimensional Array-

Syntax:
data-type array-name [array-size];

Where,
data-type - all the array elements should have the same data-type. Any valid data-type of C can be taken.
array-name - is the name of the array.
array-size - it specifies the size of the array. It tells how many elements are present in the array.

The square brackets will tell us that we are dealing with an array in that particular program. They will use a continuous memory. The size of an array is optional. If the size is specified it should be matched with the definition.

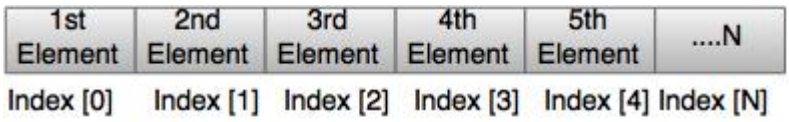


Fig: Array

In the above figure you can see the elements and the index that are alloted to them.
We can have as many elements as we want in an array.

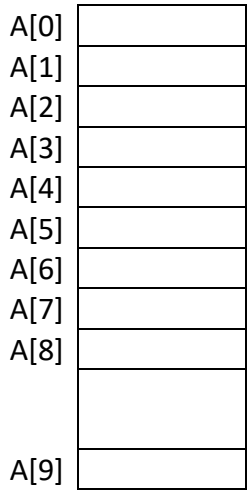
An array declaration cannot have the initial values.

E.g:
1. If we want to store 10 integer numbers then the array declaration will be,

int A[10];

Here A[0],A[1],A[2] ….. A[9] are the names given to continuous memory locations allocated for array A. The number enclosed within [ ] is called

as subscript/index of the array. These subscripts will specify position of elements in the array. Index should be a positive integer number.
2. float A[5];
3.double val[15];
4. char letter[6];
Total memory allocated for declared array is calculated using following **formula,**



Total memory=size of data_type * size of the array

For eg: int num[15];

Total memory allocated = sizeof(int)*size of the array= 2*15=30 bytes. Here data type is int which occupies 2-bytes and size of array is 15.

Initialization of arrays:

There are two ways of initializing arrays,

1. Compile-time initialization
2. Run-time initialization

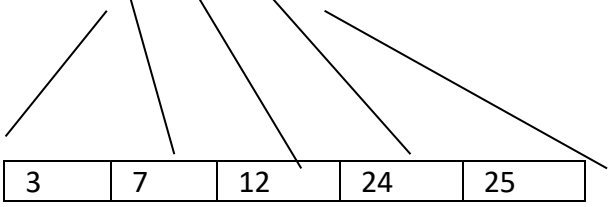
1. Compile-time initialization:

There are 4 types of compile-time initialization,

1. Basic initialization
2. Initialization without size
3. Partial initialization
4. Initialization to all zeros

1.Basic initialization:

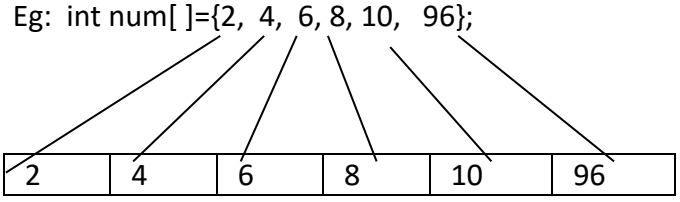
Eg: int num[5]={3, 7, 12, 24, 25};



Here, values to be initialized are enclosed within a pair of flower braces and are separated by comma. Number of initialized values is equal to the size of the array.

2. Initialization without size:

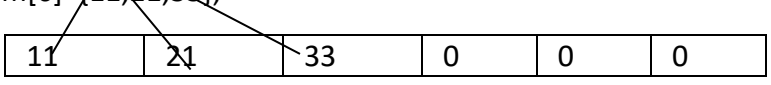
Eg: int num[ ]={2, 4, 6, 8, 10, 96};



Number elements present between {} will decide the size of the array. In the above eg size of array is 6.

3.Partial initialization:

Eg: int num[6]={11,21,33};



Here numbers of values to be assigned are only 3 but size of the array is 6. These 3 values are assigned to first 3 locations of the array and remaining positions are updated with zero automatically.

4. Initialization to all zero:

Eg: int val[5]={0};

0	0	0	0	0	0

2. Run-time initialization

Using standard input function scanf() we can initialize array during run-time.

