Programming in C

I semester, BCA

Ambition Guru College

Compiled by :
Ankit Bhattarai

Unit V

(7 hrs.)

Array, Pointer and String

- Concept of array, Array declare, access and initialization, Multi-dimensional array, Strings and string manipulation, Passing array and string to function, Concept of Pointer, Pointer address, dereference, declaration, assignment, initialization, Pointer Arithmetic, Array and Pointer, String functions in C

Introduction to Array

- Array is a set of similar data elements which are stored in consecutive memory location under a common variable name.

So to be an array,

- All elements must be of same data type.
- All elements are stored in consecutive memory location.

E.g. If we want to store 100 integers in a sequence, we can create array for it.

```
int num[100];
```

Note: *The size and type of array cannot be changed after its declaration*

Introduction to Array

- If we want many elements of similar type than it is not feasible to declare all variables and also manipulate these elements. So in this case we use array.

For example, if we want 100 integer variables, then instead of writing all 100 variables, we use array like int num[100]. Here 100 integer elements num[0], num[1], num[2]. . . . . .num[99] are declared.

Introduction to Array

Limitation of Array

- Memory allocation in array are static in nature. Which means memory is allocated before the execution of program begins (During compilation).In this type of allocation the memory cannot be resized after initial allocation.

So it has some limitations.

- Wastage of memory
- Overflow of memory

e.g. int num[100];

- Here, the size of an array has been fixed to 100.If we just enter to 10 elements only, then their will be wastage of 90 memory location and if we need to store more than 100 elements there will be memory overflow.

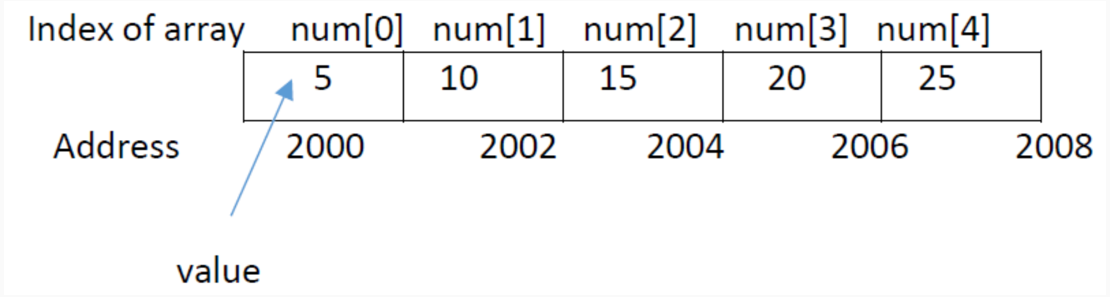
Types of Array

1. One dimensional Array
2. Multidimensional Array

Types of Array

One dimensional array

- Elements of an array can be represented either as a single row or single column.
- There is a single subscript or index whose value refers to the individual array element, which ranges from 0 to n-1, where n is the size of array.
- Eg. int num[5]={5,10,15,20,25};
- Assuming base address 2000, now array elements can be illustrated as:

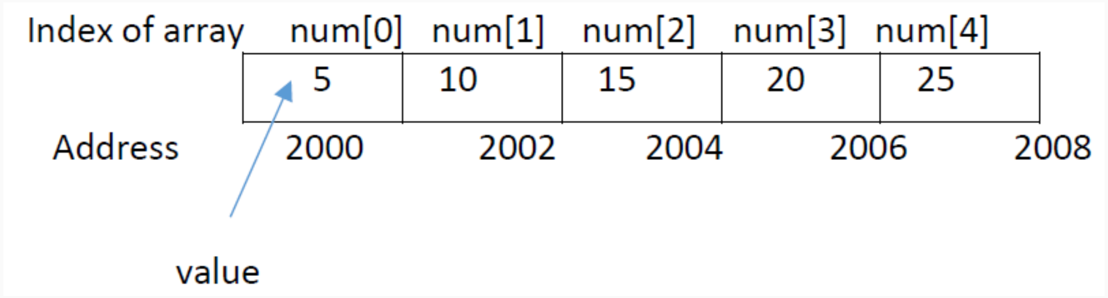


Types of Array

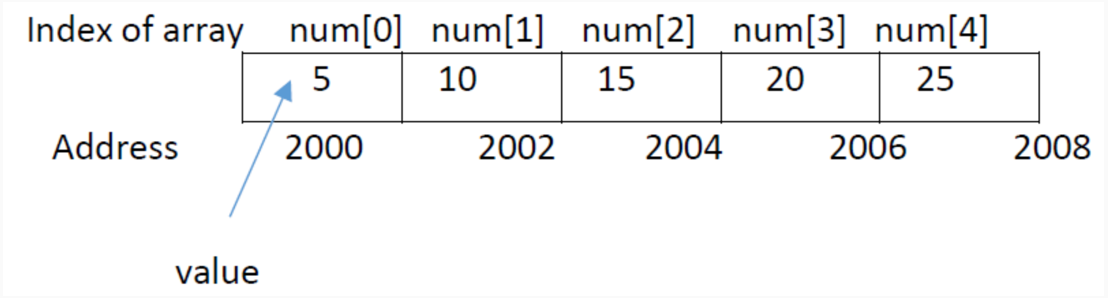
Declaration:

data_type array_name[size];

- eg. int num[10];
- Here, int is a data type and num is the name of the array and 10 is the size of the array. It means num can only contain 10 elements of int type.



Types of Array



<u>**Declaration:**</u>

data_type array_name[size];

- eg. int num[10];
- Here, int is a data type and num is the name of the array and 10 is the size of the array. It means num can only contain 10 elements of int type.

Types of Array : Initialization of 1D array

Compile time initialization

General form: `data_type array_name[size]={list of values};`

The value in the list are separated by commas.

- Example:
- `int num[5]={10,20,30,40,50};` //integer array initialization
- `float area[5]={33.4,55.6,88.9,77.8,5.5};` //float array initialization

Note:

It is also possible to initialize array without defining its size.

- `int num[]={67,87,56,24,77};`

Types of Array : Initialization of 1D array

Compile time initialization

Example:

```
#include<stdio.h>

#include<conio.h>

int main()

{

int i;

int num[5]={67,87,56,24,77};

for(i=0;i<5;i++)

{

        printf("\t%d",num[i]

);

}

getch();

return 0;

}
```

Output

67 87 56 24 77

Types of Array : Initialization of 1D array

Runtime initialization

- An array can also be initialized at runtime using scanf() function. This Approach is used for initializing large arrays with user specified values.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int num[5];
int i;
printf("\nEnter the array elements");
for(i=0;i<5;i++)
{
                scanf("%d",&num[i]);
}
printf("\nArray elements are");
for(i=0;i<5;i++)
{
                printf("\n%d",num[i]);
}
getch();
return 0;
}
```

Multidimensional Array

Multidimensional Array

- Multidimensional array are those who have two or more than two dimensions.
- Multidimensional array are defined in the same manner as one dimensional arrays, except that separate pair of square brackets is required for each subscript.
- Thus, two dimensional array will require two pair of square brackets, a three dimensional array will require three pairs of square brackets and so on.

General form:

