CSE101-Lec#15,16, 17

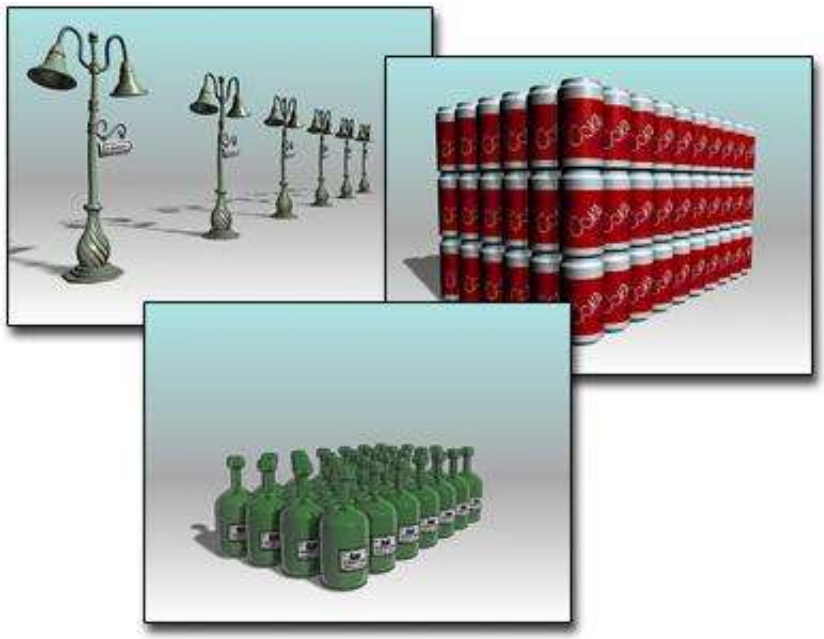
- What are Arrays?
- To declare an array
- To initialize an array
- To display address of the array
- Basic program examples of 1D array
- To pass an array to a function(By reference and By value)
- Applications and Operations on 1D Array

Outline

- To declare an array
- To initialize an array
- To display address of the array
- Basic program examples of 1D array
- To pass an array to a function(By reference and By value)
- Applications and Operations on 1D Array

Introduction

- Arrays
  - Collection of **related** data items of **same data type**.
  - Static entity – i.e. they remain the same size throughout program execution

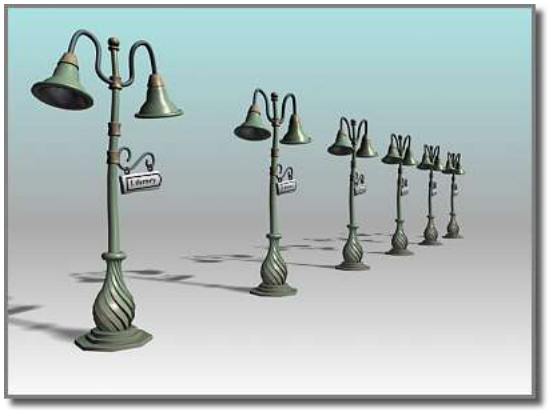


Types

- One Dimensional (e.g. int a[100])
- Multidimensional(2D,3D….)(e.g. int a[5][5], int a[5][5][5])

1-D array

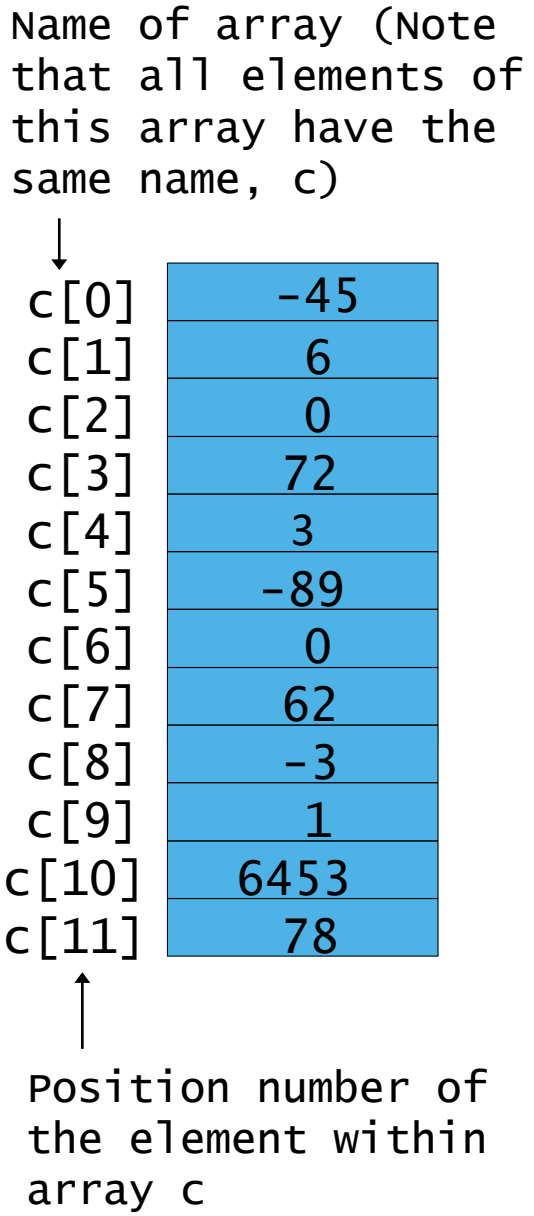
- A single- dimensional or 1-D array consists of a fixed number of elements of the same data type organized as a single **linear** sequence.
- The elements of a single dimensional array can be accessed by using a single subscript
- The arrays we have studied till now were 1D array or linear array.
- Example: `int a[n];`



Arrays(1D)

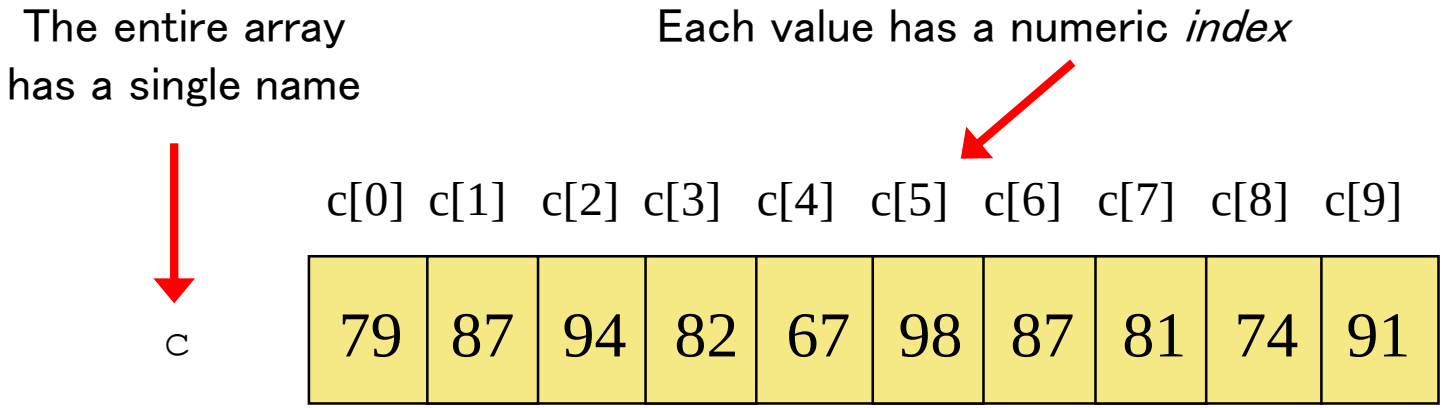
- Array
  - Group of consecutive memory locations
  - Same name and data type
- To refer to an element, specify:
  - Array name
  - Position number in square brackets([])
- Format:

