



ABES Engineering College, Ghaziabad.
Affiliated to Dr. A.P.J Abdul Kalam Technical University, Lucknow.
Department of CSE-Data Science

Title	Lecture Notes
Subject	Programming for Problem Solving
Topics	Array: Searching and Sorting and 2D Array
Lecture Date	May-June
Faculty Name	Mr. Dilip Kr. Bharti
Section	First Year Section-B

SEARCHING

Linear Search:

Searching, in normal ways, can be coined as "to find the hidden thing". In Array the searching algorithm is used to find whether a given number is present and if it is present then at what location it occurs.

There are three types of searching algorithm present in data structure through which searching any data become more easy.

1. Direct Search or Index wise searching.
2. Linear search or sequential search
3. Binary search

Direct Search or Index wise searching:

Program:

```
#include<stdio.h>
int main()
{
    int arr[100],size,i,pos;

    // length of array
    printf ("Enter the Size of Array:");
    scanf ("%d",&size);
    printf ("Enter %d Elements:\n",size);
    for(i=0;i<size;i++)
        scanf ("%d",&a[i]);
    // enter position to be searched
    printf ("Enter Searching Element:\n");
    scanf ("%d",&pos);
    // logic for searching:

    if(pos<=size)
        printf ("%d value at %d Posistion:",a[pos-1],pos);
    else
        printf ("Please Enter Correct position!!");
    return 0;
}
```

Linear search or sequential search:

Implementation:-

- Step 1- Take array size, input and searching element in the array .
- Step 2- Take the input for the item to be searched in the array.
- Step 3- Linearly traverse the array using a for loop.
- Step 4 – For each array item check if **a[i] == item**
 - If such condition is true print value and its position
 - If no such element is found and the whole array is traversed. Print not found



```
for(i=0;i<size;i++)
{
    If(a[i]==search)
    {
        printf("%d value at %d Position.",a[i],i+1);
        break;
    }
}
```

Algorithm:

Linear(A[], n, Item) Here A is an array of n elements. Item contains the value which we want to search.

In the algorithm we use variables 'location', 'i' and 'found'

1. Start
2. [Initialize] Set found=0
3. [loop]
Repeat step 3 for i=0
to n-1 If A[i] = Item
then
 - a) Set found=1
 - b) break
4. If found==0 then
Write search unsuccessful
else
Write search
successful Write
loc=i+1
5. Stop

Program:

```
// Time Complexity : O(N)
// Space Complexity : O(1)
// Auxiliary Space Complexity : O(N) due to function call stack
#include<stdio.h>
int main()
{
    int arr[100],size,i, search;

    // length of array
    printf ("Enter the Size of Array:");
    scanf("%d",&size);
    printf("Enter %d Elements:\n",size);
    for(i=0;i<size;i++)
        scanf("%d",&a[i]);
    // item to be searched
    printf("Enter Searching Element:\n");
    scanf("%d",&search);
    // logic for searching:
    for(i=0;i<size;i++)
    {
        if(a[i]==search)
        {
            printf("%d value at %d Position:",a[i],i+1);
            break;
        }
    }

    if(i==size)
        printf("%d value Not Found in Array!!",search);
    return 0;
}
```

Binary Search:

Binary search is a very fast and efficient searching algorithm. It requires to be list be in sorted order, ie; either in ascending or descending.

Lets have a look at binary search below –

Time Complexity (Best)	$\Omega(1)$
Time Complexity (Avg)	$\Theta(\log n)$
Time Complexity (Worst)	$O(\log n)$
Space Complexity	$O(1)$

How Binary Search in C works?

1. The array needs to be sorted in either ascending or descending order
2. In our case we are taking an example for an array sorted in ascending order.
3. The searching algorithm proceed from any of two halves
4. Depends upon whether the element you are searching is greater or smaller than the central element
5. If the element is small then, searching is done in first half
6. If it is big then searching is done in second half.

It is fast search algorithm with the complexity of $O(\log n)$.