```
        datatype array_name[s1][s2][s3]….[sn]
```

where, si is the size of ith dimension.

Two Dimensional Array (2D array)

A two dimensional array is a collection of similar data items, structured in two dimensions (referred to as rows and columns).

- It is also called array of arrays.
- It is required to manipulate the data in table format or in matrix format which contains rows and columns.

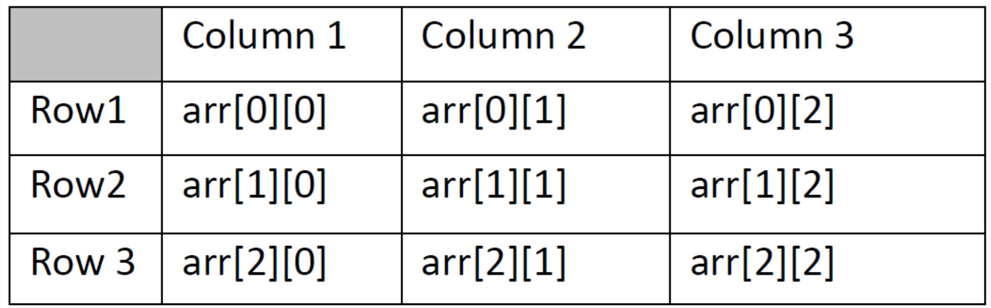
Declaration:

```
        data_type array_name[row_size][column_size];
```

eg. `int arr[3][3];`

Multidimensional Array: Two Dimensional Array (2D array)

It creates two dimensional array to store 9 elements of integer type. There are 3 rows and 3 columns in array matrix.



	Column 1	Column 2	Column 3
Row1	arr[0][0]	arr[0][1]	arr[0][2]
Row2	arr[1][0]	arr[1][1]	arr[1][2]
Row 3	arr[2][0]	arr[2][1]	arr[2][2]

Two Dimensional Array (2D array) : Initialization

Compile time initialization: Two dimensional array may be initialized by following their declaration with a list of values enclosed in braces.

Example: `int arr[3][3]={ {2,4,6},{8,9,12},{15,16,18}};`

These are equivalent to following assignments

```
arr[0][0]= 2           arr[0][1]= 4           arr[0][2]= 6

arr[1][0]= 8           arr[1][1]= 9           arr[1][2]= 12

arr[2][0]=15           arr[2][1]= 16          arr[2][2]=18
```

Types of Array : Initialization of 2D array

Compile time initialization

Example

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i,j;
int arr[3][3]={{12,14,16},{5,7,15},{15,25,45}};
printf("The matrix is\n");
for(i=0;i<3;i++)
{
         for(j=0;j<3;j++)
         {
         printf("%d\t",arr[i][j]);
         }
         printf("\n");
}
getch();
return 0;
}
```

Types of Array :
Initialization of 2D array

Run time initialization

Program to input 3*3 matrix and display it.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i,j;
int arr[3][3];
printf("Enter the elements of matrix\n");
for(i=0;i<3;i++)
            {
            for(j=0;j<3;j++)
            {
            scanf("%d",&arr[i][j]);
            }
}
```

Types of Array : Initialization of 2D array

Run time initialization

```
printf("Elements of matrix are\n");
for(i=0;i<3;i++)
{
            for(j=0;j<3;j++)
            {
            printf("%d\t",arr[i][j]);
            }
            printf("\n");
}
getch();
return 0;
}
```

Some Questions

WAP to input 10 number in an array and display it.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i,num[10];
printf("Enter 10 array elements\n");
for(i=0;i<10;i++)
    {
          scanf("%d",&num[i]);
     }
printf("The array elements are\n");
for(i=0;i<10;i++)
     {
     printf("%d\n",num[i]);
     }
getch();
return 0;
}
```

Some Questions

WAP to input 10 number in an array and display it.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i,num[10];
printf("Enter 10 array elements\n");
for(i=0;i<10;i++)
    {
            scanf("%d",&num[i]);
     }
printf("The array elements are\n");
for(i=0;i<10;i++)
     {
     printf("%d\n",num[i]);
     }
getch();
return 0;
}
```

Some Questions

- **WAP to read n elements in array and display them in reverse order**

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i,num[100],n;
printf("Enter number of array elements \n");
scanf("%d",&n);
printf("Enter %d array elements\n",n);
for(i=0;i<n;i++)
{
            scanf("%d",&num[i]);
}
printf("Array elements in reverse order are\n");
for(i=n-1;i>=0;i--)
{
            printf("%d\n",num[i]);
}
getch();
return 0;
}
```

Some Questions

- WAP to read n numbers using array and find their average.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int n,i;
float num[100],sum=0,avg;
printf("Enter number of array elements \n");
scanf("%d",&n);
printf("Enter %d elements\n",n);
for(i=0;i<n;i++)
{
              scanf("%f",&num[i]);
}
for(i=0;i<n;i++)
{
              sum=sum+num[i];
}
avg=sum/n;
printf("Average=%f\n",avg);
getch();
return 0;
}
```

Some Questions

- WAP to input n numbers in an array and find the sum of all even numbers and count them.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int num[100],i,n,esum=0,ecount=0;
printf("Enter number of array elements\n");
scanf("%d",&n);
printf("Enter %d elements\n",n);
for(i=0;i<n;i++)
{
           scanf("%d",&num[i]);
}
for(i=0;i<n;i++)
{
   if(num[i]%2==0)
   {
   esum=esum+num[i];
   ecount++;
   }
}
printf("Sum of Even numbers=%d\n",esum);
printf("Number of Even numbers=%d",ecount);
getch();
return 0;
}
```

String

String

- Strings are sequence of characters stored in consecutive memory location.
- Each character in string occupies one byte of memory.
- Strings always terminated with null character ‘\0’.

Declaration:

```
char string_name[size];
```

Here, size determines the number of characters in string_name.

Eg. `char name[5];`

Note: *When the compiler assigns a character string to character array, it automatically supplies a null character (‘\0’) at the end of string. Therefore, size should be equal to maximum number of characters in string plus one*

String: Initialization

Compile time initialization

General form:

```
char string_name[size]=”list of character”;
```

eg. `char name[5]=”ram”;`

OR

```
char string_name[size]={list of character};
```

eg. `char name[5]={‘r’,’a’,’m’,’\0’};`

Note: C also permits to initialize a character array without specifying the number of elements. In such cases size of array will determined automatically, based on number of elements initialized.

Eg. `char name[ ]={‘r’,’a’,’m’,’\0’};`

defines, the array string as a four element array

String

Compile time initialization

Example

```
#include<stdio.h>
#include<conio.h>
int main()
{
char greeting[6]={'H','e','l','l','o','\0'};
char name[4]="ram";
printf("Greeting message:%s",greeting);
printf("\nName: %s",name);
getch();
return 0;
}
```

Output:

```
Greeting message:Hello
Name:ram
```

String: Initialization

Runtime initialization

The familiar input function scanf() can be used with %s format specification to read string.

Example:

```
char name[20];

scanf("%s",name);
```

Limitation: Here, string variable takes only single word, It is because when whitespace is encountered the scanf() function terminates

String: Initialization

To overcome this problem the gets() function is used to read a string of text containing whitespaces, until newline character is encountered.

```
#include<stdio.h>
#include<conio.h>
int main()
{
char name[20];
printf("Enter your name:");
gets(name);
printf("Your name is:");
puts(name);
getch();
return 0;
}
```

Output:

```
Enter your name: Hari Bahadur
Your name is: Hari Bahadur
```

String as Array of Characters

String as Array of Characters

- Array of strings means two dimensional array of characters.

Eg. `char name[5][10];`

Here, first dimension tells, how many strings can be stored in array. The second dimension tells the maximum length of each string.

In above declaration, we can store 5 strings, each can store maximum 9 characters. Last 10th space is for null terminator in each string.

