DATA STRUCTURE AND ALGORITHM

LAB -2

SEPTEMBER 1, 2025

Sharjeel Memon (24k-0555)

Solutions

Task#1

```
#include <iostream>
                                                            // Easy Access to Input and
Output Operations


using namespace std;
                                                            // So we don't have to write
"std::" before every library feature (like cout, cin, endl, etc.)


class TwoDMatrix {
                                                                    // Defining the
TwoDMatrix Class

private:

  int** matrix;
                                                            // Data Member to hold the
2D array (matrix)

  int row;
                                                                    // Number of rows in
the matrix

  int col;
                                                                    // Number of columns
in the matrix


public:

  // Constructor to initialize the matrix

  TwoDMatrix(int row, int col) : row(row), col(col) {
                        // Parameterized Constructor

    cout << "Matrix Initialized!" << endl;
                                        // Print message when matrix is initialized
```

```
        matrix = new int*[row];
                                                          // Allocate memory for the rows

        for (int i = 0; i < row; ++i) {
                                                  // Loop through rows

            matrix[i] = new int[col]{0};
                                                  // Initialize each column with 0

        }

    }


    // Destructor to deallocate memory

    ~TwoDMatrix() {
                                                                              // Destructor to free up
memory when the object is destroyed

        for (int i = 0; i < row; i++) {
                                                  // Loop through each row

            delete[] matrix[i];
                                                          // Deallocate memory for each row

        }

        delete[] matrix;
                                                                  // Deallocate the memory for the
matrix itself

    }


    // Function to resize the matrix

    void ResizingMatrix(int newRow, int newCol, int value) {
                                  // Resizes the matrix to new dimensions

        if (row == newRow && col == newCol) return;
                                          // No resizing needed if the dimensions are the same

        else {

            // Resize when the new size is smaller
```

```
if (row > newRow || col > newCol) {
                        // If the new size is smaller, resize accordingly

  int** temp = new int*[newRow];
                               // Allocate memory for the new matrix

  for (int i = 0; i < newRow; i++) {
                        // Loop through the rows

    temp[i] = new int[newCol];
                        // Allocate memory for the columns

  }

  // Copy the old matrix to the new matrix

  for (int i = 0; i < newRow; i++) {
                        // Loop through the rows of the new matrix

    for (int j = 0; j < newCol; j++) {
                // Loop through the columns of the new matrix

      temp[i][j] = matrix[i][j];
                // Copy each element

    }

  }

  // Deallocate old matrix

  for (int i = 0; i < row; i++) {
                // Loop through the rows

    delete[] matrix[i];
                        // Free the memory for each row

  }

  delete[] matrix;
                                       // Free the memory for the entire matrix

  // Update matrix pointer and sizes

  row = newRow;
                                               // Update row size

  col = newCol;
                                       // Update column size
```

```
        matrix = temp;
                                                                  // Update the matrix pointer to the
new matrix

        return;
                                                                  // End of function

      }

      // Resize when the new size is larger, initialize with the given value

      if (row < newRow || col < newCol) {
                                          // If the new size is larger, initialize new cells with a given
value

        int** temp = new int*[newRow];
                                                        // Allocate memory for the new matrix

        for (int i = 0; i < newRow; i++) {
                                          // Loop through the rows

          temp[i] = new int[newCol];
                                          // Allocate memory for the columns

          // Initialize new cells with the given value

          for (int j = 0; j < newCol; j++) {
                                   // Loop through the columns

            temp[i][j] = value;
                                          // Set the value for the new cell

          }

        }

        // Copy the old matrix to the new matrix

        for (int i = 0; i < row; i++) {