Implementation of Binary Search

- Take a sorted array (mandatory)
- Find mid using formula $m = (l+r)/2$
- If the item to be searched is greater than mid
 - Check the right subarray
- If the item to be searched is lesser than the mid
 - Check the left subarray
- If mid element == item return with the position where found
- Else keep doing the above steps until you violate the bounds of the array

Algorithm:

Binary(A[],LB,UB,Item,loc):

Here A is a sorted array with lower bound LB and upper bound UB. An item is an element which we want to search. In the algorithm, we use variables 'beg', 'end' and 'mid' denotes the beginning, ending and middle locations of the sorted array. This algorithm finds the location 'loc' of item in the array 'A' or initially we set loc=NULL

1. Start
2. [initialize segment variable]
Set beg=LB end=UB and
 $mid = \text{int}(\text{beg} + \text{end}) / 2$
3. Repeat steps 3 and 4 While (beg<=end and A[mid] is not equal Item) IF item<A[mid] then
Set end=mid-1 ELSE
Set beg=mid+1 [end of if structure]
4. Set $mid = \text{int}(\text{beg} + \text{end}) / 2$ [end of step 2 loop]
5. IF a[mid]==item Set loc=mid+1 ELSE loc=NULL
[end of if structure]
6. If loc==NULL then
Write "search unsuccessful" Otherwise

Write "search successful"[end of if structure]

7. Stop

```
#include<stdio.h>
int main()
{
    int arr[100],size,i,search,first,last,mid;

    // length of array
    printf ("Enter the Size of Array:");
    scanf ("%d",&size);
    // Entered Element should be sorted.
    printf ("Enter %d Elements:\n",size);
    for(i=0;i<size;i++)
        scanf ("%d",&a[i]);
    // item to be searched
    printf ("Enter Searching Element:\n");
    scanf ("%d",&search);
    // logic for searching:
    first=0;
    last=size-1;
    mid=(first+last)/2;
    while(first<=last)
    {
        if(a[mid]<search)
            first= mid+1;
        else if(a[mid]==search)
        {
            printf ("%d value found at %d Position.",a[mid],mid+1);
            break;
        }
        else
            last=mid-1;
        mid=(first+last)/2;
    }
    if(first>last)
        printf ("%d value Not Found in Array!!",search);
    return 0;
}
```

Sorting is the process of arranging data into meaningful sequence so that we can consider it with more ease and convenience. Sort means arranging items into an order either alphabetical or, numerical.

Sorting Techniques

1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Heap Sort
5. Merge Sort
6. Quick Sort
7. Counting Sort

8. Radix Sort
9. Bucket Sort

Sorting of Array In C Programming

Problem Statement – How to arrange array in ascending order

Sorting is the process of arranging elements in a list or array in a specific order, typically in ascending or descending order. Sorting is a fundamental problem in computer science and has many applications in areas such as searching, data compression, and data analysis.

There are many sorting algorithms that can be used to sort an array. Some of the most popular algorithms include:

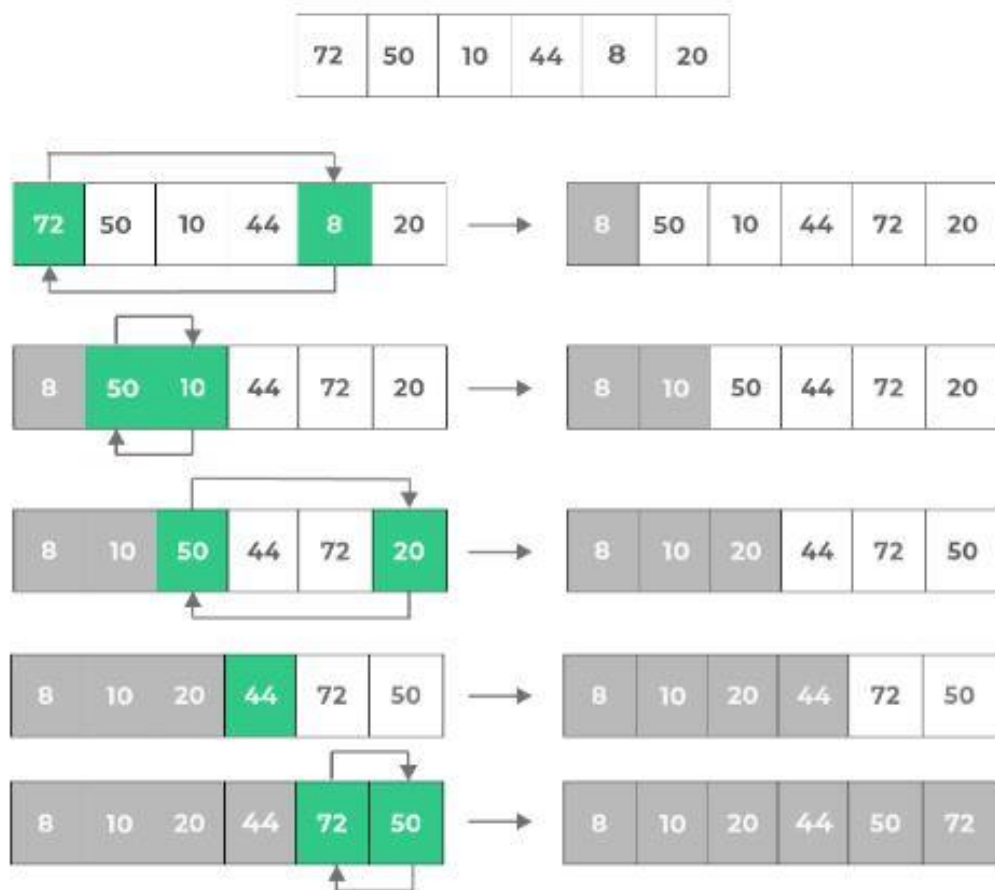
1. **Bubble Sort:** This is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order.
2. **Selection Sort:** This algorithm sorts an array by repeatedly finding the minimum element from the unsorted part of the array and putting it at the beginning.
3. **Insertion Sort:** This algorithm builds the final sorted array one item at a time, by inserting each item in the correct position in the array.

Insertion Sort:

Steps for Selection Sort in C

There are following Step of selection sort algorithm.

- Step 1-Select the smallest value in the list.
- Step 2-Swap smallest value with the first element of the list.
- Step 3-Again select the smallest value in the list (exclude first value).
- Step 4- Repeat above step for (n-1) elements untill the list is sorted.



Algorithm: Selection(a[],n)

This algorithm sort the 'n' elements of array 'a'; Here a[] is an array and it contains n elements.

In the algorithm, we use variables 'i', 'j' 'Min' and 'temp'.

Step 1: BEGIN:

Step 2: FOR i=0 to n-1 DO

Min=i;

FOR j=i+1 to n DO

IF a[j]<a[Min] THEN

Min=j[End of if]

[exchange a[i] and a[Min]]temp=a[i] a[i]=a[min] a[min]=temp

[End of inner loop][End of outer loop]

Step 3: Repeat step 3 for i=0 to n-1

Display a[i]

Step 4: Stop

/ Time Complexity : $O(N^2)$

// Space Complexity : $O(1)$

// Best, Avg, Worst Cases : All of them $O(N^2)$

#include<stdio.h>

int main()

{

int arr[100],size,i,temp;

```

// length of array
printf ("Enter the Size of Array:");
scanf ("%d",&size);
printf ("Enter %d Elements:\n",size);
for(i=0;i<size;i++)
scanf ("%d",&a[i]);
printf ("Before Sorting Elements are:\n");
for (i=0; i < size; i++)
printf ("%d ",a[i]);

// Loop to iterate on array
for (i = 0; i < size-1; i++)
{
    // Here we try to find the min element in array
    for (j = i+1; j < size; j++)
    {
        if (array[i] > a[j])
        {
            // Here we interchange the min element with first one
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }
    printf ("\nAfter sorting: \n");
    for (i=0; i < size; i++)
    printf ("%d ",a[i]);

    return 0;
}

```

Bubble Sort :

Sorting is the process of arranging the data in some logical order. Bubble sort is an algorithm to sort various linear data structures.