Arrays of String

WAP to input name of 5 students and display it.

Example

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i;
char name[5][20];
printf("Enter the name of 5 students\n");
for(i=0;i<5;i++)
{
           gets(name[i]);
}
printf("The names are\n");
for(i=0;i<5;i++)
{
           puts(name[i]);
}
getch();
return 0;
}
```

String Handling Functions

String Handling Functions

- C provides several built-in functions for string manipulation, which are declared in the <string.h> header file. These functions can be used for tasks such as copying, concatenation, comparison, and finding the length of strings.

Some of them are as follows:

1. **strlen()** : Finds the length of a string excluding null character
2. **strcpy():** Copies one string to another including null character
3. strrev(): Reverses all characters in string except null character
4. strcat(): Concatenates two strings. i.e. it appends one string at the end of other
5. strlwr() converts the uppercase string into lowercase
6. strupr() converts the lowercase string into uppercase

String Handling Functions

7. **strcmp():** Compares two strings to find out whether they are same or different. This functions accepts two strings as parameters and returns an integer whose value is

i. Less than 0 if the first string is less than second

ii. Equal to 0 if both are same

iii. Greater than 0 if first string is greater than second

String handling function

Example

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main ()
{
int l ;
char str1[20] = “rajarshi";
char str2[20] = "university";
char str3[20];
l=strlen(str1);
printf("Length of the str1 is %d\n",l);
strcpy(str3,str1);
printf("string copied from str1 to str3 is %s\n",str3 );
strcat(str1,str2);
printf("string after concatenation(s1+s2)is %s\n",str1);
strrev(str3);
printf("Reverse of string str3 is %s\n",str3);
strupr(str3);
printf("Uppercase of string str3 is %s\n",str3);
getch();
return 0;
}
```

String handing functions : strlen()

- This function returns the an integer which denotes the length of a string passed. Length of a string is number of characters present in it, excluding the terminating null character.

Its syntax is :

```
integer_variable =strlen(string);
```

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
char str[20];
int len;
printf("Enter the string:");
gets(str);
len=strlen(str);
printf("The length of a string is
%d",len);
getch();
return 0;
}
```

Output:

```
Enter the string : ramesh
The length of string is 6
```

String handing functions : strcpy()

- This function copies one string to another.
- The function accepts two strings as parameters and copies the second string character by character into first one upto including the null character of the second string.

The syntax is :

```
strcpy(destination_string,source_string);
```

i.e. strcpy(str2,str1) means the content of str1 is copied to str2

```
#include<stdio.h>

#include<conio.h>

#include<string.h>

int main()

{

char str1[20]="ramesh",str2[20];

strcpy(str2,str1);

printf("The copied string is

%s",str2);

getch();

return 0;

}
```

Output:
The copied string is ramesh

String handing functions : strcat()

- This function concatenates two strings i.e it appends one string at the end of another.
- This function accepts two strings as parameters and stores the contents of second string at the end of first. Its syntax is :

```
strcat (string1,string2);
```

ie.string1=string1+string2

```
#include<stdio.h>

#include<conio.h>

#include<string.h>

int main()

{

char

str1[20]="rajarshi",str2[20]="university";

strcat(str1,str2);

printf("The concatenated string is %s",str1);

getch();

return 0;

}
```

Output:

```
The concatenated string is rajarshiuniversity
```

String handing functions : strupr()

- This function converts the lowercase string into uppercase.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
char str[20]="hello";
strupr(str);
printf("The uppercase of given string is %s",str);
getch();
return 0;
}
```

Output:

```
The uppercase of given string is HELLO
```

String handing functions : strlwr()

- This function converts the uppercase string into lowercase.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
char str[20]="HELLO";
strlwr(str);
printf("The lowercase of given string is %s",str);
getch();
return 0;
}
```

Output:

```
The lowercase of given string is hello
```

String handing functions : strrev()

This function is used to reverse all characters in a string except null character at the end of string. The reverse of string “abc” is “cba”.
It’s syntax is strrev(string);

For example:
strrev(s) means it reverses the characters in string s and stores reversed string in s.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
char str[20]="hello";
strrev(str);
printf("Reversed string is %s\n",str);
getch();
return 0;
}
```

Output:

```
Reversed string is olleh
```

String handing functions : strcmp()

This function accepts two strings as parameters and returns an integer whose value is

i)less than 0 if the first string is less than second

ii)equal to 0 if both are same

iii)greater than 0 if first string is greater than second

The two strings are compared character by character until there is a mismatch or end of one string is reached. Whenever two characters in two string differ, the string which has the character with higher ASCII value is greater.

For example, consider two strings "ram" and "rajesh".The first two characters are same but the third character in string ram and that is in rajesh are different. Since ASCII value of character m in string is ram is greater than that of j in string rajesh, the string ram is greater than rajesh.

String handing functions : strcmp()

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
char str1[20],str2[20];
int diff;
printf("Enter the first string:");
gets(str1);
printf("Enter the second string:");
gets(str2);
diff=strcmp(str1,str2);
if(diff>0)
{
    printf("%s is greater than %s",str1,str2);
}
else if(diff<0)
{
    printf("%s is greater than %s ",str2,str1);
}
else
{
     printf("Both strings are same");
}
getch();
return 0;
}
```

Output:

```
Enter first string: ram
Enter second string: rajesh
ram is greater than rajesh
```

Passing Array to Function

Passing Array to Function

Passing one dimensional Array to function

In C, arrays are automatically passed by reference to a function. The name of the array represents the address of its first element. By passing the array name, we are in fact, passing the address of the array to the called function. The array in the called function now refers to same array stored in the memory. Therefore any changes in the array in the called function will be reflected in the original array.

- The corresponding formal argument in function definition is written in the same manner, though it must be declared as an array.
- When declaring a one dimensional array as a formal argument, the array name is written with a pair of empty square brackets. The size of the array is not specified within the formal argument declaration.
- To pass an array as an argument to function, the array name must be specified, without brackets or subscripts, and the size of an array as arguments.

Passing Array to Function

Syntax for function declaration that receives array as argument

```
return_type function_name(data_type array_name[ ] );
```

Syntax to pass array to the function.

```
function_name(arrayname);
```

Note: We cannot pass a whole array by value as we did in the case of ordinary variables

Passing Array to Function

Program to illustrate passing array to a function one element at a time.

```
#include<stdio.h>
#include<conio.h>
void display(int x);
int main()
{
    int i;
    int a[5]={5,10,15,20,25};
    for(i=0;i<5;i++)
    {
        display(a[i]);
    }
    getch();
    return 0;
}
```

```
void display(int x)
{
    printf("%d\t",x);
}
```

Passing Array to Function

Write a program to pass one dimensional array to a function and display that array in that called function.

```
#include<stdio.h>
#include<conio.h>
void display(int x[],int n);
int main()
{
   int i;
   int a[5]={5,10,15,20,25};
   display(a,5);
   getch();
   return 0;
}

void display(int x[],int n)
{
    int i;
    for(i=0;i<n;i++)
    {
      printf("%d\t",x[i]);
    }
}
```

Passing Array to Function

Write a program to read n number in an array and sort them in ascending order using function.