                                   // Loop through the rows of the old matrix

          for (int j = 0; j < col; j++) {
                            // Loop through the columns of the old matrix

            temp[i][j] = matrix[i][j];
                                   // Copy each element

          }
        }

        // Deallocate old matrix

        for (int i = 0; i < row; i++) {
                      // Loop through the rows

          delete[] matrix[i];
                              // Free the memory for each row

        }

        delete[] matrix;
                                              // Free the memory for the entire matrix

        // Update matrix pointer and sizes

        row = newRow;
                                                      // Update row size

        col = newCol;
                                              // Update column size

        matrix = temp;
                                                      // Update the matrix pointer to the
new matrix

        return;
                                                      // End of function

      }

    }

  }


  // Function to transpose the matrix

  void Transpose() {
                                                              // Transpose the matrix (rows
become columns and vice versa)

    int** transposed = new int*[col];
                                      // Allocate memory for the transposed matrix

    for (int i = 0; i < col; i++) {
                              // Loop through the columns of the original matrix
      transposed[i] = new int[row];
                                                // Allocate memory for the rows of the transposed
matrix
    }

    // Transpose the matrix
    for (int i = 0; i < row; i++) {
                                                // Loop through the rows of the original matrix
      for (int j = 0; j < col; j++) {
                              // Loop through the columns of the original matrix
        transposed[j][i] = matrix[i][j];
                              // Swap the elements (i,j) with (j,i)
      }
    }

    // Deallocate old matrix
    for (int i = 0; i < row; i++) {
                                                // Loop through the rows
      delete[] matrix[i];
                                                          // Free the memory for each row
    }
    delete[] matrix;
                                                                    // Free the memory for the entire
matrix

    // Update matrix and row/col sizes
    matrix = transposed;
                                                                    // Update the matrix pointer to the
transposed matrix
    int temp = row;
                                                                    // Swap the row and column sizes
```

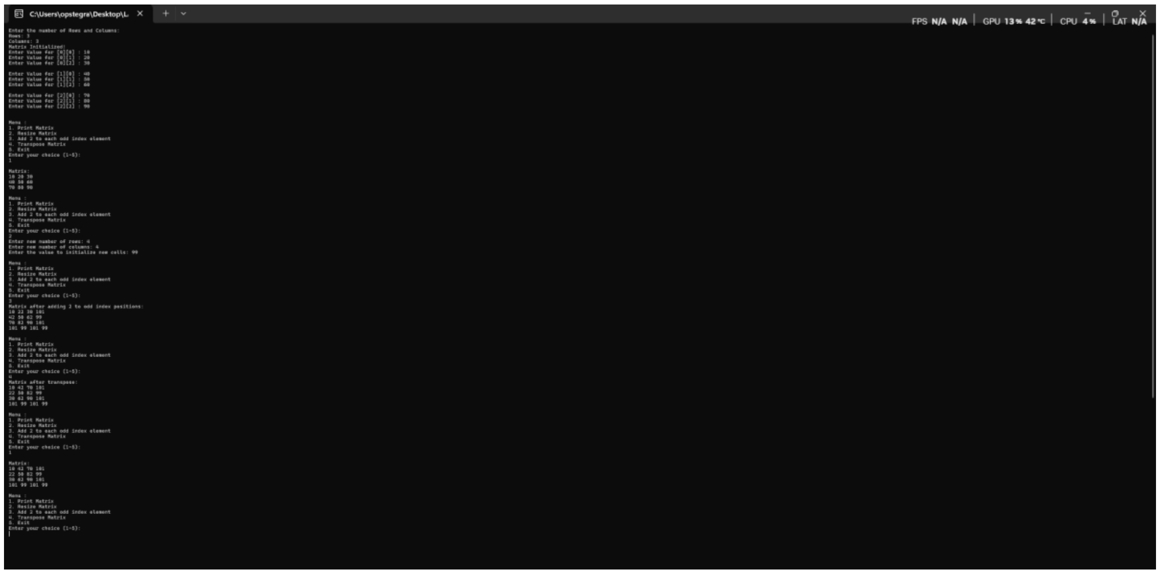
```
        row = col;

        col = temp;
                                                            // Swap rows and columns for
the transposed matrix

    }


    // Function to print the matrix

    void PrintMatrix() {
                                                      // Print the matrix to the console

        for (int i = 0; i < row; i++) {
                                          // Loop through the rows

            for (int j = 0; j < col; j++) {
                                    // Loop through the columns

                cout << matrix[i][j] << " ";
                                    // Print each element

            }

            cout << endl;
                                                      // Print a new line after each row

        }

    }


    // Function to set values in the matrix

    void SetValue() {
                                                            // Set the value of each
element in the matrix

        for (int i = 0; i < row; i++) {
                                          // Loop through the rows

            for (int j = 0; j < col; j++) {
                                    // Loop through the columns

                cout << "Enter Value for [" << i << "][" << j << "] : ";
               // Prompt user for input
            cin >> matrix[i][j];
                                                // Get the input from the user
          }
          cout << endl;
                                                        // Print a new line after each row
       }
    }


    // Function to add 2 to each odd index element and print
    void AddTwoToOddIndex() {
                                                            // Add 2 to elements at odd
indices
       for (int i = 0; i < row; i++) {
                                                // Loop through the rows
          for (int j = 0; j < col; j++) {
                                            // Loop through the columns
             if ((i + j) % 2 != 0) {
                                            // Odd index position
                matrix[i][j] += 2;
                                                // Add 2 to the element
             }
          }
       }