Eg: To read 10 integer values into an array val following logic can be used

```
int val[10]; //declare array val
scanf("%d",&val[0]); //Reads an integer number into 0th position of array val
scanf("%d",&val[1]); //Reads an integer number into 1st  position of array val
scanf("%d",&val[2]); //Reads an integer number into 2nd  position of array val
.
.
.
scanf("%d",&val[9]); //Reads an integer number into 9th  position of array val
```

By observing above instruction we can say that we are repeatedly reading values into continuous index of the array i.e, val[0], val[1], val[2], …… val[9], therefore we can use looping concept as follows,

```
void Read(int val[10],int n)
{       int  i;
        for(i=0;i<n;i++)
        {   scanf("%d",&val[i]);  // Reads values into index locations 0,1,2…..9
        }
}
```

To display the contents of an array we can use the following logic.,

```
void print(int val[10],int n)
{       int  i;
        for(i=0;i<n;i++)
        {     printf("%d\t",val[i]);  Prints values from index locations 0,1,2…..9
        }
}
```

Accessing the elements of an array

An array element is accessed by using a subscript i.e. the index number is used.
Example:To show how elements are accessed
double bonus = arr[5];

In the above example, you can see that the 6th element will be accessed and the action will be performed on that. A loop is used for accessing all the elements of an array. The value of the subscript may vary and it should be an integral value or an expression that evaluates to an integral value.

Example: Setting each element of the array to 2
```
int i, arr[10];
for (i=0; i<10; i++)
arr[i] = 2;
```

All the elements of the array will be set to 2.
Due to the for loop first value of arr[0] is set to 2 and it will keep on going till arr[9].

Arrays can be passed as parameters to the functions in the following ways:

1**. Passing individual elements of the array**
Individual elements of an array can be passed to a function either by passing their address or the data values.

a) Passing by data values
The elements can be passed in the same manner as we pass the variables of other data type.
The only thing is that the data type of the array element should match the type of the function parameter.
Example
```
void main()     // Calling function
{
      int score[5] = {2,4,6,8,10};
      func (score[3]);
}

void func (int n)     // Called function
{
     printf("%d",n);
}
```

In the above example, one element of the array is passed to the called function.

b) Passing addresses

We can pass the elements by using the address operator (&) to the elements referenced index.

Since, we need to pass an address to the function we will use the indirection (*) operator.

Example

```
void main()     // Calling function
{
      int score[5] = {2,4,6,8,10};
      func (score[3]);
}
void func (int *n)     // Called function
{
      printf("%d",n);
}
```

2. Passing the entire array

While passing an entire array to a function we need to write only the name without the square brackets and the subscripts.

The formal argument which is corresponding is written in the same way but is declared inside the function.

The array dimension cannot be included in the formal declaration of the array.

Example

```
void main()     // Calling function
{
      int score[5] = {2,4,6,8,10};
      func (score);
}
void func (int score[5])     // Called function
{
      int i;
      for (i=0; i<5; i++)
      printf("%d",score[i]);
}
```

CASE STUDY – 1

A middle class family would always like to spend the money wisely, because of their financial status. They can't afford to purchase luxurious items unlike the Rich Class family. Even if they do, they do it very occasionally. In order to keep track of the expenditure of the entire year, the middle class family maintains a record of how much they spent every month of that year. Assume that for the year 2021, the family maintains a record of expenditure for each month. At the end of the year, the family would like to know the total and average expenditure. Apply problem solving framework to solve the problem.

Solution:PSF

1. **Understand the Problem**:

Knowns/Input: Number of months

Expenditure every month.

Unknowns/Output: Average expenditure of one year.

Extra information: Middle class family, Rich class family.

Assumptions: For the year 2021.

Months are considered as numbers.

(For Ex.: January as 1, February as 2 and so on).

Constraints:Number of months should be equal to 12.

2. **Devise a plan to solve the problem:**

Problem Representation: Numerically,Natural language.

Problem Solving Strategy: Mathematical Reasoning.

total_spent = total_spent + money[i]

average+spent = total_spent / 12

3. **Carry out the plan:**

Algorithm for main logic:

Algorithm: To find the average money spent in a year.

Step 1: Start

Step 2: Intialize total_spent ← 0

Step 3: Repeat until i ← months 0 to 11

total_spent ← total_spent + money[i]

Step 4: average_spent ← total_spent / 12

Step 5: Display total_spent, average_spent

Step 6: Stop

Write Complete modular C program with following modular functions:
read_money( ), display_money( ), compute_money_spent( ).