```
#include<stdio.h>
#include<conio.h>
void sort(int x[],int n);
void disp(int d[],int m);
int main()
{
 int a[100],n,i;
 printf("Enter how many elements\n");
 scanf("%d",&n);
 printf("Enter the array elements\n");
 for(i=0;i<n;i++)
 {
   scanf("%d",&a[i]);
 }
 printf("\nArray elements before sorting are\n");
 disp(a,n);
 sort(a,n);
 printf("\nArray elements after sorting are\n");
 disp(a,n);
 getch();
 return 0;
}
```

Passing Array to Function

Write a program to read n number in an array and sort them in ascending order using function.

```
void sort(int x[],int n)
{
    int i,j,temp;
    for(i=0;i<n-1;i++)
   {
       for(j=0;j<n-1-i;j++)
            {
                if(x[j]>x[j+1])
                    {
                     temp=x[j];
                      x[j]=x[j+1];
                      x[j+1]=temp;
                     }
            }
     }
}
```

```
void disp(int d[],int m)
{
            int i;
            for(i=0;i<m;i++)
            {
            printf("%d\n",d[i]) ;
            }
}
```

Some Questions to Work On

- WAP to read n numbers in an array find the largest number among them.
- WAP to input n number in an array and sort them in Ascending order
- Write a program to add two 3X3 matrix. Display the sum stored in third matrix
- WAP to check whether the given number is present in a array or not.
- Write a program to enter values in 3*3 order matrix and compute the sum of odd elements.

Passing 2D Array to Function

Passing two dimensional Array to function

It is similar as in 1 dimensional array, when two dimensional array are passed as a parameter, the base address of the actual array is sent to function.

- Any change made to element inside function will carry over to the original location of array that is passed to function.
- The size of all dimensions except the first must included in the function prototype (declaration) and in function definition.

Syntax: `return_type function_name(array_name, size, size);`

Eg. `void display(int[][10],int,int);`

is a valid declaration.

Passing 2D Array to Function

Write a program to show addition of two m*n order matrix using function.

```
#include<stdio.h>
#include<conio.h>
void input(int a[][20],int x,int y);
void disp(int d[][20],int m,int n);
void addition(int a1[][20],int a2[][20],int p,int q);
int i,j;
int main()
{
            int matrix1[20][20],matrix2[20][20],r,c;
            printf("Enter the row and column size of matrix\n");
            scanf("%d%d",&r,&c);
            printf("Enter the first matrix");
            input(matrix1,r,c);
            printf("Enter the second matrix");
            input(matrix2,r,c);
            printf("The first matrix is\n");
            disp(matrix1,r,c);
            printf("The second matrix is\n");
            disp(matrix2,r,c);
            addition(matrix1,matrix2,r,c);
            getch();
            return 0;
}
```

Passing 2D Array to Function

Write a program to show addition of two m*n order matrix using function.

```
void input(int a[][20],int x,int y)
{
            for(i=0;i<x;i++)
            {
                        for(j=0;j<y;j++)
                        {
                           scanf("%d",&a[i][j]);
                        }
            }
}

void disp(int d[][20],int m,int n)
{
            for(i=0;i<m;i++)
            {
                        for(j=0;j<n;j++)
                        {
                        printf("%d\t",d[i][j]);
                        }
            printf("\n");
}
}
```

Passing 2D Array to Function

Write a program to show addition of two m*n order matrix using function.

```
void addition(int a1[][20],int a2[][20],int p,int q)
{
        int sum[20][20];
        for(i=0;i<p;i++)
        {
                for(j=0;j<q;j++)
                {
                sum[i][j]=a1[i][j]+a2[i][j] ;
                }
        }
        printf("Resultant matrix is\n");
        disp(sum,p,q);
}
```

Passing Strings to Function

Passing Strings to Function

In C, we can pass a string to a function using a character array or a pointer to a character. Since strings in C are stored as arrays of characters and are null-terminated (\0), passing a string essentially means passing a pointer to its first character.

Two approaches:

1. Passing String as a Character Array
2. Passing string as Pointer

Passing String as a Character Array

- A string in C is represented as a character array (char array[]).
- When we pass an array to a function, we are actually passing the base address of the array.

Passing Strings to Function

```
#include<stdio.h>
void printString(char str[]);

int main() {
    char myString[] = "Hello, C!";

    // Passing the string to function
    printString(myString);

    return 0;
}

// Function to print a string
void printString(char str[]) {
    printf("The string is: %s\n", str);
}
```

Concept of Pointer, Pointer address, dereference, declaration, assignment, initialization, Pointer Arithmetic, Array and Pointer

Concept of Pointer

- Pointer is a variable that stores/points the address of another variable. Pointer variable can only contains the address of a variable of the same data type.

eg.

```
int *ptr; //pointer declaration
int a=5;
ptr=&a; //address of variable a is assigning to pointer variable ptr
```

Here `ptr` is a pointer variable that only contains the address of integer data type.

Pointer Declaration

- Pointer variable is declared with an asterisk (*) operator before a variable name. This operator is called pointer/indirection or dereference operator.

Syntax: `data_type *pointer_variable_name;`

Data type of the pointer must be same as the data type of the variable to which the pointer variable is pointing.

eg. `int *ptr;`

Declares the variable `ptr` which is a pointer variable that points to an integer data type.

Initialization of Pointers

- Pointer initialization is the process of assigning address of variable to a pointer variable .In C programming language ampersand (&) operator is used to determine the address of variable. The & (immediately preceding a variable name) returns the address of variable associated with it.
- Pointer variable always points to variables of same data type.

```
int main()
{
int a=10;
int *ptr; //pointer declaration
ptr=&a; //pointer initialization
}
```

Here, pointer variable ptr of integer type is pointing the address of integer variable a.

Initialization of Pointers

Once, address of variable is assigned to pointer, then to access the value of variable, pointer is dereferenced using indirection/dereference operator (*).

Example

```
#include<stdio.h>
#include<conio.h>
int main()
{
int a=5;
int *ptr; //declaration of pointer variable
ptr=&a; //initializing the pointer
printf("\n%d",a); //prints the value of a
printf("\n%d",*ptr); //prints the value of a
printf("\n%d",*(&a)); //prints the value of a
printf("\n%u",&a); //prints the address of a
printf("\n%u",ptr); //prints the address of a
printf("\n%u",&ptr); //prints the address of pointer variable ptr
getch();
return 0;
}
```

Some Definitions:

- **Bad Pointer:** When a pointer variable is declared and if any valid address is not assigned to it then pointer is known as bad pointer.
- A dereference operation on a bad pointer is a serious runtime error. Each pointer must be assigned a valid address before it can support dereference operations. Before that pointer is bad and must not be used.
- It is always a best practice to initialize a pointer NULL values, when they are declared and check for whether the pointer is NULL pointer when using the pointer.

Some Definitions:

- **Void pointer** is a special type of pointer that can point any data type (int or float or char or double).

Declaration: `void *pointer_name;`

Example:

```
#include<stdio.h>
#include<conio.h>
int main()
{
int a=10;
float b=4.5;
void *ptr;
ptr=&a;
printf("a=%d",*(int*)ptr);
ptr=&b;
printf("b=%f",*(float*)ptr);
getch();
return 0;
}
```

Output:

```
a=10
b=4.500000
```

Some Definitions: Null Pointer

- A Null pointer is a pointer which is pointing to nothing.
- Pointer which is initialized with NULL value is considered as NULL pointer.
- We can define a null pointer using a predefined constant NULL, which is defined in header files stdio.h , stdef.h, stdlib.h.

__**Some of the most common use cases for NULL are**__

i. To initialize a pointer variable when that pointer variable isn't assigned any valid memory address yet.