    }


//    // Filler function to initialize the matrix with increasing values
//    void Filler() {
                                                            // Initialize the matrix with
increasing values starting from 10
//       int value = 10;
                                                        // Start with 10
//      for (int i = 0; i < row; i++) {
                                                        // Loop through the rows
//          for (int j = 0; j < col; j++) {
                                              // Loop through the columns
//              matrix[i][j] = value;
                                                        // Set the value for each element
//              value += 10;
                                                                        // Increase the value by 10 for the
next element
//          }
//      }
//   }
};


// Function to handle input validation for row and column values
void GetValidInput(int &row, int &col) {
                                                                // Validate the row and column input
   while (true) {
       cout << "Rows: ";
                                                                        // Prompt user for the number of
rows
       cin >> row;
                                                                                // Get the input
       cout << "Columns: ";
                                                                        // Prompt user for the number of
columns
       cin >> col;
                                                                        // Get the input


       if (row <= 0 || col <= 0) {
                                                        // Check if the input is invalid
```

```
            cout << "Invalid input! Rows and Columns must be positive integers." << endl;

            cout << "Please enter again!" << endl;
                                                // Ask user to re-enter the values

        } else {

            break;
                                                                    // Exit the loop if the input is
valid

        }

    }

}


int main() {

    int row = 0, col = 0;


    // Get valid input for rows and columns

    cout << "Enter the number of Rows and Columns:" << endl;

    GetValidInput(row, col);


    // Create the TwoDMatrix object

    TwoDMatrix matrix(row, col);

    matrix.SetValue();
                                                                    // Set values for the matrix


    // Menu loop for operations

    int choice;

    do {

        cout << endl<<"Menu :"<<endl;

        cout << "1. Print Matrix" <<endl;
        cout << "2. Resize Matrix "<< endl;

        cout << "3. Add 2 to each odd index element" << endl;

        cout << "4. Transpose Matrix" << endl;

        cout << "5. Exit" <<endl;

        cout << "Enter your choice (1-5): "<<endl;

        cin >> choice;


        switch (choice) {

            case 1:

              cout << endl << "Matrix:" << endl;

              matrix.PrintMatrix();
                                                         // Print the matrix

              break;


            case 2: {

              int newRow, newCol, value;

              cout << "Enter new number of rows: ";
                                                // Prompt user for new rows

              cin >> newRow;

              cout << "Enter new number of columns: ";
                                         // Prompt user for new columns

              cin >> newCol;

              cout << "Enter the value to initialize new cells: ";
              // Prompt user for value to initialize new cells

              cin >> value;

              if (newRow <= 0 || newCol <= 0) {
                                                // Check if the new size is invalid

               cout << "Invalid Size!" << endl;
```

```
        break;

    }

    matrix.ResizingMatrix(newRow, newCol, value);
                              // Resize the matrix

    break;

}


case 3:

    matrix.AddTwoToOddIndex();
                                                      // Add 2 to odd index elements

    cout << "Matrix after adding 2 to odd index positions:"<<endl;

    matrix.PrintMatrix();
                                       // Print the updated matrix

    break;


case 4:

    matrix.Transpose();
                                       // Transpose the matrix

    cout << "Matrix after transpose:"<<endl;

    matrix.PrintMatrix();
                                       // Print the transposed matrix

    break;


case 5:

    cout << "Exiting program..."<<endl;
                              // Exit the program

    break;
```

```
        default:

            cout << "Invalid choice! Please try again."<<endl;
                        // Invalid menu choice

            break;

        }

    } while (choice != 5);



    return 0;

}
```

Q1. Screenshots



```
C:\Users\opstegra\Desktop\L
Enter the number of Rows and Columns:
Rows: 3
Columns: 3
Matrix Initialized!