  arrayname[position_number]
  - First element is always at position `0`
  - Eg. `n` element array named `c`:
    - `c[0], c[1]...c[n - 1]`



Arrays

- An *array* is an ordered list of values



An array of size **N** is indexed from **zero to N−1**

This array holds 10 values that are indexed from 0 to 9

Arrays

- Array elements are like normal variables

```
c[0] =  3;/*stores 3 to c[0] element*/
scanf ("%d", &c[1]);/*reads c[1] element*/
printf ("%d, %d", c[0], c[1]); /*displays
                                c[0] & c[1] element*/
```

- The position number inside square brackets is called **subscript/index**.
- Subscript must be integer or an integer expression

```
c[5 - 2] = 7;  (i.e. c[3] = 7)
```

Defining Arrays

- When defining arrays, specify:
  - Name
  - Data Type of array
  - Number of elements

```
datatype arrayName[numberOfElements];
```

  - Examples:

```
int students[10];
float myArray[3284];
```

- Defining multiple arrays of same data type
  - Format is similar to regular variables
  - Example:

```
int b[100], x[27];
```

Initializing Arrays

Different ways of initializing ID Arrays

1) Initializing array at the point of declaration

- int a[5]={1,2,3,4,5};
- int a[]={1,2,3,4,5};//Here compiler will automatically depict the size:5
- int a[5]={1,2,3};//Partial initialization[Remaining elements will be initialized to default values for integer, i.e. 0]
- int a[5]={};//All elements will be initialized to zero
- int a[5]={1};//First element is one and the remaining elements are default values for integer, i.e. 0

Initializing Arrays

2) Initializing array after taking input from the user

- Array is same as the variable can prompt for value from the user at run time.
- Array is a group of elements so we use **for** loop to get the values of every element instead of getting single value at a time.
- Example: int array[5], i; // array of size 5

```
for(i=0;i<5;i++){// loop begins from 0 to 4
    scanf("%d", &array[i]);
}
```

Printing base address of the array and address of any array element

```
#include<stdio.h>
int main()
{
int a[5]={1,2,3,4,5};
int i;
printf("\n Printing base address of the array:");
printf("\n%u %u %u",&a[0],a,&a);
printf("\n Printing addresses of all array elements:");
for(i=0;i<5;i++)
{
printf("\n%u",&a[i]);
}
return 0;
}
```

Program example 1-WAP to read and display elements of 1D Array

```
#include<stdio.h>
int main()
{
        int a[100],n,i;
        printf("\n Enter number of elements:");
        scanf("%d",&n);
        printf("\n Enter array elements:");
        for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
        }
        printf("\n Entered array elements are:");
        for(i=0;i<n;i++)
        {
                printf("\n%d",a[i]);
        }
        return 0;
}
```

Program example-2 WAP to find the sum of all 1D array elements

```
#include<stdio.h>
int main()
{
        int a[100],n,i,sum=0;
        printf("\n Enter number of elements:");
        scanf("%d",&n);
        printf("\n Enter array elements:");
        for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
        }
        for(i=0;i<n;i++)
        {
                sum=sum+a[i];
        }
        printf("\n Sum of array elements is:%d",sum);
        return 0;

}
```

Program example 3-WAP to display the largest and smallest element from 1D array elements

```
#include<stdio.h>
int main()
{
        int n,a[10],i,max,min;
        printf("\n Enter number of elements:");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
        scanf("%d",&a[i]);
        }
        min=a[0];
        for(i=1;i<n;i++)
        {
        if(a[i]<min)
        {
        min=a[i];
        }
        }
        max=a[0];
        for(i=1;i<n;i++)
        {
        if(a[i]>max)
        {
        max=a[i];
        }
        }
        printf("\nMaximum element is: %d",max);
        printf("\nMinimum element is: %d",min);
        return 0;
        }
```

Passing Arrays to Function

- Arrays can be passed to functions in two ways:

1. **Pass entire array**
2. **Pass array element by element**

Pass entire array

- Here entire array can be passed as an argument to the function
- Function gets **complete access** to the original array
- While passing entire array Address of first element is passed to function, any changes made inside function, directly **affects the Original value.**

```
void modifyArray(int b[], int arraySize);
```

- Function passing method: **" Pass by Address"**

Passing array element by element

- Here individual elements are passed to the function as argument
- Duplicate **carbon copy of Original variable** is passed to function
- So any changes made inside function **does not affects the original value**
- Function doesn't get complete access to the original array element.

```
void modifyElement(int e);
```

- Function passing method: **" Pass by Value"**

Passing Arrays to Functions

- Function prototype

```
void modifyArray(int b[], int arraySize);
```

  - Parameter names optional in prototype
    - `int b[]` could be written `int []`
    - `int arraySize` could be simply `int`

```
void modifyArray(int [], int);
```

- Function call

```
int a[SIZE];
modifyArray(a, SIZE);
```

Program example-1 Passing Array to a function using by reference(or Passing entire array at once)

```
#include<stdio.h>
void reference(int[],int);
int main()
{
        int arr[100],n;
        int i;
        printf("\n Enter n:");
        scanf("%d",&n);
        printf("\n Enter array elements:");
        for(i=0;i<n;i++)
  {
        scanf("%d",&arr[i]);
  }
        printf("\n Elements by reference:");
        reference(arr,n);//Passing array by call by reference
        return 0;
}
```

```
void reference(int x[],int size)
{
        int i;
        for(i=0;i<size;i++)
        {
                printf("%d ",x[i]);
        }
}
```

Program example-2 Passing Array to a function using by value(or Passing element by element)

```
#include<stdio.h>
void value(int);
int main()
 {
            int arr[100],n;
             int i;
             printf("\n Enter n:");
             scanf("%d",&n);
             printf("\n Enter array elements:");
             for(i=0;i<n;i++)
            {
             scanf("%d",&arr[i]);
            }
             printf("\n Passing elements by value:");
             for(i=0;i<5;i++)
             {
             value(arr[i]);//Passing array value by Call by value
              }
             return 0;
}
```

```
void value(int u)
{
            printf("%d ",u);
}
```

Application Of Array :

Stores Elements of Same Data Type

- Array is used to store the number of elements that are of same data type.
- Eg: `int students[30];`
- array of marks of five subjects for single student.

```
float marks[5];
```

- array of marks of five subjects for 30 students.

```
float marks[30][5]
```

- Similarly if we declare the character array then it can hold only character.
- So in short character array can store character variables while floating array stores only floating numbers.