```
int * ptr = NULL;
```

ii. To check for null pointer before accessing any pointer variable. By doing so, we can perform error handling in pointer related code

e.g. dereference pointer variable only if it's not NULL.

```
 if(ptr != NULL) /*We could use if (ptr) as well*/
{
/*Some code*/
}
else
{
/*Some code*/
}
```

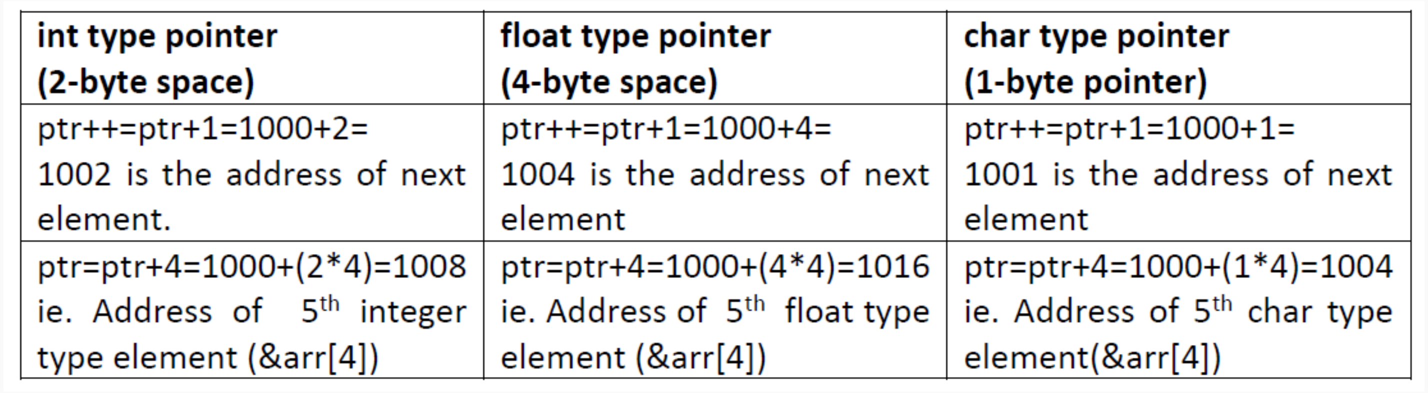
Pointer Arithmetic

Pointer Arithmetic

As pointer holds the memory address of variable, some arithmetic operations can be performed with pointers. They are as follows.

- Increment
- Decrement
- Addition
- Subtraction

If arithmetic operations done values will be incremented or decremented as per data type chosen. Let `ptr=&arr[0]=1000` i.e Base address of array.

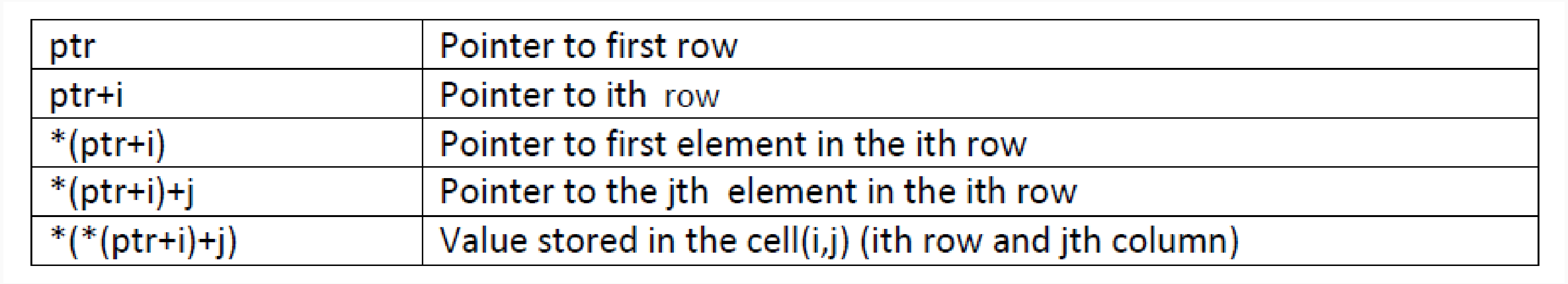


int type pointer (2-byte space)	**float type pointer (4-byte space)**	**char type pointer (1-byte pointer)**
ptr++=ptr+1=1000+2= 1002 is the address of next element.	ptr++=ptr+1=1000+4= 1004 is the address of next element	ptr++=ptr+1=1000+1= 1001 is the address of next element
ptr=ptr+4=1000+(2*4)=1008 ie. Address of 5th integer type element (&arr[4])	ptr=ptr+4=1000+(4*4)=1016 ie. Address of 5th float type element (&arr[4])	ptr=ptr+4=1000+(1*4)=1004 ie. Address of 5th char type element(&arr[4])

Note: Similar operation can be done for decrement

Pointer Arithmetic

- Note:



ptr	Pointer to first row
ptr+i	Pointer to ith row
*(ptr+i)	Pointer to first element in the ith row
*(ptr+i)+j	Pointer to the jth element in the ith row
((ptr+i)+j)	Value stored in the cell(i,j) (ith row and jth column)

Pointer Arithmetic

- Example for pointer increment/decrement

```
#include<stdio.h>
#include<conio.h>
int main()
{
 int arr[5] = {10,20,30,40,50};
 int *ptr1,*ptr2;
 ptr1=&arr[0];
 ptr1++;
 printf("\nvalue %d has address %u",*ptr1,ptr1);
 ptr2=&arr[4];
 ptr2-- ;
 printf("\nvalue %d has address %u",*ptr2,ptr2);
 getch();
 return 0;
}
```

Output:

```
value 20 has address 7274120
value 40 has address 7274128
```

Pointer Arithmetic

- **Example for pointer addition/subtraction with integer constant**

Output:

```
value 50 has address 6487588
value 10 has address 6487572
```

```
#include<stdio.h>
#include<conio.h>
int main()
{
int arr[5] = {10,20,30,40,50};
int *ptr1,*ptr2;
ptr1=&arr[0];
ptr1=ptr1+4;
printf("\nvalue %d has address %u",*ptr1,ptr1); //points to
5th element

ptr2=&arr[4];
ptr2=ptr2-4 ;

printf("\nvalue %d has address %u",*ptr2,ptr2); //points to
1st element
getch();
return 0;
}
```

Pointer Arithmetic

- Other operations

1) A pointer variable can be assigned the address of an ordinary variable.

```
int main()
{
int a=5;
int *ptr;
ptr=&a;
printf("value of a=%d",a);
printf("Address of a=%u",ptr);
return 0;
}
```

Pointer Arithmetic

- **Other operations**

2) Content of one pointer variable can be assigned to other pointer variable provided they point to same data type.

```
int main()
{
int arr[5]={1,2,3,4,5};
int *ptr1,*ptr2;
ptr1=&arr[0];
ptr2=ptr1; //assign element of ptr1 to ptr2
printf("ptr1 contains %u and ptr2 contains %u",ptr1,ptr2);
return 0;
}
```

Pointer Arithmetic

- **Other operations**

3) One pointer variable can be subtracted from another provided that both variables points to the element of same array.

```
int main()
{
int arr[5]={1,2,3,4,5};
int *ptr1,*ptr2;
ptr1=arr;
ptr2=&arr[4];
printf("Difference of two pointer =%d",ptr2-ptr1);
return 0;
}
```

Pointer Arithmetic

- **Other operations**

4) Two pointers variables can be compared provided both pointers points to object of same data type.