Enter Value for [0][0] : 10
Enter Value for [0][1] : 20
Enter Value for [0][2] : 30

Enter Value for [1][0] : 40
Enter Value for [1][1] : 50
Enter Value for [1][2] : 60

Enter Value for [2][0] : 70
Enter Value for [2][1] : 80
Enter Value for [2][2] : 90


Menu :
1. Print Matrix
2. Resize Matrix
3. Add 2 to each odd index element
4. Transpose Matrix
5. Exit
Enter your choice (1-5):
1

Matrix:
10 20 30
40 50 60
70 80 90

Menu :
1. Print Matrix
2. Resize Matrix
3. Add 2 to each odd index element
4. Transpose Matrix
5. Exit
Enter your choice (1-5):
2
Enter new number of rows: 4
Enter new number of columns: 4
Enter the value to initialize new cells: 99

Menu :
1. Print Matrix
2. Resize Matrix
3. Add 2 to each odd index element
4. Transpose Matrix
5. Exit
Enter your choice (1-5):
3
Matrix after adding 2 to odd index positions:
10 22 30 101
42 50 62 99
70 82 90 101
101 99 101 99

Menu :
1. Print Matrix
2. Resize Matrix
3. Add 2 to each odd index element
4. Transpose Matrix
5. Exit
Enter your choice (1-5):
4
Matrix after transpose:
10 42 70 101
22 50 82 99
30 62 90 101
101 99 101 99

Menu :
1. Print Matrix
2. Resize Matrix
3. Add 2 to each odd index element
4. Transpose Matrix
5. Exit
Enter your choice (1-5):
1

Matrix:
10 42 70 101
22 50 82 99
30 62 90 101
101 99 101 99

Menu :
1. Print Matrix
2. Resize Matrix
3. Add 2 to each odd index element
4. Transpose Matrix
5. Exit
Enter your choice (1-5):
```

Task#2

```
#include <iostream>

                                                // Easy Access to Input and Output
Operations
```

```
using namespace std;

                                                  // So we dont have to write "std::" before
every library feature (like cout, cin, endl, etc.)


int main() {


                                             int row;

                                                                                     // Declare row
variable to store number of students
                                             int col;

                                                                                     // Declare col
variable to store number of subjects


                                             // Prompt user for input

                                             cout << "Enter Number of rows :";

                                                              // Prompt user for the number of
rows (students)
                                             cin >> row;

                                                                             // Store the user input
in the row variable


                                             cout << "Enter Number of Columns :";

                                                              // Prompt user for the number of
columns (subjects)
                                             cin >> col;

                                                                             // Store the user input
in the col variable
```

```
// Dynamically allocate memory for student marks
int** arr = new int*[row];
                    // Dynamically allocate
memory for 2D array to store student marks
int* total = new int[row];
                    // Dynamically allocate
memory for total marks of each student
int* subavg = new int[col];
            // Dynamically allocate memory for
sum of marks for each subject

// Allocate memory for each student's subjects
for(int i = 0; i < row; i++) {
            // Loop through rows (students)
    arr[i] = new int[col];
                    // Dynamically allocate
memory for columns (subjects) for each student
}

// Initialize total array to zero for each student
for(int i = 0; i < row; i++) {
            // Initialize total marks for each
student
    total[i] = 0;
                            // Set total marks of
each student to 0 initially
}
```

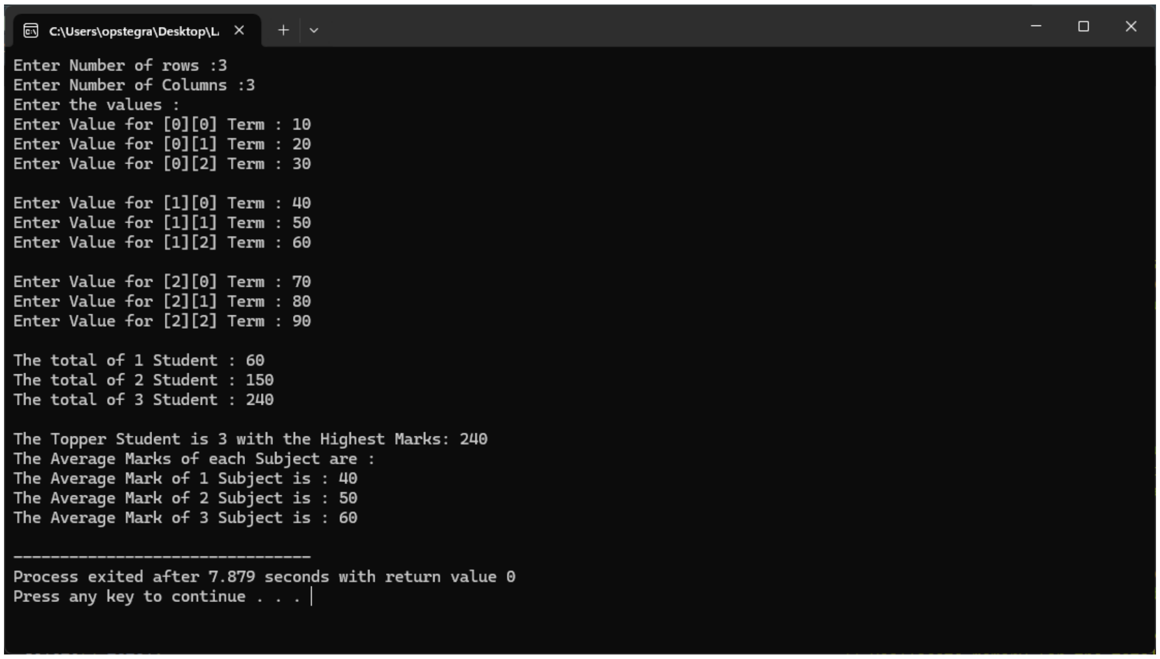
```
                                        // Initialize subavg array to zero for each subject
                                        for(int i = 0; i < col; i++) {
                                                        // Initialize total for each subject
                                            subavg[i] = 0;
                                                                            // Set sum of each
subject's marks to 0 initially
                                        }

/*                                      //Only for Quick Testing
                                        int filler=10;
                                        for(int i=0;i<row;i++) {
                                            for(int j=0;j<col;j++) {
                                                        arr[i][j] =filler;
                                                        filler   += 10;
                                            }
                                        }

*/

                                        // Input marks for each student
                                        cout << "Enter the values :" << endl;
                                                        // Prompt user to enter the marks for
each student
                                        for(int i = 0; i < row; i++) {
                                                        // Loop through each student (row)
                                        for(int j = 0; j < col; j++) {

                                             // Loop through each subject
(column)

                                             cout << "Enter Value for [" << i << "]["