4. Assess the result**:**

Case1		
Input	Processing	Output
Months: 12 Expenditure: 49832 34829 21102 58222 10238 33390 12220 47472 25000 11200 34000 29005	Initialize total_spent=0 Repeat i from month 0 to 11 total_spent = total_spent + money[i] total_spent= 366510 average_spent= 366510 / 12 = 30542.5	total_spent= 366510 average_spent = 366510 / 12 = 30542.500000
Case 2		
Input	Processing	Output
Months: 12 Expenditure: 26322 15341 21333 45422 43222 3390 12421 43372 17000 13250 39500 7900	Initialize total_spent=0 Repeat i from month 0 to 11 total_spent = total_spent + money[i] total_spent= 288473 average_spent= 288473 / 12 = 24039.416667	total_spent= 288473 average_spent = 288473 / 12 = 24039.416667

Examples on 1-D array:

1.W.A.C-program to read elements into 1-D array and display

```
#include<stdio.h>
#include<stdlib.h>
void read(int A[30],int n);
void display(int A[30],int n);
main()
{
```

```
   int A[25],i,n;
   printf("Enter size of the array:\n");
   scanf("%d",&n);
   if(n<=0 || n>25)
   {
     printf("Invalid array size");
     exit(0);
   }
   read(A,n);
   display(A,n);
}

void read(int A[30],int n)
{
   int i;
    printf("Enter elements of the aray:\n");
   for(i=0;i<n;i++)
    scanf("%d",&A[i]); // Reads elements into array
}

void display(int A[30],int n)
{
   int i;
   printf("Elements of the aray are:\n");
   for(i=0;i<n;i++)
    printf("%d\t",A[i]); // Printd elements from array
}
```

2.W.A.C-program to read two arrays and perform the following operation, C=2A+B where A,B and C are 1-D arrays.

```
#include<stdio.h>
#include<stdlib.h>
void read(int A[30],int n);
void display(int A[30],int n);
void cal(int A[30],int B[30],int C[30],int n);
main()
{
   int A[25],B[25],C[25]={0}; //Array declaration
   int i,n,biggest,pos;

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

   if(n<=0 || n>25)
   {
     printf("Invalid array size");
```

```
      exit(0);
   }

   printf("Enter elements of the aray A:\n");
    read(A,n); // Reads elements into array A
   printf("Enter elements of the aray B:\n");
    read(B,n); // Reads elements into array B
    cal(A,B,C,n); //calculate C=2A+B
   printf("Elements of the aray C are:\n");
   display(C,n);// Print elements from array C
   }

void read(int A[30],int n)
{
   int i;
   for(i=0;i<n;i++)
     scanf("%d",&A[i]); // Reads elements into array
}

void display(int A[30],int n)
{
   int i;
   for(i=0;i<n;i++)
     printf("%d\t",A[i]); // Printd elements from array
}

void cal(int A[30],int B[30],int C[30],int n)
{
   int i;
   //Calculate C=2A+B
   for(i=0;i<n;i++)
     C[i]=2*A[i]+B[i];
}
```

3.W.A.C-program to read elements into the array and perform linear search operation also find how many times the key element is present in the array.

```
#include<stdio.h>
#include<stdlib.h>
void read(int A[30],int n);
void display(int A[30],int n);
int linear(int A[30],int n,int key);
main()
{
   int A[25],i,n,key,occ=0;
```

```
   printf("Enter size of the array:\n");
   scanf("%d",&n);

   if(n<=0 || n>25)
   {
     printf("Invalid array size");
     exit(0);
   }

   printf("Enter elements of the aray:\n");
   read(A,n); // Reads elements into array
   printf("Elements of the aray are:\n");
   display(A,n); // Displays elements of the array
   printf("\nEnter key element to be searched:\n");
   scanf("%d",&key);
   occ=linear(A,n,key);

   if(occ>0) //If it has occurred more than zero times then search successful
   {
     printf("Search is successful!!!\n");
     printf("Occurannce=%d\n",occ);
   }
   else
    printf("Search not successful!!!\n");

}

void read(int A[30],int n)
{
   int i;
   for(i=0;i<n;i++)
    scanf("%d",&A[i]); // Reads elements into array
}

void display(int A[30],int n)
{
   int i;
   for(i=0;i<n;i++)
    printf("%d\t",A[i]); // Printd elements from array
}

int linear(int A[30],int n,int key)
{
   int i,occ=0;

  for(i=0;i<n;i++)
```

```
   {
      if(key==A[i]) //compare key ele with all ele of the array
      {
        occ++; //Increment occurrence count
      }
   }
   return(occ);
}
```