Array Used for Maintaining multiple variable names using single name

Suppose we need to store 5 roll numbers of students then without declaration of array we need to declare following -

int roll1,roll2,roll3,roll4,roll5;

1. Now in order to get roll number of first student we need to access roll1.
2. Guess if we need to store roll numbers of 100 students then what will be the procedure.
3. Maintaining all the variables and remembering all these things is very difficult.

- So we are using array which can store multiple values and we have to remember just single variable name.

Array Can be Used for Sorting Elements

- We can store elements to be sorted in an array and then by using different sorting technique we can sort the elements.

Different Sorting Techniques are :

1. Bubble Sort
2. Insertion Sort
3. Selection Sort

Array Can Perform Matrix Operation

Matrix operations can be performed using the array. We can use 2-D array

- To store the matrix.
- To perform all mathematical manipulations on matrix.
- Matrix can be multi-dimensional.

Searching in Arrays

- The process of finding a particular element of an array is called searching.
- Search an array for a *key* value.
- Two searching techniques:
  - Linear search
  - Binary search

Linear search

- Linear search
  - Simple
  - Compare each element of array with key value
  - Useful for small and unsorted arrays
- It simply examines each element sequentially, starting with the first element, until it finds the key element or it reaches the end of the array.

  Example: If you were looking for someone on a moving passenger train, you would use a sequential search.

Program example-WAP to implement linear search in 1D array element

```
#include <stdio.h>
int main()
{
  int a[50];
  int i, loc = -1, key,n;
  printf("\n Enter value of n:");
  scanf("%d",&n);
  printf("\n Enter the elements:");

  for(i=0;i<n;i++)
  {
                scanf("%d",&a[i]);
  }
  printf("Enter integer value to search in array:");
  scanf( "%d", &key );
  // attempt to locate searchKey in array a
  for ( i = 0; i < n; i++ )
  {
   if ( a[i] == key )
   {
    loc = i; // location of key is stored
    break;
   } // end if
  } // end for
  if(loc!= -1)
   {
    printf("Element found at %d",loc+1);
   }
   else
   {
    printf("Element not found");
   }
} // end main
```

Binary search

- Binary search
  - Applicable for **sorted** arrays
- The algorithm locates the **middle** element of the array and compares it to the key value.
  - Compares `middle` element with the `key`
    - If equal, match found
    - If `key < middle`, looks in left half of `middle`
    - If `key > middle`, looks in right half of `middle`
    - Repeat (the algorithm is repeated on **one-quarter** of the original array.)

Binary search

- It repeatedly divides the sequence in two, each time restricting the search to the half that would contain the element.
- This is a tremendous increase in performance over the linear search that required comparing the search key to an average of half of the array elements.
- You might use the binary search to look up a word in a dictionary

Program example-WAP to implement Binary Search in 1D array elements

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

 int a[50],n,loc=-1, key, beg,last,mid,i;
 printf("\n Enter number of array elements:");
 scanf("%d",&n);
 printf("\n Enter array elements:");
 for(i=0;i<n;i++)
 {
                scanf("%d",&a[i]);
 }
 beg=0;
 last=n-1;
 printf("Enter integer value to search in sorted array:");
 scanf( "%d", &key );
  while(beg<=last)//Loop will run until unless only one element is not
remaining
  {
   mid = (beg + last) / 2; // determine index of middle element
   if(a[mid]==key)
   {
    loc=mid; //save the location of element.
    break;
   }
 else if(a[mid]>key) //Middle element is greater than key
   {
    last=mid-1;//If middle element is greater than key, we
need to search left subarray
   }
  else if(a[mid]<key) //Middle element is less than key
   {
    beg=mid+1;//If middle element is less than key, we
need to search right subarray
   } //end of if else
  } //end of while
  if(loc!=-1)
  {
   printf("element found at %d", loc+1);//Location is exact
position, not index
  }
  else
  {
   printf("element not found");
  }
  return 0;
}
```

Dry running

Algorithm is: (Basic logic)

BINARY SEARCH