<< j << "] Term : ";                                   // Prompt user to
enter marks for each student in each subject

                                             cin >> arr[i][j];

                                                       // Store the entered
value into the array

                                             if(arr[i][j]<0 ) arr[i][j]=0;

                                        }

                                        cout << endl;

                                                       // Move to the next
line after entering marks for all subjects of a student

                                    }


                                    // Calculate total marks for each student

                                    for(int i = 0; i < row; i++) {

                                             // Loop through each student (row)

                                        for(int j = 0; j < col; j++) {

                                             // Loop through each subject
(column)

                                             total[i] += arr[i][j];

                                                  // Add the marks for each
subject to the student's total

                                        }

                                    }
```

```
                                        // Output total marks for each student

                                        for(int i = 0; i < row; i++) {

                                                    // Loop through each student

                                             cout << "The total of " << i + 1 << " Student : "
<< total[i] << endl;                                        // Output the total marks of
each student

                                        }

                                        cout << endl;

                                                                                 // Print a
newline for better readability



                                        // Find the highest total marks among students

                                        int highest = total[0];

                                        int topperStudent = 0;  // Variable to store the
topper student

                                        for (int i = 0; i < row; i++) {

  if (total[i] > highest) {

                                        // Checking for Highest

    highest = total[i];

    topperStudent = i;  // Update the topper student

  }

}

                                        cout << "The Topper Student is " << topperStudent
+ 1 << " with the Highest Marks: " << highest << endl;




                                        // Output the average marks of each subject
```

```
cout << "The Average Marks of each Subject are :"
<< endl;
// Output header for average marks
for(int i = 0; i < row; i++) {
        // Loop through each student (row)
    for(int j = 0; j < col; j++) {
        // Loop through each subject
(column)
        subavg[j] += arr[i][j];
        // Add each student's marks for a
subject to the subject's total
    }
}


// Output average marks for each subject
for(int i = 0; i < col; i++) {
        // Loop through each subject
(column)
    subavg[i] = subavg[i] / row;
        // Calculate average by dividing the
total marks for a subject by the number of students
    cout << "The Average Mark of " << i + 1 << "
Subject is : " << subavg[i] << endl;      // Output the average marks for each subject
}


// Free the dynamically allocated memory
```

```
                                             for(int i = 0; i < row; i++) {

                                                       // Loop through each row (student)

                                                 delete[] arr[i];

                                                             // Deallocate memory for
each student's subject marks

                                             }

                                             delete[] arr;

                                                                   // Deallocate memory
for the array of student rows

                                             delete[] total;

                                                                   // Deallocate memory
for the total marks array

                                             delete[] subavg;

                                                                   // Deallocate memory
for the subject averages array


                                             return 0;

                                                                         // End of
program

}
```

Q2. Screenshots



```
Enter Number of rows :3
Enter Number of Columns :3
Enter the values :
Enter Value for [0][0] Term : 10
Enter Value for [0][1] Term : 20
Enter Value for [0][2] Term : 30

Enter Value for [1][0] Term : 40
Enter Value for [1][1] Term : 50
Enter Value for [1][2] Term : 60

Enter Value for [2][0] Term : 70
Enter Value for [2][1] Term : 80
Enter Value for [2][2] Term : 90

The total of 1 Student : 60
The total of 2 Student : 150
The total of 3 Student : 240

The Topper Student is 3 with the Highest Marks: 240
The Average Marks of each Subject are :
The Average Mark of 1 Subject is : 40
The Average Mark of 2 Subject is : 50
The Average Mark of 3 Subject is : 60

--------------------------------
Process exited after 7.879 seconds with return value 0
Press any key to continue . . .
```

Task#3

```
#include <iostream>