The logical order can be ascending and descending in the case of numeric values or dictionary order in the case of alphanumeric values.

- Bubble Sort is a very simple and easy to implement sorting technique.
- In the bubble sort technique, each pair of element is compared.
- Elements are swapped if they are not in order.
- The worst-case complexity of bubble sort is $O(n^2)$.

Time Complexity	$O(n^2)$
Best Case	$O(n)$
Worst Case	$O(n^2)$
Space Complexity	$O(1)$
Avg. Comparisons	$n(n-1)/2$

Implementation of Bubble Sort

The algorithm works on the principle (For ascending order)

- Linearly traverse an array
- Start from the leftmost item
- If left item is greater than its right item swap them

That is $a[i] > a[i+1]$ swap them

			0	1	2	3	4	5	6	7	
			10	6	2	18	16	4	8	14	
Pass 1											
$i = 0$	(Swap)	$j = 0$	6	10	2	18	16	4	8	14	Iteration 1
	(Swap)	$j = 1$	6	2	10	18	16	4	8	14	Iteration 2
	(Swap)	$j = 2$	6	2	18	10	16	4	8	14	Iteration 3
	(Swap)	$j = 3$	6	2	10	16	18	4	8	14	Iteration 4
	(Swap)	$j = 4$	6	2	10	16	4	18	8	14	Iteration 5
	(Swap)	$j = 5$	6	2	10	16	4	8	18	14	Iteration 6
	(Swap)	$j = 6$	6	2	10	16	4	8	14	18	Iteration 7
											18
Pass 2											
$i = 1$	(Swap)	$j = 0$	2	6	10	16	4	8	14		Iteration 1
	(No Swap)	$j = 1$	2	6	10	16	4	8	14		Iteration 2
	(No Swap)	$j = 2$	2	6	10	16	4	8	14		Iteration 3
	(Swap)	$j = 3$	2	6	10	4	16	8	14		Iteration 4
	(Swap)	$j = 4$	2	6	10	4	8	16	14		Iteration 5
	(Swap)	$j = 5$	2	6	10	4	8	14	16		Iteration 6
											16
Pass 3											
$i = 2$	(No Swap)	$j = 0$	2	6	10	4	8	14			Iteration 1
	(No Swap)	$j = 1$	2	6	10	4	8	14			Iteration 2
	(Swap)	$j = 2$	2	6	4	10	8	14			Iteration 3
	(Swap)	$j = 3$	2	6	4	8	10	14			Iteration 4
	(No Swap)	$j = 4$	2	6	4	8	10	14			Iteration 5
											14
Pass 4											
$i = 3$	(No Swap)	$j = 0$	2	6	4	8	10				Iteration 1
	(Swap)	$j = 1$	2	4	6	8	10				Iteration 2
	(No Swap)	$j = 2$	2	4	6	8	10				Iteration 3
	(No Swap)	$j = 3$	2	4	6	8	10				Iteration 4
											10
Pass 5											
$i = 4$	(No Swap)	$j = 0$	2	4	6	8					Iteration 1
	(No Swap)	$j = 1$	2	4	6	8					Iteration 2
	(No Swap)	$j = 2$	2	4	6	8					Iteration 3
											8
Pass 6											
$i = 5$	(No Swap)	$j = 0$	2	4	6						Iteration 1
		$j = 1$	2	4	6						Iteration 2
											6
Pass 7											
$i = 6$	(No Swap)	$j = 0$	2	4							Iteration 1
											4

Algorithm: Bubble(a[],n)

This algorithm sorts the 'n' elements of array 'a'; Here a[] is an array and it contains n elements.

In the algorithm, we use variables 'i', 'j' and 'temp'.

