CL1002 – Programming Fundamentals Lab

Lab # 4

2D Dynamic Arrays and Introduction to Structures

Instructor: Muhammad Saad Rashad

Email: saad.rashad@nu.edu.pk

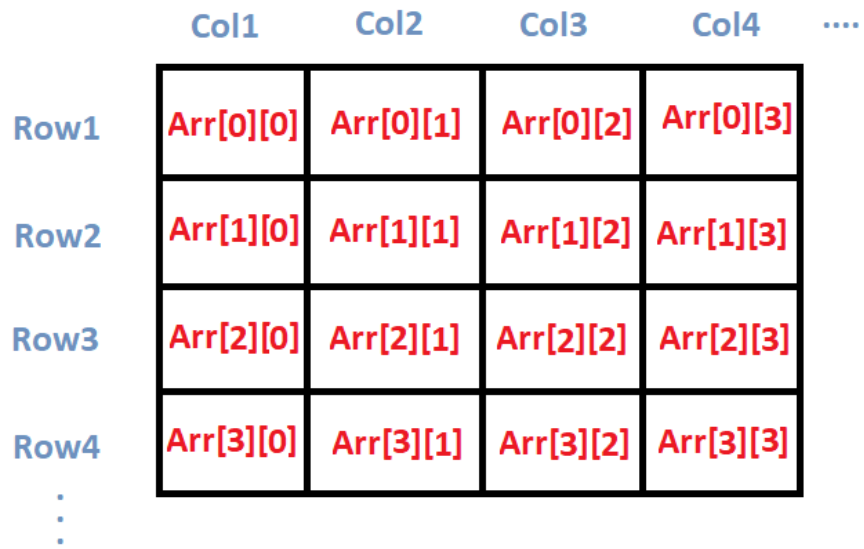
Department of Computer Science,

National University of Computer and Emerging Sciences FAST

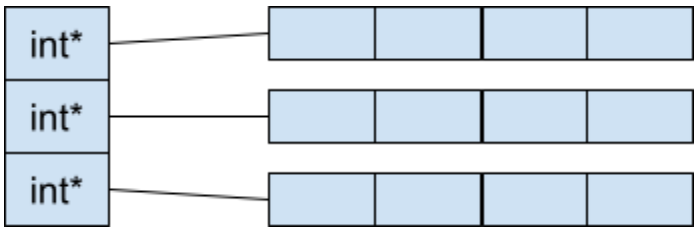
Peshawar

1. **2D Dynamic Array:**

   In general, a 2D can be defined as an array of arrays. We can imagine it as a matrix or something like shown in the image below.



   To dynamically create a 2D array we have to create an array of pointers. We can term dynamic 2D array as “pointer to pointer”.

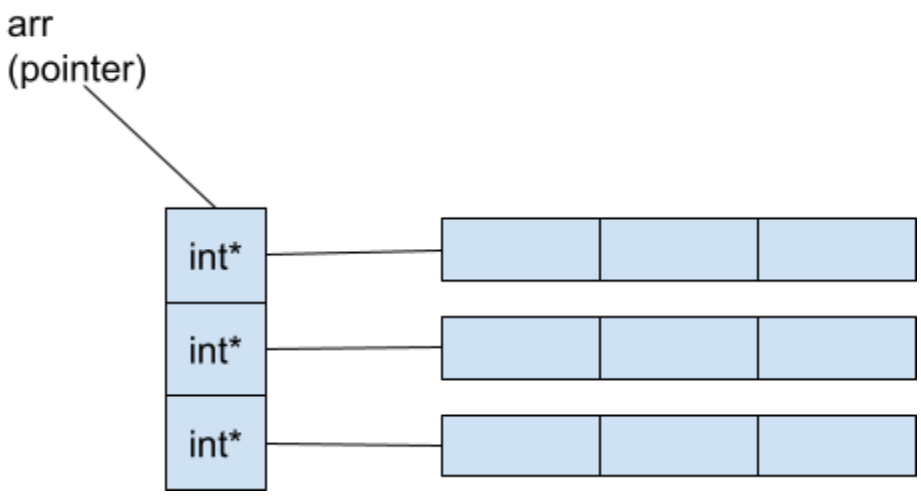


   How to create this? In the above image how many int* do we need? Let say 3
   We can write it as:
   new int*[3];