```
int main()
{
int arr[5]={1,2,3,4,5};
int *ptr1,*ptr2;
ptr1=&arr[0];
ptr2=&arr[4];
if(ptr2>ptr1)
printf("ptr2 is far from ptr1");
else
printf("ptr1 is far from ptr2");
return 0;
}
```

Pointer Arithmetic

Invalid pointer operations

- Addition of two pointer (ptr1+ptr2)
- Multiplication of two pointer(ptr1*ptr2)
- Addition or subtraction of float or double data type to or from a pointer
- Multiplication of pointer with constant (ptr1*5)
- Division of two pointer or with constant (ptr1/5)

Passing pointer to function

Passing pointer to function

- Pointers can be passed to functions as arguments.
- When we pass a pointer, the address of the variable is sent, not the value.
- This allows the function to access and modify the original variable.
- Any changes made through the pointer affect the actual variable.
- This method is known as Call by Reference.

It is useful when:

✓ We want the function to change original values.

✓ We want to return multiple values from a function.

Passing pointer to function

- Program to illustrate passing pointer to function.

```
#include<stdio.h>
#include<conio.h>
void addGraceMarks(int *m);
int main()
{
int marks;
printf("Enter the actual marks\n");
scanf("%d",&marks);
addGraceMarks(&marks);
printf("The final marks is:%d",marks);
getch();
return 0;
}
void addGraceMarks(int *m)
{
   *m=*m+10;
}
```

Passing pointer to function

- Program to convert uppercase letter into lower and vice versa passing pointer to function.

```
#include<stdio.h>
#include<conio.h>
void conversion(char *);
int main()
{
char ch;
printf("Enter the character\n");
scanf("%c",&ch);
conversion(&ch);
printf("The corresponding character is:%c",ch);
getch();
return 0;
}

void conversion(char *c)
{
   if(*c>='a'&&*c<=‘z’)
      {
      *c=*c-32;
      }
   else if (*c>='A'&&*c<=‘Z’)
      {
      *c=*c+32;
      }
}
```

Returning multiple values from function

Returning multiple values from function

- Can function return more than one value? Justify your answer with examples,
- Does function return single or multiple value? When and how a function will return single or multiple value? Illustrate with examples.
- How can function return multiple values? Explain with example?

- Normally, functions in C cannot return more than one value when arguments are passed by value.
- By using pointers, we can effectively return multiple values from a function.
- This is done by passing the address of variables to the function.
- Inside the function, we can modify the values at those addresses using pointers.
- These changes are reflected in the original variables (actual arguments).
- This technique is known as Call by Reference.
- We can pass any number of variables as pointers to return multiple results

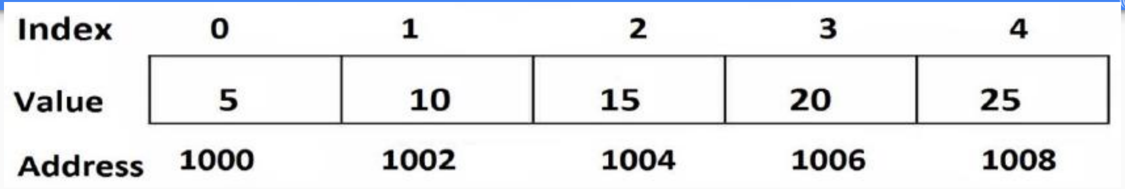
Advantages of Pointer

Advantages of Pointer

- Pointers enable dynamic memory allocation (DMA), allowing efficient use of memory at runtime.
- Pointers simplify array manipulation, enabling fast traversal and element access through arithmetic.
- Pointers allow efficient handling of structures, especially when passing large data to functions without copying.
- Pointers enable function arguments to be passed by reference, allowing in-place modification of variables.
- Pointers provide flexibility and efficiency by directly accessing and managing memory addresses.

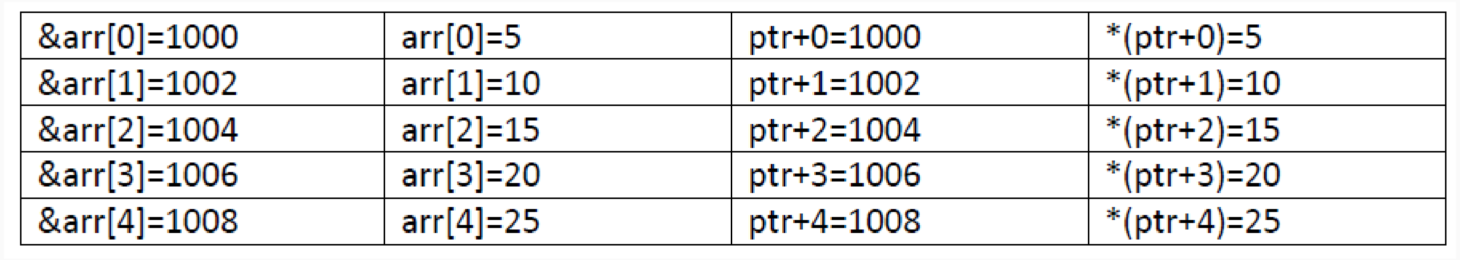
Array and pointer

1D array with pointer



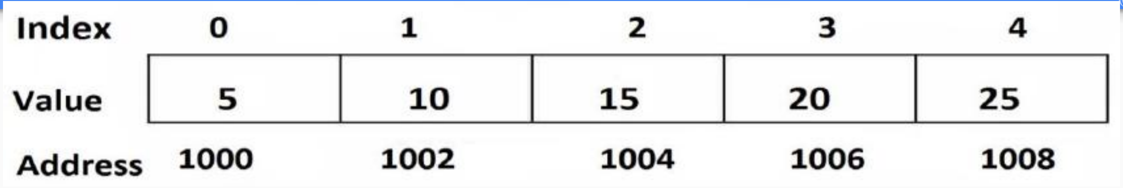
Index	0	1	2	3	4
Value	5	10	15	20	25
Address	1000	1002	1004	1006	1008

```
int arr[5]={5,10,15,20,25};
int *ptr, i;
ptr=arr; //similar to ptr=&arr[0]
By assuming array starts at location 1000
```



&arr[0]=1000	arr[0]=5	ptr+0=1000	*(ptr+0)=5
&arr[1]=1002	arr[1]=10	ptr+1=1002	*(ptr+1)=10
&arr[2]=1004	arr[2]=15	ptr+2=1004	*(ptr+2)=15
&arr[3]=1006	arr[3]=20	ptr+3=1006	*(ptr+3)=20
&arr[4]=1008	arr[4]=25	ptr+4=1008	*(ptr+4)=25

1D array with pointer



Index	0	1	2	3	4
Value	5	10	15	20	25
Address	1000	1002	1004	1006	1008

Accessing value	Accessing Address
printf("%d",*(ptr+i)); //displays the arr[i] value	printf("%u",(ptr+i)); //displays the address of arr[i]
printf("%d",*ptr); //display the arr[0] value	printf("%u", ptr); //display the address of arr[i]
printf("%d",*(arr+i); //displays the arr[i] value	printf("%u",(arr+i)); //displays the address of arr[i]

1D array with pointer

WAP to input n number in array and display it using pointer.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int arr[100],i,n,*ptr;
ptr=arr;
printf("Enter the number of elements you want
to enter");
scanf("%d",&n);

printf("\nEnter %d elements",n);
for(i=0;i<n;i++)
{
   scanf("%d",(ptr+i));
}

printf("Entered elements are\n");
for(i=0;i<n;i++)
{
   printf("%d",*(ptr+i));
}
getch();
return 0;
}
```