Step 1: BEGIN:

Step 2: FOR i=0 to n-2 DO

Step 3: FOR j=0 to n-1-i DO

Step 4: IF $a[j] > a[j+1]$ THEN

[exchange $a[j]$ and $a[j+1]$]
temp= $a[j]$

$a[j]=a[j+1]$ $a[j+1]=temp$

[End of if] [End of step 3 loop]

[End of step 2 loop] Step 5: FOR i=0 to n-1 DO

Display a[i]

Step 5: Stop

```
#include<stdio.h>
int main()
{
    int arr[100],size,i,temp;

    // length of array
    printf("Enter the Size of Array:");
    scanf("%d",&size);
    printf("Enter %d Elements:\n",size);
    for(i=0;i<size;i++)
        scanf("%d",&a[i]);
    printf("Before Sorting Elements are:\n");
    for (i=0; i < size; i++)
        printf("%d ",a[i]);

    for (i = 0; i < size-1; i++){

        // Since, after each iteration right-most i elements are sorted
        for (j = 0; j < size-i-1; j++)
            if (a[j] > a[j+1])
            {
                temp = a[j]; // swap the element
                a[j] = a[j+1];
                a[j+1] = temp;
            }
        }
    printf("After bubble sort: \n");
    for (i=0; i < size; i++)
        printf("%d ",a[i]);
    return 0;
}
```

Insertion Sort:

In this technique, we pick an Element and then insert it at the appropriate position in ascending and descending order.

The inspiration has been taken from playing cards

Time Complexity	$O(n^2)$
Best Case	$\Omega(n)$
Worst Case	$O(n^2)$
Aux. Space Complexity	$O(1)$
Best case when	Array is already sorted
Worst case when	Array is reverse sorted

Insertion Sort in C

Insertion sort is just like how to sort playing cards in your hands. You pick one card and try to place that card in its correct position.

The array is kind of split into a sorted part and an unsorted part. A value from the unsorted part is picked and placed into its correct position in the sorted part.

Below are examples on how you can do the same.

Execution of Insertion Sort in C

Insertion Sort in C

- Initial Array

8	6	4	20	24	2	10	12
---	---	---	----	----	---	----	----

- Since, $6 < 8$

8	6	4	20	24	2	10	12
---	---	---	----	----	---	----	----

6 will get **inserted** before 8

- Since, $4 < 6$

6	8	4	20	24	2	10	12
---	---	---	----	----	---	----	----

4 will get **inserted** before 6

- 20 is at correct position, no insertion needed

4	6	8	20	24	2	10	12
---	---	---	----	----	---	----	----

- 24 is at correct position, no insertion needed

4	6	8	20	24	2	10	12
---	---	---	----	----	---	----	----

- Since, $2 < 4$

4	6	8	20	24	2	10	12
---	---	---	----	----	---	----	----

2 will get **inserted** before 4

- Since, $10 < 20$

2	4	6	8	20	24	10	12
---	---	---	---	----	----	----	----

10 will get **inserted** before 20

- Since, $12 < 20$

2	4	6	8	10	20	24	12
---	---	---	---	----	----	----	----

12 will get **inserted** before 20

2	4	6	8	10	12	20	24
---	---	---	---	----	----	----	----

Approach of Insertion Sort

- Iterate over the whole array, picking each item in the array once iteratively
- Call this picked item key
- If this key item is smaller than its predecessor, compare it to the item before
- To make space for this swapped item move items greater than key one position right
- Keep doing this until you find a smaller element than key element.

ALGORITHM Insertion Sort ($A[]$, N)

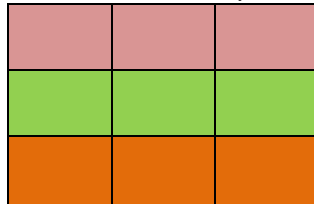
- BEGIN:
- FOR $j=1$ TO $N-1$ DO
Key = $A[j]$
//insert $A[j]$ into sorted sequence ($A[1... j-1]$)
- $i = j-1$
- WHILE $i \geq 0$ AND $A[i] > \text{Key}$ DO
 $A[i+1] = A[i]$
 $i = i-1$
- $i += \text{key};$
- END;