   It means that each int* is pointing to a 1D array like previously we have seen. However, we need a mechanism(a pointer) that will eventually point to the left side of the image.

   int arr = new int*[3];**

   We can imagine it something like shown in the image below:



Example 1:

Creating 2D dynamic Array:

```
#include<iostream>
using namespace std;

int main()
{
    //Step 1 create rows
    int** arr = new int*[3];

    //Step 2 create columns

    for(int i=0;i<3;i++)
    {
        arr[i]=new int[4];
    }

    // Now we have a 3 x 4 Matrix/2D array/grid/table....
}
```

Example 2:

Adding some data to the created array:

```
    cout<<"Enter the values: ";
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<4;j++)
        {
            cin>>arr[i][j];
        }
    }

    //Display the values:
        for(int i=0;i<3;i++)
    {
        for(int j=0;j<4;j++)
        {
            cout<<arr[i][j]<<"\t";
        }
        cout<<"\n";
    }
}
```

OR

```
// Input the values using pointers
    cout << "Enter the values: ";
    for(int i = 0; i < 3; i++)
    {
        for(int j = 0; j < 4; j++)
        {
            cin >> *(*(arr + i) + j); // Equivalent to arr[i][j]
        }
    }

    // Display the values using pointers
    cout << "The values are: " << endl;
    for(int i = 0; i < 3; i++)
    {
        for(int j = 0; j < 4; j++)
        {
            cout << *(*(arr + i) + j) << "\t"; // Equivalent to arr[i][j]
        }
        cout << "\n";
    }
```

Example 3:

Deallocation of the occupied memory to avoid memory leakage:

```
//Delete each row and then the pointer that is pointing to these rows

for(int i=0;i<3;i++)
{
    delete[] arr[i];
    arr[i]=NULL;        //Neutralizes dangling pointer of each row
}
delete[] arr;

arr=NULL;               //Neutralizes arr from dangling
```

Example 4:

Passing 2D array to a function:

```
#include<iostream>
using namespace std;
void addition(int**arr,int row,int col); //Function Prototype

int main()
{
    int row = 3,col=3;
    int** arr = new int*[row];

    for(int i=0;i<row;i++)
    {
        arr[i]=new int[col];
    }

    addition(arr,row,col);

    for(int i=0;i<3;i++)
    {
        delete[] arr[i];
        arr[i]=NULL;
    }

    delete[] arr;
    arr=NULL;
}
```

```
void addition(int**arr,int row,int col)
{
    cout<<"Enter the values for first array:\n";
    for(int i=0;i<row;i++)
    {
        for(int j=0;j<col;j++)
        {
            cin>>arr[i][j];
        }
    }

    int** arr2=new int*[row];

        for(int i=0;i<row;i++)
    {
        arr2[i]=new int[col];
    }

    cout<<"\nEnter the values for second array: "<<"\n";
    for(int i=0;i<row;i++)
    {
        for(int j=0;j<col;j++)
        {
            cin>>arr2[i][j];
        }
    }
```

```
    cout<<"\nSum of two array is: \n";
    for(int i=0;i<row;i++)
    {
        for(int j=0;j<col;j++)
        {
            cout<<arr[i][j] + arr2[i][j]<<"\t";
        }
        cout<<"\n";
    }

        for(int i=0;i<row;i++)
    {
        delete[] arr2[i];
        arr2[i]=NULL;       //Neutralizes dangling pointer of each row
    }
    delete[] arr2;

    arr2=NULL;

}
```

2. **Structures:**
    - In C++, a **structure** (often referred to as a struct) is a user-defined data type that allows you to group variables of different data types under a single name.
    - Structures are useful for representing real-world entities or complex data types.
    - **Example**: A "student" can be described by their name, age, roll number, and grade. These attributes can be grouped into a single unit called a "structure."
    - A structure is like a "container" or a "box" that holds related information about something.
    - Without structures, you'd need separate variables for each attribute (e.g., string name, int age, etc.), which can become messy.
    - Structures allow you to group related data into a single unit, making the code more organized and easier to manage.