                                                        // Easy Access to Input and Output
Operations



using namespace std;

                                                        // So we don't have to write "std::"
before every library feature (like cout, cin, endl, etc.)



class Student {

                                                                // Defining the Student class

private:

  int id;

                                                                        // Student ID

  double* marks;
```

```
                                                            // Pointer to store marks for
subjects
  const int subjects = 5;

                                                    // Number of subjects (fixed to 5 as
per the task)
  static int studentcount;

                                            // Static variable to keep track of the student
count across all instances


public:
  // Default constructor for Student class
  Student() : id(studentcount++) {
                                                                              //
Initialize student ID with the static counter
    marks = new double[subjects];
                                                                              //
Dynamically allocate memory for marks array
  }


  // Destructor to deallocate memory
  ~Student() {

                                                            // Destructor to delete
dynamically allocated marks array
    delete[] marks;

                                                 // Delete the dynamically allocated marks
array
  }


  // Function to set marks for the student
  void setmarks() {

                                                     // Set the marks for the student

    cout << "Enter Marks for Student " << id  << " : " << endl;
                                               // Prompt user to enter marks for the student

    for (int i = 0; i < subjects; i++) {
                                                                // Loop through each
subject

      cout << "Subject [" << i + 1 << "]: ";
                                                                // Display the subject
number

      cin >> marks[i];

                                            // Take input for marks

      if (marks[i] < 0) marks[i] = 0;
                                                                // Ensure no negative
marks

    }

  }


  // Function to calculate and return the average marks of the student

  double getAverage() {

                                               // Calculate average marks

    double sum = 0;

                                               // Variable to store the sum of marks

    for (int i = 0; i < subjects; i++) {
                                                                // Loop through all
subjects

      sum += marks[i];

                                            // Add marks to the sum

    }
```

```
        return sum / subjects;

                                                   // Return the average marks

    }


    // Function to get the highest mark of the student

    double getHighest() {

                                                        // Get the highest mark

        double highest = marks[0];

                                                   // Start with the first subject's mark

        for (int i = 1; i < subjects; i++) {
                                                                                 // Loop through all
subjects

            if (marks[i] > highest) {
                                                                                                //
Update the highest mark

                highest = marks[i];

            }

        }

        return highest;

                                                        // Return the highest mark

    }


    // Function to get the lowest mark of the student

    double getLowest() {

                                                        // Get the lowest mark

        double lowest = marks[0];

                                                   // Start with the first subject's mark
```

```
};

        for (int i = 1; i < subjects; i++) {
                                                                            // Loop through all
subjects
            if (marks[i] < lowest) {
                                                                                        //
Update the lowest mark
                lowest = marks[i];
            }
        }
        return lowest;
                                                            // Return the lowest mark
    }


    // Function to auto-fill marks (filler function)
    //void filler() {
                                                            // Auto-fill marks with dummy values
    //    double value = 10.0;
                                                // Set starting mark for each subject
    //    for (int i = 0; i < subjects; i++) {
                                                                            // Loop through each
subject
    //        marks[i] = value;
                                                // Assign value to each subject
    //        value += 10;
                                                    // Increment value for the next subject
    //    }
    //}
};
```

```
// Static variable initialization
int Student::studentcount = 1;
                                        // Initialize static student count to 0

class Department {