4. **W.A.C-program to perform binary search operation**
Note : Binary search operation is done only on sorted list or array

```
#include<stdio.h>
#include<stdlib.h>
void read(int A[30],int n);
void display(int A[30],int n);
void Binary(int A[30],int n,int key);
main()
{
   int A[25],i,n,key,flag;
   printf("Enter size of the array:\n");
   scanf("%d",&n);
   if(n<=0 || n>25)
   {
     printf("Invalid array size");
     exit(0);
   }

   printf("Enter elements of the aray:\n");
   read(A,n); // Reads elements into array
   printf("Elements of the aray are:\n");
   display(A,n); // Displays elements of the array
   printf("\nEnter key element:\n");
   scanf("%d",&key);
   flag=binary(A,n,key);
   if(flag==1)
    printf("Serach is successful!!!");
   else
   printf("Serach not successful!!!");

}

void read(int A[30],int n)
{
   int i;
   for(i=0;i<n;i++)
    scanf("%d",&A[i]); // Reads elements into array
```

```
}

void display(int A[30],int n)
{
   int i;
   for(i=0;i<n;i++)
     printf("%d\t",A[i]); // Printd elements from array
}

int binary(int A[30],int n,int key)
{
   int lower,upper,mid;

   lower=0; //set lower limit to zero subscript
   upper=n-1; //set upper limit to last subscript

   while(lower<upper)
   {
      mid=(lower+upper)/2; //Find middle position
      if(key==A[mid]) // condition for successful search
        return(1);
      else if(key<A[mid]) //key present in the 1st half of array
        upper=mid-1;
      else   //key present in the 2nd half of array
        lower=mid+1;
   }

}
```

5.C-program to carry out Bubble sort

```
#include<stdio.h>
void read(int A[30],int n);
void display(int A[30],int n);
void bubble(int A[30],int n);
main()
{
  int A[20],i,n,temp,j,k;

  printf("Enter the size of array\n");
  scanf("%d",&n);
  printf("Enter elements of the aray:\n");
  read(A,n); // Reads elements into array
  printf("Elements of the aray before sorting:\n");
  display(A,n); // Displays elements of the array
  bubble(A,n); //Call sorting function
  printf("\nElements of the aray after sorting:\n");
```

```
  display(A,n); // Displays sorted array
}
void read(int A[30],int n)
{
   int i;
   for(i=0;i<n;i++)
     scanf("%d",&A[i]); // Reads elements into array
}

void display(int A[30],int n)
{
   int i;
   for(i=0;i<n;i++)
     printf("%d\t",A[i]); // Printd elements from array
}

void bubble(int A[30],int n)
{
   int i,j,temp;
 //Bubble sort logic
  for(i=0;i<=n-2;i++)    //Loop to pickup element for comparison from unsorted part
  {
      for(j=0;j<=(n-2-i);j++)  //Loop to settle  biggest element to the bottom of array
       {
          if(A[j]>A[j+1])  //Compare adjacent elements and exchange if condition is true
          {
             temp=A[j];
             A[j]=A[j+1];
             A[j+1]=temp;
          }
       }
  }
}
```

6. Program to reverse the elements of the given list

```
#include<stdio.h>
void read(int A[30],int n);
void display(int A[30],int n);
void reverse(int A[30],int n);
main()
{
  int A[10],i,n,temp,j,k;

  printf("Enter the size of array\n");
  scanf("%d",&n);
  printf("Enter elements of the aray:\n");
  read(A,n); // Reads elements into array
```

```
  printf("Elements of the aray before sorting:\n");
  display(A,n); // Displays elements of the array
  reverse(A,n); //Call sorting function
  printf("\nElements of the aray after sorting:\n");
  display(A,n); // Displays sorted array
  }



void read(int A[30],int n)
{
   int i;
   for(i=0;i<n;i++)
     scanf("%d",&A[i]); // Reads elements into array
}


void display(int A[30],int n)
{
   int i;
   for(i=0;i<n;i++)
     printf("%d\t",A[i]); // Printd elements from array
}


void reverse(int A[30],int n)
{
   int i,k,temp;

  for(i=0,k=n-1;i<k;i++,k--)
  {
     temp=A[i];
     A[i]=A[k];
     A[k]=temp;
   }
}
```