```
beg = 0;        (beg, last, a, n, key (element to search))
last = n-1;
while ( beg <= last)
{
  mid = (beg + last) / 2;
  if (a[mid] == key)
  {
    loc = mid
    break;
  }
  else if (a[mid] > key)
  {
    last = mid - 1;
  }
  else if ( a[mid] < key)
  {
    beg = mid + 1;
  }
}
if ( loc != -1)
{
printf ( "\n Found at: %d", loc+1);
}
else
{
printf ("\n Not found");
}
```

Dry running

Take sorted array :- n = 11

0	1	2	3	4	5	6	7	8	9	10
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]

Key = 9 (consider)

beg = 0 [starting index]

last = 10 [Last index]

Iteration 1: beg <= last (0 <= 10) True

mid = (0+10)/2 = 10/2 = 5.

a[mid] == key, a[5] ≃ 5 == 9 ×

a[mid] > key, 5 > 9 ×

a[mid] < key, 5 < 9 ✓

So, beg = mid + 1, beg = 5+1 = 6.

Now, beg = 6, last = 10

Iteration 2: beg <= last (6 <= 10) True

mid = (6+10)/2 = 16/2 = 8.

a[mid] == key, a[8] ≃ 8 == 9 ×

a[mid] > key, 8 > 9 ×

a[mid] < key, 8 < 9 ✓

So, beg = mid + 1, beg = 8+1 = 9

Now, beg = 9, last = 10

Iteration 3: beg <= last (9 <= 10) True

mid = (9+10)/2 = 19/2 = 9

a[mid] == key, a[9] ≃ 9 == 9 ✓ Element found

loc = 9 | loop stops

So, Element found.

index : 9, Exact position : 10

Sorting

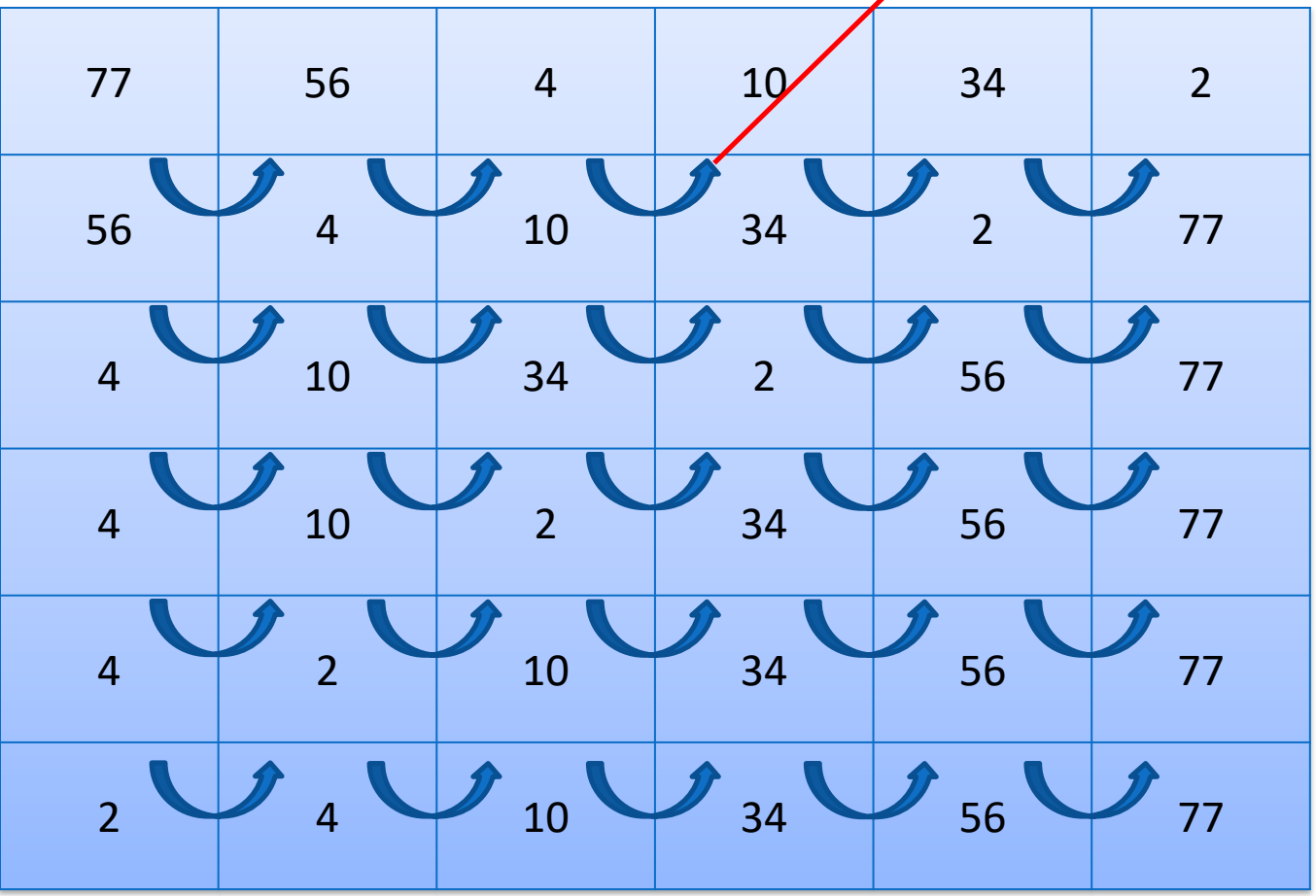
- Sorting data
  - Important computing application
  - Virtually every organization must sort some data

Bubble sort

Bubble sort (sinking sort)

- A simple but inefficient sorting technique.
  - Several passes through the array
  - Successive pairs of elements are compared
    - If increasing order (or identical), no change
    - If decreasing order, elements exchanged
  - Repeat

Bubble sort



Total number of pass required for sorting: n-1

Bubble sort

- It's called bubble sort or sinking sort because smaller values gradually "bubble" their way to the top of the array (i.e., toward the first element) like air bubbles rising in water, while the larger values sink to the bottom (end) of the array.
- The technique uses nested loops to make several passes through the array.
  - Each pass compares successive pairs of elements.
  - If a pair is in increasing order (or the values are equal), the bubble sort leaves the values as they are.
  - If a pair is in decreasing order, the bubble sort swaps their values in the array.

Bubble sort

- The first pass compares the first two elements of the array and swaps their values if necessary. It then compares the second and third elements in the array. The end of this pass compares the last two elements in the array and swaps them if necessary.
- After one pass, the largest element will be in the last index. After two passes, the largest two elements will be in the last two indices.

Program example-WAP to sort elements of 1D array in ascending order using Bubble sort

```
#include <stdio.h>
int main()
{
  int a[100];
  int hold,i,j,n;
  printf("\n Enter value of n:");
  scanf("%d",&n);
  printf("\n Enter elements:");
  for(i=0;i<n;i++)
  {
            scanf("%d",&a[i]);
  }
  printf( "Data items in original order" );
  for (i=0;i<n;i++ )
  {
    printf("%d ",a[i]);//Elements will
come with space
  } // end for
  // bubble sort
// loop to control number of passes(no. of passes are
always n-1)
  for (i=0;i<n-1;i++)
  {
   // loop to control number of comparisons per
pass(There is one comparison less)

                for (j=0;j<n-i-1;j++)
                 {
      // compare adjacent elements and swap them if
first
      // element is greater than second element
      if (a[j]>a[j+1])
                                {
        hold=a[j];
        a[j]=a[j+1];
        a[j+1]=hold;
      } // end if
    } // end inner for
  } // end outer for
  printf( "\nData items in ascending order" );
  for (i=0;i<n;i++)
  {
    printf("%d ",a[i]);
  } // end for
} // end main
```

Dry running

BUBBLE SORT