Syntax:

```
struct StructureName {
dataType1 member1;
dataType2 member2;
// ...
};
```

----> Where struct is a keyword and StructureName is the user-given name of the structure.

Example 5:

1. Define a structure eg “student”
2. Declare a variable of that structure type eg “variable1”
3. Assigning values to structure members using the variable name and dot operator eg “variable1.name”
4. Access the assigned values to members using the variable name and dot operator and display them on the screen.
   E.g “cout<<variable.name”

```cpp
#include<iostream>
#include<string>
using namespace std;

struct student
{
    string name;
    int age;
    float percentage;
};

int main()
{
    student variable1;  //structure type variable

    variable1.name = "Saad"; //Assigning values to name using dot operator
    variable1.age = 20;
    variable1.percentage = 78.2;

    cout<<variable1.name<<"\n";
    cout<<variable1.age<<"\n";
    cout<<variable1.percentage<<"\n";
}
```

Tasks: USE DMA FOR ALL THE ARRAY-RELATED TASKS

1. Write a code that takes two string arrays(C-style string) and checks whether the two strings are in reverse of each other.

2. Write a code that takes a 2D string array with values {“Read”, “Bear”, “Hat”, “beat”, “ ”} swaps the values as shown in the example:

   {“Bead”, “Rear”, “Bat”, “Heat”, “ ”}

   And then calculates the total no of characters in the 2D array

3. You are given a maze represented by a two-dimensional array. Each cell in the maze can be either a wall (#) or a path (.). Write a recursive function that counts the number of paths from the starting position to the destination using a function.

4. You are building a program to perform matrix operations. Write a function that accepts a two-dimensional array (matrix) and its dimensions as input. The function should return the transpose of the matrix using pointers.

5. There is a robot on an m x n grid. The robot is initially located at the top-left corner (i.e., grid[0][0]). The robot tries to move to the bottom-right corner (i.e., grid[m - 1][n - 1]). The robot can only move either down or right at any point in time. Given the two integers m and n, return the number of possible unique paths that the robot can take to reach the bottom-right corner.

6. Write a C++ program that uses a 2D array to represent the seating chart of a movie theater, where each element of the array is either 0 for an empty seat or 1 for a reserved seat. The program prompts the user to input the row and seat number of the seat they want to reserve, check if the seat is available, and reserve it if it is. The program keeps track of the total number of reserved seats and prints it out at the end

Test Case:

```
Enter row and seat number to reserve (0 0 to stop): 1
1
Seat reserved successfully.
Enter row and seat number to reserve (0 0 to stop): 29
2
Invalid row or seat number. Please try again.
Enter row and seat number to reserve (0 0 to stop): 2
9
Seat reserved successfully.
Enter row and seat number to reserve (0 0 to stop): 10
2
```

```
Enter row and seat number to reserve (0 0 to stop): 12
3
Invalid row or seat number. Please try again.
Enter row and seat number to reserve (0 0 to stop): 11
19
Invalid row or seat number. Please try again.
Enter row and seat number to reserve (0 0 to stop): 11
8
Invalid row or seat number. Please try again.
Enter row and seat number to reserve (0 0 to stop): 10
30
Invalid row or seat number. Please try again.
Enter row and seat number to reserve (0 0 to stop): 10
9
Seat reserved successfully.
```

7. **Write a C++ Program to Perform Matrix Multiplication**

- The program should take two matrices, A and B, as input.
- Check if the number of columns in matrix A is equal to the number of rows in matrix B.
- If not, display an error message: "Matrices cannot be multiplied. Number of columns in A must be equal to number of rows in B." and exit the program.
- If the matrices can be multiplied, compute the product matrix C and display it.
- The program should handle matrices of any valid size (within reasonable limits).

8. Create a Structure for a Book

a. Define a structure named Book with the following members:

- title (a string to store the title of the book)
- author (a string to store the author's name)
- pages (an integer to store the number of pages)
- price (a float to store the price of the book)

b. Write a main() function to:

- Create an instance of the Book structure.
- Prompt the user to input the details of the book (title, author, pages, and price).
- Display the details of the book.