Differences between linear search and binary serach:

BINARY SEARCH	LINEAR SEARCH
1.Works only on sorted items	1. Works on sorted as well as unsorted items
2.Very efficient if the items are sorted	2. Very efficient if the items are less and present in the beginning of the list
3.Works well with arrays and not on linked lists	3.Works with arrays and linked lists
4.Number of comparisons are less	4.More number of comparisons are required if the items are present in the later part of

	the array or if elements are more

STRINGS

A string is an one-dimensional array of characters terminated by NULL character ('\0').

NULL character is also called as "terminating character" and is important because it is the only way with which we can know the end of the string.

String at compile time can be initialized as follows,

a. char college[8] = "BVBCET"

The following is the memory representation of the above declaration

B	V	B	C	E	T	'\0'	'\0'
1000	1001	1002	1003	1004	1005	1006	1007

(1000 – 1007) are the memory address assumed randomly. We know that character takes 1 byte, so each location is after 1 byte. Note that the remaining characters are filled with NULL.

The size of the array must be greater than or equal to the size of string. (Sizeof(college) >= sizeof("BVBCET"))

Note that in this type of declaration, where we initialize the string in double quotes, NULL character is not necessary. C automatically inserts NULL character. ----------

b. char college[8]={'B','V','B','C','E','T','\0'};

The following is the memory representation of the above declaration

B	V	B	C	E	T	'\0'	JUNK
2000	2001	2002	2003	2004	2005	2006	2007

(2000 – 2007) are the memory address assumed randomly. We know that character takes 1 byte, so each location is after 1 byte.

Note that the remaining characters are not filled with NULL. They are filled with JUNK (GARBAGE) values.

The size of the array must be greater than or equal to the size of string. (Sizeof(college) >= sizeof("BVBCET"))

Note that in this type of declaration, where we initialize the string character by character, we need to explicitly specify the NULL character. C doesnot automatically inserts NULL character.

%s format specifier is used to read a string

Program to read and display the string

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

```
char name[100];
printf("Enter name\n");
scanf("%s",name);
printf("Name is:");
printf("%s\n",name);
}
```

Output:
Enter name
Ram
Name is: Ram

Note: We don't use '&' in scanf function to read the string.

String is an array of characters. Array represents address. (Array represents base address). & is an address operator. We use & in other data types to get the address. Here we are getting address by default (due to array name), so there is no need to use &.

```
#include<stdio.h>
void read_string(char string[]);
void display_string(char string[]);
main()
{
char string[100];
printf("Enter string:\n");
read_string(string);
printf("The string is :");
display_string(string);
}
void read_string(char string[])
{
scanf("%s",string);
}
void display_string(char string[])
{
printf("%s",string);
}
```

Note:

1. for multi-word string you can use gets, in place of scanf and puts inplace of printf.
2.We pass string to a function with its name. Same as array, as string is character array.

Working of String handling functions with an example.

a. strcpy()
b. strcat()
c. strcmp()
d. strlen()

a. strcpy() : copies one string to another. str stands for string and cpy for copy.
General Syntax: strcpy(s1,s2); copies s2 to s1.
Ex: Let s1="Hubli" and s2="Karnataka"

after we execute, strcpy(s1,s2);
s1 is overwritten with contents of s2 (Karnataka)
---
b. strcat() : concatenates (appends /adds) one string at the end of other string. str stands for string and cat for concatenate

General Syntax: strcat(s1,s2); concatenates string s2 at the end of string s1
Ex: Let s1="Hubli" and s2="Karnataka"
after we execute, strcat(s1,s2);
s1 becomes "HubliKarnataka"
---

c. **strcmp() :** compare 2 strings and returns ASCII difference of 1st non- matching character. str stands for string and cmp for compare.
General Syntax: strcmp(s1,s2); compares string s1 with string s2.