```
for( i= 0; i< n-1; i++)
    for( j=0; j< n-i-1; j++)
        if (a[j] > a[j+1])
        {
            hold = a[j];
            a[j] = a[j+1];
            a[j+1] = hold;
        }
```

Consider, Unsorted array: 5, 3, 1, 9, 8, 2, 4, 7 $n = 8$

Iteration 1: $i=0$ **Pass 1** $0<8-1$, $0<7$ True

inner loop

$j=0$, $0<8-0-1$, $0<7$

5 3 1 9 8 2 4 7

3 5 1 9 8 2 4 7 [1<7]

3 1 5 9 8 2 4 7 (No Swap) [2<7]

3 1 5 9 8 2 4 7 [3<7]

3 1 5 8 9 2 4 7 [4<7]

3 1 5 8 2 9 4 7 [5<7]

3 1 5 8 2 4 9 7 [6<7]

3 1 5 8 2 4 7 9 [7<7]

Inner Loop stops

Iteration 2: $i=1$ **(Pass 2)** $1<8-1$, $1<7$ True

3 1 5 8 2 4 7 9 (0<6)

1 3 5 8 2 4 7 9 (No Swap) (1<6)

1 3 5 8 2 4 7 9 (No Swap) (2<6)

1 3 5 8 2 4 7 9 (3<6)

1 3 5 2 8 4 7 9 (4<6)

1 3 5 2 4 8 7 9 (5<6)

1 3 5 2 4 7 8 9 (6<6) ✗ Inner Loop

Iteration 3: $i=2$ **(Pass 3)** $2<7$

1 3 5 2 4 7 8 9 (No Swap)

1 3 5 2 4 7 8 9 (No Swap)

1 3 5 2 4 7 8 9

1 3 2 5 4 7 8 9

1 3 2 4 5 7 8 9 (No Swap)

1 3 2 4 5 7 8 9 [No Swap]

Iteration 4: $i=3$ **(Pass 4)** $3<7$

1 3 2 4 5 7 8 9 (No Swap)

1 3 2 4 5 7 8 9

1 2 3 4 5 7 8 9 (No Swap)

1 2 3 4 5 7 8 9 (No Swap)

1 2 3 4 5 7 8 9

Iteration 5: $i=4$ **(Pass 5)** $4<7$

1 2 3 4 5 7 8 9 (No Swap)

1 2 3 4 5 7 8 9 (No Swap)

1 2 3 4 5 7 8 9 (No Swap)

1 2 3 4 5 7 8 9

Iteration 6: $i=5$ **(Pass 6)** $5<7$

1 2 3 4 5 7 8 9 (No Swap)

1 2 3 4 5 7 8 9 (No Swap)

1 2 3 4 5 7 8 9

Iteration 7: $i=6$ **(Pass 7)** $6<7$

1 2 3 4 5 7 8 9 (No Swap)

1 2 3 4 5 7 8 9 - SORTED

Operations on arrays

- Insertion of element into an array
- Deletion of element from an array

Write a program to insert an element at a given position in 1D array

```
#include <stdio.h>
int main()
{
  int array[100], position, c, n, value;
  printf("Enter number of elements in array:\n");
  scanf("%d", &n);
  printf("Enter %d elements:\n", n);
  for (c = 0; c < n; c++)
   {
                scanf("%d", &array[c]);
                }
  printf("Enter the location where you wish to insert
an element:\n");
  scanf("%d", &position);
  printf("Enter the value to insert:\n");
  scanf("%d", &value);

  for (c = n - 1; c >= position - 1; c--)
  {
  array[c+1] = array[c];
  }
  array[position-1] = value;
 printf("Resultant array is:\n");
  for (c = 0; c <= n; c++)
  {
  printf("%d\n", array[c]);
  }
  return 0;
}
```

.

Dry running

Insertion

main logic :

Take I/P for position and Value

```
for (c=n-1; c >= position-1; c--)
{
  array[c+1] = array[c];
}
array[position-1] = value;
for (c=0; c<=n; c++)
{
  printf("%d\n", array[c]);
}
```

Values of array after insertion will be printed [one value will be extra]

Dry running

Consider array :-

1	2	3	4	5
[0]	[1]	[2]	[3]	[4]

Consider position = 2, Value to insert = 55, n = 5

[c=4; c >= 1; c--] - Loop

Iteration 1 : 4 >= 1 (True)

array[c+1] = array[c] => array[5] = array[4]

1	2	3	4	5	5
[0]	[1]	[2]	[3]	[4]	[5]

Iteration 2 : 3 >= 1 (True)

array[c+1] = array[c] => array[4] = array[3]

1	2	3	4	4	5
[0]	[1]	[2]	[3]	[4]	[5]

Iteration 3 : 2 >= 1 (True)

array[c+1] = array[c] => array[3] = array[2]

1	2	3	3	4	5
[0]	[1]	[2]	[3]	[4]	[5]

Iteration 4 : 1 >= 1 (True)

array[c+1] = array[c] => array[2] = array[1]

1	2	2	3	4	5
[0]	[1]	[2]	[3]	[4]	[5]

0 >= 1 (false) Loop stops

array[position-1] = value;

array[1] = 55

1	55	2	3	4	5
[0]	[1]	[2]	[3]	[4]	[5]

(Value inserted at position : 2)

Write a program delete an element from a given position in 1D array

```
#include <stdio.h>
int main()
{
  int array[100], position, c, n;
  printf("Enter number of elements in array\n");
  scanf("%d", &n);
  printf("Enter %d elements\n", n);
  for (c = 0; c < n; c++)
   {
              scanf("%d", &array[c]);
              }
  printf("Enter the location where you wish to delete
from an array\n");

  scanf("%d", &position);
  for (c = position-1; c < n-1; c++)
   {
              array[c] = array[c+1];
   }

.
```

```
 printf("Resultant array
is\n");
  for (c = 0; c < n-1; c++)
   {
          printf("%d\n",
array[c]);
          }
}

.
```

Dry running

Deletion

main logic:

Take input for position from user.

```
for(c = position-1; c<n-1; c++)
  {
  array[c] = array[c+1];
  }
for( c=0; c<n-1; c++)
  {
  printf("%d\n", array[c]);
  }
```

→ Values of array will be printed after deletion [one value will be less]

Dry running

Consider: (n=5)

array:

1	2	3	4	5
[0]	[1]	[2]	[3]	[4]

position from where you want to delete the element : 2.

[c = 2-1 ; c<4 ; c++] → Loop

Iteration 1: c = 2-1 = 1 (1<4) True

array[1] = array[2]

1	3	3	4	5
[0]	[1]	[2]	[3]	[4]

Iteration 2: (2<4) True

array[2] = array[3]

1	3	4	4	5
[0]	[1]	[2]	[3]	[4]

Iteration 3: (3<4) True

array[3] = array[4]

1	3	4	5	5
[0]	[1]	[2]	[3]	[4]

4<4 (false) Loop stops

So, finally 1, 3, 4, 5 are displayed and 2 is deleted.

Now display the array elements

```
for( c=0; c<4; c++)
```

when
c=0, 0<4 → 1 is displayed
c=1, 1<4 → 3 is displayed
c=2, 2<4 → 4 is displayed
c=3, 3<4 → 5 is displayed
c=4, 4<4 → Loop stops.

Q1

If the size of the array is 100, then last index will be:

A. 100

B. 99

C. 98

D. 0

Q2

For the given array, int a={22,3,44,8,9}; what value will be coming for a[3]??

A. 22

B. 44

C. 9

D. 8

Q3

For the given array, int a[5]={}; what value will be coming for a[1]??

A.1

B. Garbage value

C.0

D.-1

Q4

What will be the output of following code?

```
#include<stdio.h>
int main()
{
int a[5],i;
for(i=0;i<5;i++)
{
a[i]=i;
}
printf("%d",a[2]);
return 0;
}
```

A. 0
B. 2
C. 1
D. Garbage value

Q5

What will be the output of following code?