1D array with pointer

WAP to input n number in array and find sum of all elements using pointer

```
#include<stdio.h>
#include<conio.h>
int main()
{
int arr[100],n,i,*ptr,sum=0;
ptr=arr;
printf("Enter number of elements you want to
enter");
scanf("%d",&n);

printf("Enter %d elements",n) ;
for(i=0;i<n;i++)
{
   scanf("%d",(ptr+i));
}

for(i=0;i<n;i++)
{
   sum=sum+*(ptr+i);
}
printf("The sum of all elements is %d",sum);
getch();
return 0;
}
```

1D array with pointer

Write a program to sort n integer values in an array using pointer.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int arr[100],n,i,j,temp,*ptr;
ptr=arr;
printf("Enter number of elements you want to
enter");
scanf("%d",&n);
printf("Enter %d elements",n) ;
for(i=0;i<n;i++)
{
   scanf("%d",(ptr+i));
}

for(i=0;i<n-1;i++)
  {
    for(j=0;j<n-i-1;j++)
      {
        if(*(ptr+j)>*((ptr+j)+1))
         {
          temp=*(ptr+j);
          *(ptr+j)=*((ptr+j)+1);
          *((ptr+j)+1)=temp;
         }
      }
}
```

1D array with pointer

Write a program to sort n integer values in an array using pointer.

```
printf("\nThe sorted array are");
for(i=0;i<n;i++)
{
   printf("\n%d",*(ptr+i));
}
getch();
return 0;
}
```

1D array with pointer

Write a program using pointers to read in an array of integers and prints its elements in reverse order.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int arr[100],i,n,*ptr;
ptr=arr;
printf("Enter number of elements you want to enter");
scanf("%d",&n);

printf("Enter %d elements",n) ;
for(i=0;i<n;i++)
{
  scanf("%d",(ptr+i));
}

printf("\nThe array elements are");
for(i=0;i<n;i++)
{
  printf("\n%d",*(ptr+i));
}

printf("\nThe array elements in reverse order are");
for(i=n-1;i>=0;i--)
{
  printf("\n%d",*(ptr+i));
}
getch();
return 0;
}
```

Array of pointers

Array of pointers

- Array of pointers is a collection of address.
- The address present in the array of pointers can be address of variable or address of array element.

Array of pointers can be declared as:

```
datatype *pointer_name[size];
```

eg.int *ptr[4];

This statement declares an array of 4 pointers, each of which points to an integer value. The first pointer is called ptr[0] ,the second is ptr[1] ,third is ptr[2] and fourth is ptr[3].

Array of pointers

Example

```
#include<stdio.h>
#include<conio.h>
int main()
{
   int a=5,b=6,c=7,d=8,i;
   int *ptr[4];
   ptr[0]=&a;
   ptr[1]=&b;
   ptr[2]=&c;
   ptr[3]=&d;

for(i=0;i<4;i++)
{
  printf("%d",*ptr[i]);
}
getch();
return 0;
}
```

Array of pointers

WAP to input 5 integer numbers and add all elements using array of pointer.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int *ptr[5];
int arr[5],i,sum=0;

for(i=0;i<5;i++)
{
   ptr[i]=&arr[i];
}
printf("Enter 5 elements\n");

for(i=0;i<5;i++)
{
   scanf("%d",&arr[i]);
}

for(i=0;i<5;i++)
{
   sum=sum+*ptr[i];
}
printf("sum=%d",sum);
getch();
return 0;
}
```

2D array using pointer

2D array using pointer

Syntax:

```
datatype(*ptr_variable)[size2];
```

Example:

Let us suppose arr is a 2-D integer array having 3 rows and 4 columns, we can declare arr as

```
int(*ptr)[4];
```

```
#include<stdio.h>
#include<conio.h>
int main()
{
int arr[3][4] ={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
int i, j;
int (*ptr)[4];
ptr = arr; //&arr[0][0]
for(i = 0; i < 3; i++)
{
   for(j = 0; j < 4; j++)
   {
    printf("\narr[%d][%d]=%d", i, j, *( *(ptr + i) + j) );
   }
printf("\n");
}
getch();
return 0;
}
```

Note: As array name holds the base address (i.e. address of first element of array).Then, if arr is a 2D array we can access any element arr[i][j] of the array using the pointer expression.
((arr+i)+j)

2D array using pointer

WAP to add two matrix of order m*n by using the concept of pointer.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int a[20][20],b[20][20],c[20][20],i,j,m,n;
printf("Enter the value of m & n:");
scanf("%d%d",&m,&n);
printf("Enter the first matrix:");
for(i=0;i<m;i++)
{
   for(j=0;j<n;j++)
   {
     scanf("%d",(*(a+i)+j));
   }
}

printf("\nEnter second matrix:");
for(i=0;i<m;i++)
{
   for(j=0;j<n;j++)
   {
     scanf("%d",(*(b+i)+j));
   }
}
```

2D array using pointer

WAP to add two matrix of order m*n by using the concept of pointer.

```
printf("\nAddition of two Matrix:\n");
for(i=0;i<m;i++)
{
  for(j=0;j<n;j++)
  {
    *(*(c+i)+j)=*(*(a+i)+j)+ *(*(b+i)+j);
    printf("%d ",*(*(c+i)+j));
  }
  printf("\n");
}
getch();
return 0;
}
```

Array Vs Pointer

Question: Differences between array and pointer.

Arrays	Pointers
An array is a collection of elements of the same type stored in contiguous memory locations.	A pointer is a variable that stores the memory address of another variable.
Elements are accessed using the array index, e.g., array[index]	A pointer accesses elements it points to by dereferencing, e.g., *pointer
The size of an array is fixed and defined at the time of declaration	A pointer is of a fixed size (depends on the architecture, usually 4 or 8 bytes), but it can point to blocks of memory of any size.
Directly access data.	Provides an extra level of indirection to access data.
Limited to index manipulation within the array bounds.	Pointers can be reassigned to point to different memory addresses.
Declared using the syntax: *data_type name[size];*	Declared using the syntax: *data_type *name;*

Dynamic Memory Allocation

Dynamic Memory Allocation

- The process of allocating and releasing memory at runtime is known as Dynamic memory allocation.
- Using DMA memory space is reserved during program execution and release space when no longer required.

Functions used in Dynamic Memory allocation

- There are four library functions: malloc(),calloc(), free() and realloc() are for dynamic memory management.

Returning multiple values from function

- Example:

```
#include<stdio.h>
#include<conio.h>
void areaperi(int r,float *area,float *peri);
int main()
{
int rad;
float a,p;
printf("Enter the radious\n");
scanf("%d",&rad);
areaperi(rad,&a,&p);
printf("Area of circle=%f",a);
printf("Perimeter of circle=%f",p);
getch();
return 0;
}

void areaperi(int r,float *area,float *peri)
{
*area=3.14*r*r;
*peri=2*3.14*r;
}
```

Returning multiple values from function

Example:

```
#include<stdio.h>
#include<conio.h>
void areaperi(int r,float *area,float *peri);
int main()
{
int rad;
float a,p;
printf("Enter the radious\n");
scanf("%d",&rad);
areaperi(rad,&a,&p);
printf("Area of circle=%f",a);
printf("Perimeter of circle=%f",p);
getch();
return 0;
}

void areaperi(int r,float *area,float *peri)
{
  *area=3.14*r*r;
  *peri=2*3.14*r;
}
```

Some Questions to Work On

WAP to store Fibonacci series of 10 terms in array and display them.