Ex: Let s1="there" and s2="their"
after we execute, strcmp(s1,s2);
It returns 1.
Explanation: The first non-matching character is 'r' of 1st string and 'i' of 2nd string. ASCII value of 'r' = 114 and ASCII value of 'i' = 105 . (114-105)>0. Hence it returns 1.
Note: If two strings are excatly same strcmp returns 0.
---
d. strlen() : It gives string length. str stands for string and len for length.

General Syntax: strlen(s1): Gives length of string s1.
Ex: Let s1="Hubli karnataka".
after we execute, strlen(s1), it returns 15. (Even spaces are counted)

1.Simulate strcat and strcmp function.

```
#include<stdio.h>
#include<string.h>
#define MAX_SIZE 50
void string_concatenation(char s1[], char s2[]);
int string_compare(char s1[], char s2[]);
main()
{
char s1[MAX_SIZE], s2[MAX_SIZE] ;
printf("enter two strings\n");
scanf("%s%s",s1,s2);
string_concatenation(s1,s2);
printf(" after concat s1=%s\n", s1);
```

```
printf("comaparision of s1 and s2\n");
printf("enter two strings\n");
scanf("%s%s",s1,s2);
int d = string_compare(s1,s2);
if(d == 0)
printf("s1 and s2 are equal\n");
else if(d<0)
printf("s1 is less than s2\n" );
else
printf("s1 is greater than s2\n");
}
//this function is valid untill size of s1 is sufficient to accomodate s2
void string_concatenation(char s1[], char s2[])

{
int i=0,j=0;
//loop to set i to last character. i.e index of '\0'
//This while loop, finds the length of string s1
while(s1[i] != '\0')
{
i++;
}
//loop to apend s2 to s1
while(s2[j] != '\0')
{
s1[i]=s2[j];
i++;
j++;
}
//at last set null character
s1[i]='\0';
}
//function to compare two strings
int string_compare(char s1[], char s2[])
{
int i=0,flag=0, difference=0;
while(s1[i] != '\0' && s2[i] != '\0')
{
if(s1[i]==s2[i])
{
i++;
}
else
{
flag=1;
difference=s1[i]-s2[i];
break;
```

```
}
}
if(flag==0)
return 0;
else

return difference;
}
```

Example2- Seeta and Geeta were writing an article on women's empowerment. They want to know if they both have used same characters (only alphabets ignoring case) same number of time. Write a modular program which reads text of both Seeta and Geeta and help them to know if they have used same characters same number of time. (Anagram)

```
#include<stdio.h>
#include<string.h>
int isAnagram(char string1[], char string2[]);
#define MAX 200
main()
{
char string1[MAX];
char string2[MAX];
printf("Enter the text written by Seeta:\n");
gets(string1);
printf("Enter the text written by Geeta:\n");
gets(string2);
if(isAnagram(string1,string2))
{
printf("Seeta and Geeta have used same characters same
number of time\n");

}
else
{
printf("Seeta and Geeta have not used same characters same
number of time\n");

}
}
int isAnagram(char string1[], char string2[])
{
/*
Initialize two count arrays of size 26 (A-Z), for both texts
written by Seeta and Geeta. Initially all counts will be Zero.
We know that ASCII value of 'A' = 65 'B' = 66 and so on..
So count[any_character – 'A'] will give us the count of that
particular character.
*/
```

```
int count1[26]={0}; //All are initialized to ZERO.
int count2[26]={0};
int i =0;

int j=0;
char ch;
/* This while loop for text written by Seeta*/
while(string1[i]!='\0')
{
ch =toupper(string[i]); //converts to upper-case
if(ch>='A' && ch<='Z') //Consider only alphabets
{
count1[ch-'A']++;
}
i++;
}
/* This while loop for text written by Geeta*/
while(string2[j]!='\0')
{
ch =toupper(string[j]); //converts to upper-case
if(ch>='A' && ch<='Z')
{
count2[ch-'A']++;
}
j++;
}
for(i=0;i<26;i++)
{
if(count1[i]!=count2[i])
return 0;

}
return 1;

}
```

Output 1:
Enter the text written by Seeta:
SILeNT
Enter the text written by Geeta:
LISTEN
Seeta and Geeta have used same characters same number of time