```
#include<stdio.h>
int main()
{
int a[5]={11,22,33,44,55};
a[2]=a[1];
a[3]=a[2];
a[4]=a[3];
printf("%d",a[4]);
return 0;
}
```

A. 22
B. 33
C. 44
D. 55

Q6

```
#include<stdio.h>
int main()
{
int a[5]={1,2,3,4,5};
int b[5];
int i;
for(i=0;i<5;i++)
{
b[i]=++a[i];
}
printf("%d",b[0]+b[3]);
return 0;
}
```

A. 5
B. 7
C. 4
D. 3

Q7

What will be the output of the following program?

```
#include<stdio.h>
int main()
{
int a[5] = {5, 1, 15, 20, 25};
int i=2;
printf("%d",a[++i]);
return 0;
}
```

A. 15
B. 20
C. 25
D. 5

Q8

What will be the output of the following program?

```
#include<stdio.h>
int main()
{
int a[5] = {5, 1, 15, 20, 25};
int i, j, m;
i  = ++a[1];
j = a[1]++;
m = a[i++];
printf("%d %d %d",i,j,m);
return 0;
}
```

A. 3, 2, 15

B. 2, 3, 20

C. 2, 1, 15

D. 1, 2, 5

Q9

What is the output of C program.?

```
int main()
{
 float marks[3] = {90.5, 92.5, 96.5};
 int a=0;
 while(a<3)
 {
 printf("%.2f ", marks[a]);
 a++;
 }
return 0;
}
```

A) 90.5 92.5 96.5
B) 90.50 92.50 96.50
C) 0.00 0.00 0.00
D) Compiler error

Q10

If the number of elements in array are:20, then number of passes as per bubble sort will be

A. 20

B. 19

C. 18

D. 21

Q11

If array elements are already arranged in ascending order, and if the key element is 23, and middle indexed element is 35, then what expression will be used as per the condition in binary search?

A. beg=mid+1

B. last=mid-1

C. mid=mid-1

D. beg=beg+1

Here, beg: starting index, last: ending index, mid:middle index

Q12

If array elements are already arranged in ascending order, and if the key element is 98, and middle indexed element is 50, then what expression will be used as per the condition in binary search?

A. beg=mid+1

B. last=mid-1

C. mid=mid-1

D. beg=beg+1

Here, beg: starting index, last: ending index, mid:middle index

Q13

```
//What will be the output of the following program?
#include<stdio.h>
void process(int[],int);
int main()
{
int a[5] = {15, 3, 10, 4, 6};
process(a,5);
printf("%d",a[0]+a[1]);
return 0;
}
void process(int x[],int z)
{
   int i;
   for(i=0;i<z;i++)
   {
      if(x[i]%5==0)
      {
         x[i]=-1;
      }
   }
}
```

A. 18

B. 2

C. -2

D. 3

Multidimensional Arrays(2D Array)

2D-Array

- A two – dimensional array can be seen as a table with ‘x’ rows and ‘y’ columns where the row number ranges from 0 to (x-1) and column number ranges from 0 to (y-1). A two – dimensional array ‘x’ with 3 rows and 3 columns is shown below:

	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]

- Examples:

➢ Representing the marks of 60 students of class in 5 subjects, we can take a 2D array of row size:60 and column size:5

➢ Representing the sales done by all(50) sales persons of a particular branch for all months(12), we can take a 2D array of row size:50 and column size:12

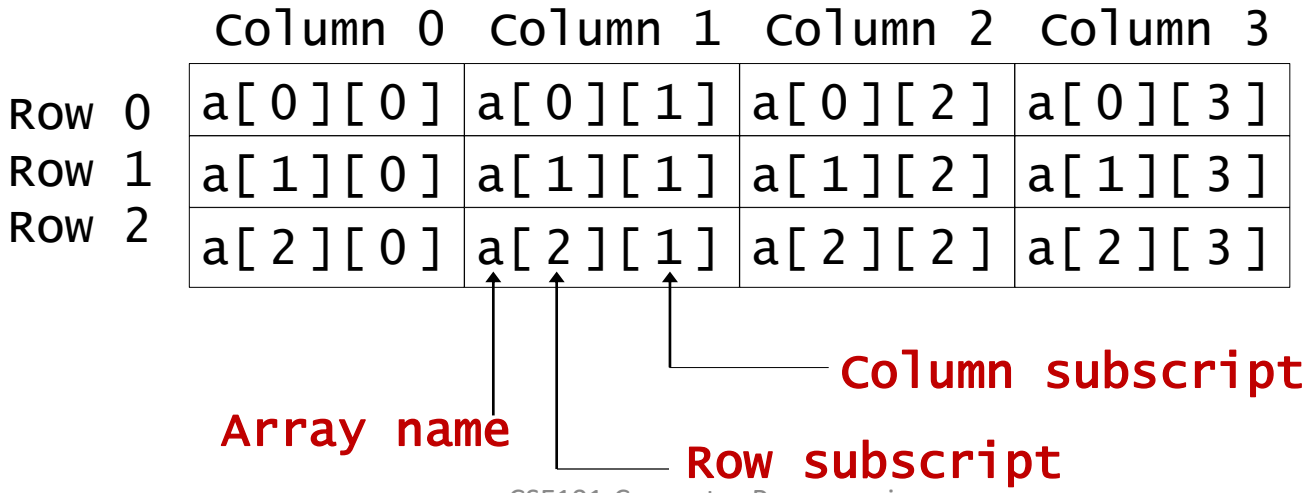
2D-Array

- The basic form of declaring a two-dimensional array of size x, y:
- Syntax:
- data_type array_name[x][y];
- data_type: Type of data to be stored. Valid C/C++ data type.
- We can declare a two dimensional integer array say ‘x’ of size 10,20 as:
- int x[10][20];
- Elements in two-dimensional arrays are commonly referred by x[i][j] where i is the row number and ‘j’ is the column number

Also known as Multiple-Subscripted Arrays

- Multiple subscripted arrays
  - Tables with rows and columns (m by n array)
  - Like matrices: specify row, then column