Program:

```
• #include<stdio.h>
• int main()
• {
•     int arr[100],size,i,j,key;
•
•     // length of array
•     printf ("Enter the Size of Array:");
•     scanf ("%d",&size);
•     printf("Enter %d Elements:\n",size);
•     for(i=0;i<size;i++)
•     scanf ("%d",&a[i]);
•     printf("Before Sorting Elements are:\n");
•     for (i=0; i < size; i++)
•     printf("%d ",a[i]);
•     for (i = 1; i < size; i++)
•     {
•         key = a[i];
•         j = i - 1;
•
•         /* Here the elements in b/w array[0 to i-1] which are greater than key are moved ahead by 1 position
each*/
•         while (j >= 0 && a[j] > key)
•         {
•             a[j + 1] = a[j];
•             j = j - 1;
•         }
•         // placing element at its correct position
•         a[j + 1] = key;
•     }
•
•     printf("After Insertion sort: \n");
•     for (i=0; i < size; i++)
•     printf("%d ",a[i]);
•     return 0;
• }
```

2-D Array

2D Array is defined as the continuous block of memory in which a similar data type is stored. 2D Array indexing starts with size 0 and ends with size -1. The size of the 2D array is fixed and it is stored in static memory.



Syntax:

data_type array_name [i][j];

Representation of 2D Array

	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

Array Declaration

Declaration of 2D Array consists of any data type and it can be declared by any random name or any random variable.

Syntax:

```
int arr[3][5];
```

// Here int is the data type of array.

// arr is the name of array.

// 3 is the size of row and 5 is the size of column of an array.

Array Initialization

In 2D Array, All the elements consist of garbage value at initial time but it can be explicitly initialized during declaration.

Syntax:

```
int Arr[i][j] = {value 1, value 2, value 3,...,value ij}
```

Accessing of elements in 2D Array:

In 2D Array elements are accessed by putting the array name and index value inside square brackets.

Syntax:

```
<arr_name>[index];
```

Rules for 2D Array Declaration:

- In 2D array we must have to declare the array variable.
- 2D Array stores the data in contiguous memory locations.
- 2D Array should be a list of similar data types.
- The size of the 2D array should be fixed.
- 2D Array should be stored in static memory.
- 2D Array indexing starts with size 0 and ends with size -1.
- In 2D Array declaration of array size within the bracket is not necessary.
- In 2D Array it must include the variable name and data type while declaring any arrays .

Program: 01

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

Output:

```
1 2 3
4 5 6
7 8 9
```

Program: 02

```
#include
int main() {
    int arr[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    printf("The even elements in the 2d array are : ");
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if(arr[i][j]%2==0) printf(" %d ", *(arr + i) + j));
        }
    }
    return 0;
}
```

Output

The even elements in the 2d array are : 2 4 6 8

Program: 03

```
#include <stdio.h>
int main() {
    int arr[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    for (int i = 0; i < 3; i++) {
```

```

for (int j = 0; j < 3; j++) {
    if(i==0 && j==0){
        printf("The first element of 2d array is : %d\n ", (*(arr + i) + j));
    }
    else if(i==2 && j==2){
        printf("The last element of 2d array is : %d ", (*(arr + i) + j));
    }
}

}
return 0;
}

```

Output

The first element of 2d array is : 1

The last element of 2d array is : 9

Addition of Two Matrix in C

Add Two Matrices :

Write a basic C program to add two matrices for understanding basic structure of multidimensional array , we can add two or more matrices easily by using proper syntax and algorithm.

Algorithm:

To add two matrices the number of row and number of columns in both the matrices must be same the only addition can take place.

Step 1: Start

Step 2: Declare matrix mat1[row][col];

and matrix mat2[row][col];

and matrix sum[row][col]; row= no. of rows, col= no. of columns

Step 3: Read row, col, mat1[][] and mat2[][]

Step 4: Declare variable i=0, j=0

Step 5: Repeat until i < row

5.1: Repeat until j < col

sum[i][j]=mat1[i][j] + mat2[i][j]

Set j=j+1

5.2: Set i=i+1

Step 6: sum is the required matrix after addition

Step 7: Stop

In the above algorithm,

- using scanf() will take the input from the user rows,columns,elements of both the matrix.
- we will the add the element of matrix 1 and matrix 2 in third matrix let say sum matrix.