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

int main()
{
int fib[10],i;
fib[0]=0;
fib[1]=1;
printf("The fibonacci series is \n");
printf("%d\t%d\t",fib[0],fib[1]);
for(i=2;i<=9;i++)
{
  fib[i]=fib[i-1]+fib[i-2];
}

for(i=2;i<=9;i++)
{
  printf("%d\t",fib[i]);
}

getch();
return 0;
}
```

Some Questions to Work On

WAP to read an array of length n. Display the total count for positive numbers, negative numbers and zeros separately.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i,num[100],n,pcount=0,ncount=0,zcount=0;
printf("Enter the number of array elements\n");
scanf("%d",&n);
printf("Enter %d elements\n",n);
for(i=0;i<n;i++)
{
     scanf("%d",&num[i]);
}
for(i=0;i<n;i++)
{
     if(num[i]>0)
     {
        pcount++;
     }
     else if(num[i]<0)
     {
        ncount++;
     }
     else
     {
        zcount++;
     }
}
printf("Number of postive numbers=%d",pcount);
printf("\nNumber of Negative numbers=%d",ncount);
printf("\nNumber of Zeros=%d",zcount);
getch();
return 0;
}
```

Some Questions to Work On

WAP to store n numbers in an array and display prime numbers stored in array and calculate the sum of those prime numbers.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int a[100],flag,n,i,j,sum=0;
printf("Enter the no. of elements:");
scanf("%d",&n);
printf("Enter %d array elements\n",n);
for(i=0;i<n;i++)
{
     scanf("%d",&a[i]);
}
printf("\nprime numbers are:\n");
for(i=0;i<n;i++)
{
   flag=0;
   for(j=2;j<a[i];j++)
   {
      if(a[i]%j==0)
      {
      flag=1;
      break;
      }
   }

if(flag==0)
{
   printf("%d\t",a[i]);
   sum=sum+a[i]
}
}
printf("\nThe sum of prime no. is %d",sum);
getch();
}
```

Some Questions to Work On

WAP to check whether the given number is present in a array or not.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int a[100],i,n,num,flag=0;
printf("Enter the number of array elements\n");
scanf("%d",&n);
printf("Enter %d elements\n",n);
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
printf("\nEnter the element that you want to search \n");
scanf("%d",&num);
for(i=0;i<n;i++)
{
     if(num==a[i])
     {
        flag=1;
        break;
     }
}
if(flag==1)
{
     printf("your number is found");
}
else
{
     printf("your number is not found");
}
getch();
return 0;
}
```

Some Questions to Work On

Write a program to enter values in 3*3 order matrix and compute the sum of odd elements.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int arr[3][3],i,j,osum=0;
printf("Enter the elements of matrix\n");
for(i=0;i<3;i++)
{
        for(j=0;j<3;j++)
        {
              scanf("%d",&arr[i][j]);
        }
}
for(i=0;i<3;i++)
{
        for(j=0;j<3;j++)
        {
                if(arr[i][j]%2!=0)
               {
                     osum=osum+arr[i][j];
               }
       }
}
printf("Sum of odd elements=%d",osum);
getch();
return 0;
}
```

Some Questions to Work On

WAP to input m*n matrix and find sum of each row.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int arr[20][20],i,j,m,n,sum;
printf("Enter the order of matrix\n");
scanf("%d%d",&m,&n);
printf("Enter %d elements of
matrix\n",m*n);
for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)
    {
       scanf("%d",&arr[i][j]);
    }
}
for(i=0;i<m;i++)
{
    sum=0;
    for(j=0;j<n;j++)
    {
       sum=sum+arr[i][j];
    }
    printf("sum of %d row is %d\n",i+1,sum);
}
getch();
return 0;
}
```

Some Questions to Work On

WAP to read two 3*3 matrix and multiply them.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int
a[3][3],b[3][3],mul[3][3],i,j,k;
printf("Enter 9 elements of first
matrix\n");
for(i=0;i<3;i++)
{
      for(j=0;j<3;j++)
      {
        scanf("%d",&a[i][j]);
      }
}

printf("Enter 9 elements of
second matrix\n");
for(i=0;i<3;i++)
{
      for(j=0;j<3;j++)
      {
          scanf("%d",&b[i][j]);
      }
}
for(i=0;i<3;i++)
{
      for(j=0;j<3;j++)
      {
          mul[i][j]=0;
          for(k=0;k<3;k++)
          {

mul[i][j]=mul[i][j]+a[i][k]*b[k][j];
          }
}
}

printf("Muliplication of matrix is
:\n");
for(i=0;i<3;i++)
{
        for(j=0;j<3;j++)
        {
           printf("%d\t",mul[i][j]);
         }
printf("\n");
}
getch();
return 0;
}
```

Some Questions to Work On

WAP to concatenate two strings inputted through keyboard. The result of concatenation should be displayed after copied on third variable.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
char str1[20],str2[20],str3[40];
printf("Enter the first string\n");
gets(str1);
printf("Enter the second string\n");
gets(str2);
strcpy(str3,strcat(str1,str2));
puts(str3);
getch();
return 0;
}
```

Some Questions to Work On

Write a program to check whether the given string is palindrome or not. (Palindrome is a word which reads same from left to right and right to left.eg LIRIL,MADAM etc.)

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

int main()
{
char str1[20],str2[20];
printf("Enter the string\n");
gets(str1);
strcpy(str2,str1);
strrev(str2);
if(strcmp(str1,str2)==0)
{
   printf("String is palindrome");
}
else
{
   printf("String is not palindrome");
}
getch();
return 0;
}
```

Some Questions to Work On

WAP to sort n students name in alphabetical order

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
int i,j,n;
char name[100][20],temp[20];
printf("Enter the number of students\n");
scanf("%d",&n);
printf("Enter the name of %d students:\n",n);
for(i=0;i<n;i++)
{
    gets(name[i]);
}
for(i=0;i<n-1;i++)
{
    for(j=0;j<n-1-i;j++)
    {
       if(strcmp(name[j],name[j+1])>0)
       {
         strcpy(temp,name[j]);
         strcpy(name[j],name[j+1]);
         strcpy(name[j+1],temp);
       }
     }
}
printf("Name of students in alphabetical order are\n");
for(i=0;i<n;i++)
{
     puts(name[i]);
}
getch();
return 0;
}
```

Questions:

1. What is a multi-dimensional array? Explain with the declaration and initialization of a 2D array. Write a program to add two 2D arrays.
2. How are arrays passed to functions in C? Write a program to pass an array to a function and display its elements.
3. Explain the concept of pointers in C. How are pointers declared and initialized? Write a program to demonstrate pointer declaration and initialization.
4. What are arithmetic operations on pointers? Describe different pointer arithmetic operations with examples.
5. How are pointers used with arrays? Write a program to demonstrate accessing array elements using pointers. Explain the output.
6. Discuss how pointers are used with strings in C. Write a program to print a string using a pointer.
7. Explain how pointers are passed to and returned from functions. Write a program to swap two numbers using pointers and functions.
8. What is a pointer to a pointer? Explain with the help of a program.
9. Explain the use of void pointers in C. Why are they called generic pointers? Write a simple program to illustrate their use.
10. Explain the concept of arrays in C. How are arrays declared and initialized? Write a program to find the sum of all elements in a 1D array.
11. Explain how to pass a string to a function and manipulate it. Write a program to convert a string to uppercase using a function.

THANK YOU

Any Queries ?