```
int a[rows][column];
```



Memory representation of 2D-Array

A 2D array's elements are stored in continuous memory locations. It can be represented in memory using any of the following two ways:

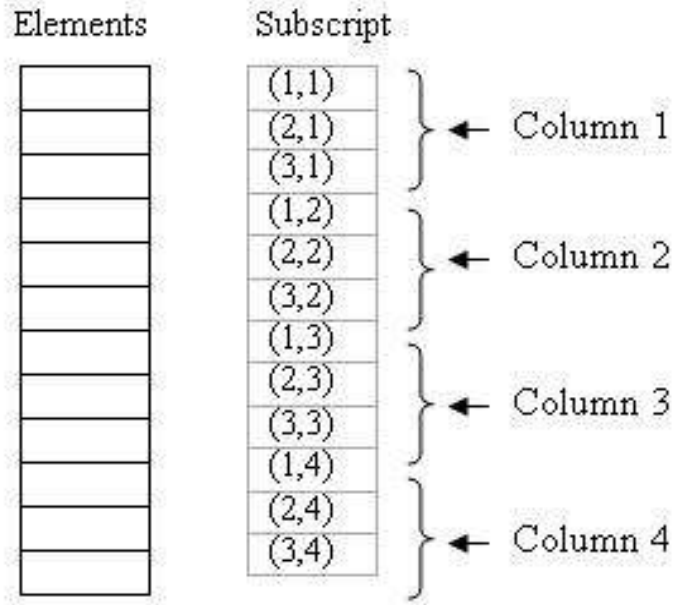
1. Column-Major Order
2. Row-Major Order

Memory representation of 2D-Array

- A 2D array's elements are stored in continuous memory locations. It can be represented in memory using any of the following two ways:

  1. Column-Major Order
  2. Row-Major Order

  1. Column-Major Order:
  In this method the elements are stored column wise, i.e. m elements of first column are stored in first m locations, m elements of second column are stored in next m locations and so on. E.g.
  A 3 x 4 array will stored as below:

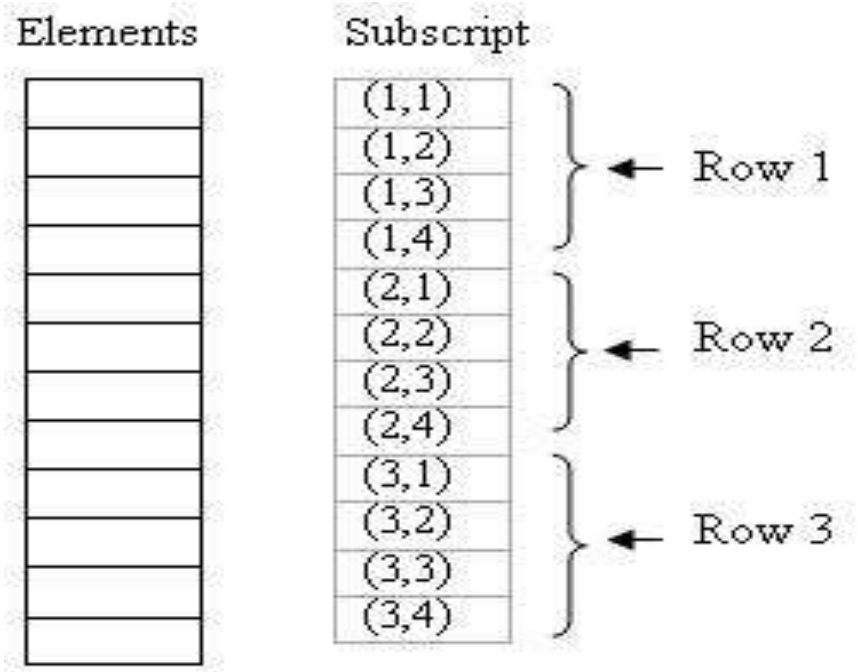


Memory representation of 2D-Array

2. Row-Major Order:

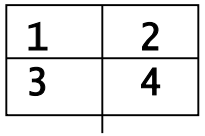
In this method the elements are stored row wise, i.e. n elements of first row are stored in first n locations, n elements of second row are stored in next n locations and so on. E.g.

A 3 x 4 array will stored as below:



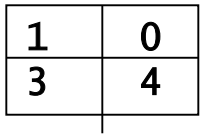
Initialization

1) Initializing at the point of declaration:



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

  Initializers grouped by row in braces
- If not enough, unspecified elements set to zero
  - `Int a[ 2 ][ 2 ] = { { 1 }, { 3, 4 } };`



- int a3[2][2]={1,2};//Remaining elements are zero
- int a4[2][2]={0};//All elements are zero

Initialization

- int a5[][2]={1,2,3};//It is possible to skip row size, if elements are initialized at the point of declaration
- int a[2][]={1,2,3};//Not possible to skip column size[Error will come]
- int a[][]={1,2,3};//Not possible to skip both row and column size[Error will come]

Initialization

2) Taking input from user

```
int a[3][3], i, j;
for(i=0; i<3; i++)
{ // for loop for rows
 for(j=0; j<3;j++)
  { // for loop for columns
   printf("enter the value ofa[%d][%d]: ", i, j);
   scanf("%d", &a[i][j]);
  } //end for columns
} //end for rows
```

```
#include<stdio.h>
int main()
{
int a[3][3], i, j;
for(i=0; i<3; i++)
{ // for loop for rows
 for(j=0; j<3;j++)
   { // for loop for columns
  printf("enter the value of a[%d][%d]: ", i, j);
  scanf("%d", &a[i][j]);
   } //end for columns
} //end for rows
printf("elements of 2D matrix are\n");
for(i=0; i<3; i++)
{
 for(j=0;j<3;j++)
   {
    printf("%d\t", a[i][j]);
   }      //end for
 printf("\n");
} //end for
return 0;
} //end main
```

Accessing(or Traversing) 2D array elements after taking input from the user

```
enter the value of a[0][1] :1
enter the value of a[0][1] :2
enter the value of a[0][1] :3
enter the value of a[0][1] :4
enter the value of a[0][1] :5
enter the value of a[0][1] :6
enter the value of a[0][1] :7
enter the value of a[0][1] :8
enter the value of a[0][1] :9
Element of 2D matrix are:
1   2   3
4   5   6
7   8   9
```

Polling Questions

What will be the output of following code?

```
#include<stdio.h>
int main()
{
int a[][3]={1,2,3,4,5,6};
printf("%d",a[0][2]);
return 0;
}
```

A. 2

B. 3

C. 4

D. Compile time error

Which of the following is invalid initialization of 2D Array?

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

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

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

D. int a[2][2]={};

What will be the output of following code?

```
#include<stdio.h>
int main()
{
   int a[3][2]={{1,2},{3,4},{5,6}};
   printf("%d",a[1][1]*a[2][1]);
   return 0;
}
```

A. 24

B. 12

C. 8

D. 20

What will be the output of following code?