- printf () will print the output on the screen
- we have to use the for loop to input the element of matrix ,sum the elements of matrices and print the final result.

Program: Add Two Matrices :

```
#include <stdio.h>
int main ()
{
    int mat1[3][3] ;
    int mat2[3][3];
    int sum[3][3], i, j;

    printf ("Enter matrix mat1: \n");
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            printf ("Enter the elements of matrix P [%d,%d]: ", i, j);
            scanf ("%d", &mat1[i][j]);
        }
    }
    printf ("Enter matrix mat2: \n");
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            printf ("Enter the elements of matrix P [%d,%d]: ", i, j);
            scanf ("%d", &mat2[i][j]);
        }
    }

    printf ("matrix 1 is :\n");
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            printf ("%d ", mat1[i][j]);

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

    printf ("matrix 2 is :\n");
```



```

for (i = 0; i < 3; i++)
{
    for (j = 0; j < 3; j++)
    {
        printf ("%d  ", mat2[i][j]);

    }
    printf ("\n\n");

}
// adding two matrices
for (i = 0; i < 3; i++)
for (j = 0; j < 3; j++)
{
    sum[i][j] = mat1[i][j] + mat2[i][j];
}

// printing the sum Of two matrices
printf ("\nSum of two matrices: \n");
for (i = 0; i < 3; i++)
{
    for (j = 0; j < 3; j++)
    {
        printf ("%d  ", sum[i][j]);

    }
    printf ("\n\n");

}

return 0;
}

```

Output :

matrix 1 is :

0 1 2

3 4 5

6 7 8

matrix 2 is :

9 10 11

12 13 14

15 16 17

Sum of two matrices:

9 11 13

15 17 19

Matrix Multiplication

Matrix Multiplication :

On this page we will write a basic of C Matrix Multiplication for understanding basic structure of multidimensional array in C programming. In programming , we can multiply two matrices easily by using proper syntax and algorithm.

Algorithm:

Matrix multiplication is an important program to understand the basics of c programming .For matrix multiplication number of columns in first matrix must be equal to the number of rows in second matrix. Also the number of rows in third matrix which contains the result of multiplication of two matrices is equal to number of row in first matrix and number of columns is equal to number of columns in second matrix.

Algorithm:

- **Step 1:** Start
- **Step 2:** Declare matrix `mat1[row1][col1]`; and matrix `mat2[row2][col2]`; and matrix `mul[row1][col2]`;
row= no. of rows, col= no. of columns
- **Step 3:** Read `mat1[row1][col1]` and `mat2[row2][col2]`
- **Step 4:** Declare variable `i=0, j=0`
- **Step 5:** Set a loop from `i=0` to `i=row1`.
- **Step 6:** Set an inner loop for the above loop from `j=0` to `j=col2`
 - Initialise the value of the element (i, j) of the new matrix (`mul[row1][col2]`) to 0.
 - Set an inner loop inside the above loop from `k=0` to `k=row1`.
 - Using the add and assign operator (`+=`) store the value of `mat1[i][k] * mat2[k][j]` in the third matrix, `mul[i][j]`.
 - Print the third matrix.
- **Step 7:** Stop

Program:

```
#include<stdio.h>
int main ()
{
    int mat1[3][3] ;
    int mat2[3][3] ;
    int mul[3][3], i, j, k;
    printf ("Enter matrix mat1: \n");
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            printf ("Enter the elements of matrix P [%d,%d]: ", i, j);
            scanf ("%d", &mat1[i][j]);
        }
    }
    printf ("Enter matrix mat2: \n");
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            printf ("Enter the elements of matrix P [%d,%d]: ", i, j);
            scanf ("%d", &mat2[i][j]);
        }
    }

    printf ("matrix 1 is :\n");
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            printf ("%d ", mat1[i][j]);
        }
        printf ("\n\n");
    }

    printf ("matrix 2 is :\n");
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            printf ("%d ", mat2[i][j]);
        }
        printf ("\n\n");
    }

    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] += mat1[i][k] * mat2[k][j];
        }
    }
}

printf ("The product of the two matrices is: \n");
for (i = 0; i < 3; i++)
{
    for (j = 0; j < 3; j++)
    {
        printf ("%d\t", mul[i][j]);
    }
    printf ("\n");
}
return 0;
}