3. **Write a modular C program to delete a substring from a given text.**
Ex: DELETE(str, position, length);
DELETE("ABCDXXXABCD",5,3);
This means delete 3 characters from position 5. Therefore the output would be ABCDABCD.

```
#include<stdio.h>
#include<string.h>
void DELETE(char str[],int position, int length);
main()
{
char str[100];
int position, length;
printf("Enter the string\n");
scanf("%s",str);
printf("From which position the characters of a string should be
deleted\n");
scanf("%d",&position);
printf("How many character should be deleted from %d
position\n",position);

scanf("%d",&length);
DELETE(str,position,length);
printf("String after deletion of characters = %s\n",str);
}
void DELETE(char str[],int position, int length)
{
int i=0;
int j =0;
char target[100];
while(i<position-1)
{
target[j]=str[i];
i++;
j++;
}
i = i + length;
while(str[i]!='\0')
{
target[j]=str[i];
i++;
j++;
}
target[j]='\0';
strcpy(str,target); }
```

Two dimensional Array

An array having two subscripts or a pair of index values is known as 2-Dimensional array. Usually used to perform matrix calculations.

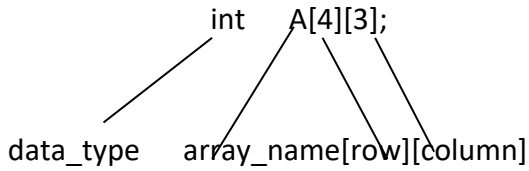
Declaration syntax:

data_type array_name[row][column];

where,

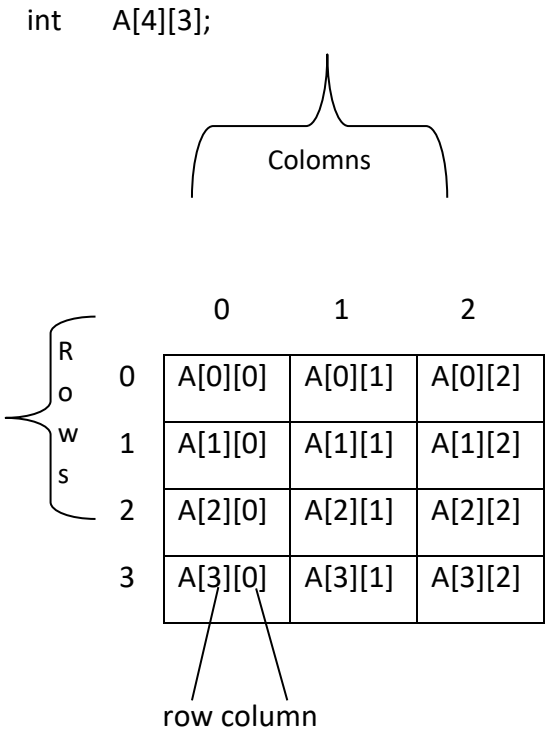
data_type: is any of the basic datatype like int,float,double or char
array_name: is name of the array which is an valid identifier
row : row size
column : column size

For eg: 1. To declare a matrix of size 4x3 we have the following declaration.



2. float mat[5][4];
3. double num[3][3];

Memory Map:



Initialization of 2-D array:

Following are the ways to initialize 2D array during compile-time.

1**.Basic inititalization:**

int M[2][3]={10,15,2,6,18,8};

	0	1	2
0	10	15	2
1	6	18	8

Above initialization can also be done as follows

```
int M[2][3]={
                {10,15,2},
                {6,18,8 }
               };
```

2.Initialization without row size :We can consider each row of 2D array as individual 1D arrays. Its possible to initialize 1D array without specifying the size of the array. Therefore we skip row size.

```
For eg:  int Array[ ][2]={
                {10,20},
                {30,40}
              };
```

3. Partial initialization: We can do the partial initialization as follows,

```
int Y[3][2]={5,2,6};
```

In the above declaration we have initialized only first three positions of the array.

Uninitialized positions are updated to zero.

	0	1
0	5	2
1	6	0
2	0	0

4. Initialization to all zero

int Y[3][2]={0};

	0	1
0	0	0
1	0	0
2	0	0

Run-time initialization can be done as follows:

```
int   B[3][2];
for(i=0;i<3;i++)       //Loop to keep track of rows
{
  for(j=0;j<2;j++)   //Loop to keep track of columns
    {
        scanf("%d",&B[i][j]);  //Reads value in B[i][j] position
     }
 }
```

Examples on 2D array:

1. C-program to find sum of principle diagonal,upper triangular and lower triangular elements

```
#include<stdio.h>
#include<stdlib.h>
void read(int A[10][10],int m,int n);
void display(int A[10][10],int m,int n);
int calsum(int A[10][10],int m,int n);
main()
{
```

```
   int A[10][10],i,j,psum=0,m,n;

   printf("Enter order of the matrix\n");
   scanf("%d%d",&m,&n);
   if(m!=n)
   {
      printf("Invalid size of the matrix\n");
      exit(0);
   }
   printf("Enter elements of the matrix\n");
   read(A,m,n);
   printf("Entered matrix is:\n");
   display(A,m,n);
   psum=calsum(A,m,n);
   printf("Sum of principle diagonal elements is=%d\n",psum);
   }
void read(int A[10][10],int m,int n)
{
    int i,j;

   for(i=0;i<m;i++)
    for(j=0;j<n;j++)
     scanf("%d",&A[i][j]);
}

void display(int A[10][10],int m,int n)
 {
   int i,j;
   for(i=0;i<m;i++)
   {
      for(j=0;j<n;j++)
       printf("%d\t",A[i][j]);

    printf("\n");
   }
   return;
}

int calsum(int A[10][10],int m,int n)
{
   int i,j,usum=0,lsum=0,psum=0;

   for(i=0;i<m;i++)
    {
      for(j=0;j<n;j++)
       {
          if(i==j)
```

```
            psum=psum+A[i][j];
           else if(i<j)
            usum=usum+A[i][j];
           else
             lsum=lsum+A[i][j];
        }
     }
     printf("Sum of upper triangular elements is=%d\n",usum);
     printf("Sum of lower triangular elements is=%d\n",lsum);
     return(psum);
 }
```

2.C-Program to find norm of the matrix

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
void read(int A[10][10],int m,int n);
void display(int A[10][10],int m,int n);
float calnorm(int A[10][10],int m,int n);
main()
{
   int A[10][10],i,j,m,n;
   float norm=0;
   printf("Enter order of the matrix\n");
   scanf("%d%d",&m,&n);
   if(m!=n)
   {
      printf("Invalid size of the matrix\n");
      exit(0);
   }
   printf("Enter elements of the matrix\n");
   read(A,m,n);
   printf("Entered matrix is:\n");
   display(A,m,n);
   norm=calnorm(A,m,n);
   printf("Norm of the given matrix is=%f\n",norm);
   }
void read(int A[10][10],int m,int n)
{
    int i,j;

   for(i=0;i<m;i++)
    for(j=0;j<n;j++)
     scanf("%d",&A[i][j]);
}

void display(int A[10][10],int m,int n)
```

```
{
   int i,j;
   for(i=0;i<m;i++)
   {
      for(j=0;j<n;j++)
       printf("%d\t",A[i][j]);

    printf("\n");
   }
   return;
}

float calnorm(int A[10][10],int m,int n)
{
  int i,j,sum=0;
  float norm;

  for(i=0;i<m;i++)
   {
     for(j=0;j<n;j++)
      sum=sum+A[i][j]*A[i][j];
   }
  norm=sqrt(sum);
  return(norm);
}
```

3.Find maximum element in each row

```
#include<stdio.h>
#include<stdlib.h>
void read(int A[10][10],int m,int n);
void display(int A[10][10],int m,int n);
void rowmax(int A[10][10],int m,int n);
main()
{
   int A[10][10],m,n;

   printf("Enter size of the matrix\n");
   scanf("%d%d",&m,&n);
   read(A,m,n);
}
void read(int A[10][10],int m,int n)
 {
    int i,j;

  printf("Enter elements of the matrix\n");
   for(i=0;i<m;i++)
```

```
    for(j=0;j<n;j++)
     scanf("%d",&A[i][j]);

  printf("Entered matrix is\n");
  display(A,m,n);
}
 void display(int A[10][10],int m,int n)
{
   int i,j;
   for(i=0;i<m;i++)
   {
      for(j=0;j<n;j++)
       printf("%d\t",A[i][j]);

    printf("\n");
   }
   rowmax(A,m,n);
}
void rowmax(int A[10][10],int m,int n)
{
    int i,j,max,k=0,l=0;

  for(i=0;i<m;i++)
   {
      max=A[i][0];
    for(j=0;j<n;j++)
    {
      if(max<A[i][j])
         max=A[i][j];
    }
    printf("Maximum element in %d row is : %d \n",i,max);
   }
}
```