```
#include <stdio.h>
    int main()
    {
        int a[2][3] = {1, 2, 3, 4, 5};
        int i = 0, j = 0;
        for (i = 0; i < 2; i++)
        for (j = 0; j < 3; j++)
        printf("%d ", a[i][j]);
        return 0;
    }
```

A. 1 2 3 4 5
B. 1 2 3 4 5 0
C. 1 2 3 4 5 Garbage value
D. Compile time error

Matrix operations using 2D arrays

- WAP to find the sum of two matrices
- WAP to display the transpose of a matrix
- WAP to find the sum of diagonal elements of a matrix
- WAP to perform multiplication of 2 matrices and display the result

WAP to find the sum of two matrices

```
#include <stdio.h>
int main()
{
  float a[3][3], b[3][3], c[3][3];
  int i, j;
  printf("Enter elements of 1st matrix\n");
  for(i=0; i<3; i++)
  {
  for(j=0; j<3 ;j++)
   {
     printf("Enter a%d%d: ", i, j);
     scanf("%f", &a[i][j]);
   }
  }
 // Taking input using nested for loop
  printf("Enter elements of 2nd matrix\n");
  for(i=0; i<3; i++)
  {
   for(j=0; j<3; j++)
   {
     printf("Enter b%d%d: ", i, j);
     scanf("%f", &b[i][j]);
   }
  }
  // adding corresponding elements of two
arrays
 for(i=0; i<3; i++)
  {
   for(j=0; j<3; j++)
   {
     c[i][j] = a[i][j] + b[i][j];
   }
  }
  // Displaying the sum
  printf("\nSum Of Matrix:\n");

  for(i=0; i<3; i++)
  {
   for(j=0; j<3; j++)
   {
     printf("%.1f\t", c[i][j]);
   }
       printf("\n");
  }
return 0;
}
```

WAP to display the transpose of a matrix

```
#include <stdio.h>
int main()
{
   int a[10][10], transpose[10][10], r, c, i, j;
   printf("Enter rows and columns of matrix:
");
   scanf("%d %d", &r, &c);

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

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

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

            printf("\n\n");
   }

   return 0;
}
```

WAP to find the sum of diagonal elements of a matrix

```
#include<stdio.h>
 int main()
 {
  int a[10][10],sum=0;
  int i,j,m,n;
  printf("Enter number of rows and column:");
  scanf("%d%d",&m,&n);
      printf("Enter Elements : ");
      for(i=0;i<m;i++)
         {
           for(j=0;j<n;j++)
              {
                scanf("%d",&a[i][j]);
              }
          }
for(i=0;i<m;i++)
          {
            for(j=0;j<n;j++)
              {
                 if(i==j)
                  {
                     sum=sum+a[i][j];
                  }
              }
          }
        printf("Sum of Diagonal Elements = %d ",sum);
 }
```

WAP to perform multiplication of 2 matrices and display the result

```
#include <stdio.h>
int main()
{
int a[10][10], b[10][10], result[10][10], r1, c1, r2, c2,
i, j, k;
   printf("Enter rows and column for first matrix: ");
   scanf("%d %d", &r1, &c1);

   printf("Enter rows and column for second matrix:
");
   scanf("%d %d",&r2, &c2);
   // Column of first matrix should be equal to
column of second matrix and
   while (c1 != r2)
   {
      printf("Error! No. of columns of first matrix not
equal to no.of row of second.\n\n");
      printf("Enter rows and column for first matrix: ");
      scanf("%d %d", &r1, &c1);
      printf("Enter rows and column for second
matrix: ");
      scanf("%d %d",&r2, &c2);
   }
   // Storing elements of first matrix.
   printf("\nEnter elements of matrix 1:\n");
   for(i=0; i<r1; i++)
   {
      for(j=0; j<c1; j++)
      {
         printf("Enter elements a%d%d: ",i,j);
         scanf("%d", &a[i][j]);
      }
   }
 // Storing elements of second matrix.
   printf("\nEnter elements of matrix 2:\n");
   for(i=0; i<r2; i++)
   {
      for(j=0; j<c2; j++)
      {
         printf("Enter elements b%d%d: ",i, j);
         scanf("%d",&b[i][j]);
      }
   }
```

```
  // Initializing all elements of result matrix to 0
   for(i=0; i<r1; i++)
   {
      for(j=0; j<c2; j++)
      {
         result[i][j] = 0;
      }
   }
   // Multiplying matrices a and b and
   // storing result in result matrix


for(i=0; i<r1; i++)
   {
      for(j=0; j<c2; j++)
      {
         for(k=0; k<c1; k++)
         {
            result[i][j]+=a[i][k]*b[k][j];
         }
      }
   }
   // Displaying the result
   printf("\nOutput Matrix:\n");
   for(i=0; i<r1; i++)
   {
      for(j=0; j<c2; j++)
      {
         printf("%d  ", result[i][j]);
      }
            printf("\n\n");
   }
   return 0;
}
```

(MATRIX MULTIPLICATION)

Dry running

Dry running

main Logic

```
for(i=0; i<r1; i++)
    for( j=0; j<c2; j++)
        result[i][j]= 0;

for( i=0; i<r1; i++)
    for(j=0; j<c2; j++)
        for(k=0; k<c1; k++)
            result[i][j] += a[i][k] * b[k][j];
```

Consider $a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ ($a[0][0]$ $a[0][1]$ / $a[1][0]$ $a[1][1]$), $b = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$ ($b[0][0]$ $b[0][1]$ / $b[1][0]$ $b[1][1]$)

r1 = 2, c1 = 2, r2 = 2, c2 = 2

Initialize all elements of result matrix to Zero:

$result = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ ($[0][0]$ $[0][1]$ / $[1][0]$ $[1][1]$)

(i) Outer for loop, i = 0, $\frac{0<2}{\text{True}}$

(j) inner for loop, j = 0, $\frac{j<2}{\text{True}}$ (0<2)

(k) innermost for loop, k = 0, $\frac{k<2}{\text{True}}$ (0<2)

result[0][0] = result[0][0] + a[0][0] * b[0][0];

result[0][0] = 0 + 1 * 5 = 0 + 5 = 5

k++ → k = 1, $\frac{1<2}{\text{True}}$

result[0][0] = result[0][0] + a[0][1] * b[1][0];

result[0][0] = 5 + 2 * 7 = 5 + 14 = 19 → result[0][0]

k++ → k = 2, 2<2 (False) → innermost k loop stops

j++ → j = 1, 1<2 (True)

k = 0, 0<2 (True)

result[0][1] = result[0][1] + a[0][0] * b[0][1]

= 0 + 1 * 6 = 0 + 6 = 6

k++, k = 1, 1<2 (True)

result[0][1] = result[0][1] + a[0][1] * b[1][1]

= 6 + 2 * 8 = 6 + 16 = 22 → result[0][1]

k++, k = 2, 2<2 (False) k loop stops

j++, → j = 2, 2<2 (False) j loop stops

i++, i = 1, 1<2 True

Similarly Next Row will be Calculated.

$result = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$

Practice Questions

- WAP to display the maximum sales done(monthly) from a particular branch in a year[Take input for number of sales persons and sales done in each month for each sales person]
- WAP to display the minimum marks scored in each subject from a particular section[Take input for marks of n no. of students in m no. of subjects from user]
- WAP to display average marks scored by n no. of students of a section in their registered m no. of courses.[Inputs will be given by user]