```

Output:

matrix 1 is:

```

1 1 1
2 2 2
3 3 3

```

Matrix 2 is:

```

1 1 1
2 2 2
3 3 3

```

The product of the two matrices is:

```

6 6 6
12 12 12
18 18 18

```

Matrix Transpose

Matrix Transpose

We will write a C program for matrix transpose and the number will be entered by the user. This will help to understand the basic structure of programming. In this program , we will display the matrix transpose of given input easily by using proper syntax and algorithms.

Working of Program :

In the program, we will require some numbers from the user to display the matrix transpose of given input.

Important Note :

- To calculate the matrix transpose, we can interchange the rows and column of the matrix i.e. write the elements of the rows as columns and the elements of columns as rows.
- The matrix transpose is can be flipped version of original matrix.

Syntax of for() Loop:-

```
for (Initialization, condition, updation)
{
    //code
}
```

Problem 1

Write a program to calculate matrix transpose.

- Firstly, we have to enter the input.
- Then print the matrix transpose.

Program:

```
#include<stdio.h>
void transpose (int p[3][3], int t[3][3]);
int main ()
{
    int i, j;
    int p[3][3], t[3][3];
    printf ("Enter matrix P: \n");
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            printf ("Enter the elements of matrix P [%d,%d]: ", i, j);
            scanf ("%d", &p[i][j]);
        }
    }
    transpose (p, t);
    printf ("Transpose of matrix P is:\n\n");
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            printf ("%d ", t[i][j]);
        }
        printf ("\n");
    }
}

void transpose (int p[3][3], int t[3][3])
{
    int row, col;
    for (row = 0; row < 3; row++)
    {
        for (col = 0; col < 3; col++)
        {
```

```

        t[row][col] = p[col][row];
    }
}

```

Output

Enter matrix P: 2

Enter the elements of matrix P [0,0]: 1
 Enter the elements of matrix P [0,1]: 2
 Enter the elements of matrix P [0,2]: 3
 Enter the elements of matrix P [1,0]: 4
 Enter the elements of matrix P [1,1]: 5
 Enter the elements of matrix P [1,2]: 6
 Enter the elements of matrix P [2,0]: 7
 Enter the elements of matrix P [2,1]: 8
 Enter the elements of matrix P [2,2]: 9
 Transpose of matrix P is:

```

1 4 7
2 5 8
3 6 9

```

Note:

In the following program we will display the matrix transpose for given input.

Problem 2

Write a program to calculate the matrix transpose.

- Firstly, we have to enter the number.
- Then print the matrix transpose.

Program:

```

#include<stdio.h>
int main()
{
    int mat[10][10] ;
    int i, j, row, col ;
    printf("Enter the order of the matrix = ") ;
    scanf("%d %d", &row, &col) ;
    printf("\nEnter the elements of the matrix = \n\n") ;
    for(i = 0 ; i < row ; i++)
        for(j = 0 ; j < col ; j++)
            scanf("%d", &mat[i][j]) ;
    printf("\nThe transpose matrix is = \n\n") ;
    for(i = 0 ; i < col ; i++)
    {
        for(j = 0 ; j < row ; j++)
        {
            printf("%d \t", mat[j][i]) ;
        }
    }
}

```

```
printf("\n");  
}  
return 0;  
}
```

Output

Enter the order of the matrix = 3

4

Enter the elements of the matrix =

11
22
33
44
55
66
77
88
99
12
23
34

The transpose matrix is =

11	55	99
22	66	12
33	77	23
4	88	34