                                                        // Defining the Department
class
private:
   int deptid;
                                                        // Department ID
   Student* students;
                                                   // Pointer to dynamically store
students in the department
   int numstudent;
                                                   // Number of students in the
department
   static int deptcount;
                                              // Static variable to keep track of department
count

public:
   // Constructor for Department class (no parameters)
   Department() : deptid(deptcount++) {
                                                                  // Initialize
department ID with static counter
```

```
        cout << "Enter the number of students for Department " << deptid  << ": ";
                                                                                  // Prompt user for number of students in this
department

        cin >> numstudent;

                                                                                  // Get the number of students

        if (numstudent <= 0) {

                                                                        // If invalid (less than or equal to 0)

            numstudent = 0;

                                                                        // Set to 0 students

        }

        students = new Student[numstudent];
                                                                                                                                    // Dynamically
allocate memory for students in the department

    }


    // Destructor to deallocate memory

    ~Department() {

                                                                                              // Destructor to delete dynamically
allocated students array

        delete[] students;

                                                                                  // Delete the dynamically allocated students
array

    }


    // Function to set marks for all students in the department

    void setMarksForAllStudents() {
                                                                                                                                                  // Set
marks for all students in the department
```

```
        for (int i = 0; i < numstudent; i++) {
                                                                                    // Loop through all
students
                                                                    cout << "Enter Marks for Students in
Department " << deptid  << " : " << endl;          // Department-wise prompt
                                                                    students[i].setmarks();
                                                        //students[i].filler();
                                                                    // Auto-fill marks for each student (or
replace with setmarks() for manual entry)
        }
    }

    // Function to print department results
    void printDepartmentResults() {
                                                                                                   //
Print the results of all students in the department
        double highest = students[0].getHighest();
                                                                                           // Start with
the first student's highest mark
        double lowest = students[0].getLowest();
                                                                                    // Start with the first
student's lowest mark
        double totalAverage = 0;
                                                        // Variable to store total average of the
department

        for (int i = 0; i < numstudent; i++) {
                                                                                    // Loop through all
students
            totalAverage += students[i].getAverage();
                                                                                    // Add the average of
each student to total average
```

```
         if (students[i].getHighest() > highest) {
                                                                         // Find the highest mark in
the department
             highest = students[i].getHighest();
         }
         if (students[i].getLowest() < lowest) {
                                                                         // Find the lowest mark in the
department
             lowest = students[i].getLowest();
         }
       }
       cout << "Department " << deptid << " Results:" << endl;
                                                                  // Display department results
       cout << "Highest Mark in Department: " << highest << endl;
                                                                         // Print highest mark in the
department
       cout << "Lowest Mark in Department: " << lowest << endl;
                                                                         // Print lowest mark in the
department
       cout << "Average Marks in Department: " << totalAverage / numstudent << endl;
                                                           // Print average marks of the department
   }
};


// Static variable initialization
int Department::deptcount = 1;
                                                      // Initialize static department count to 0


class University {
```

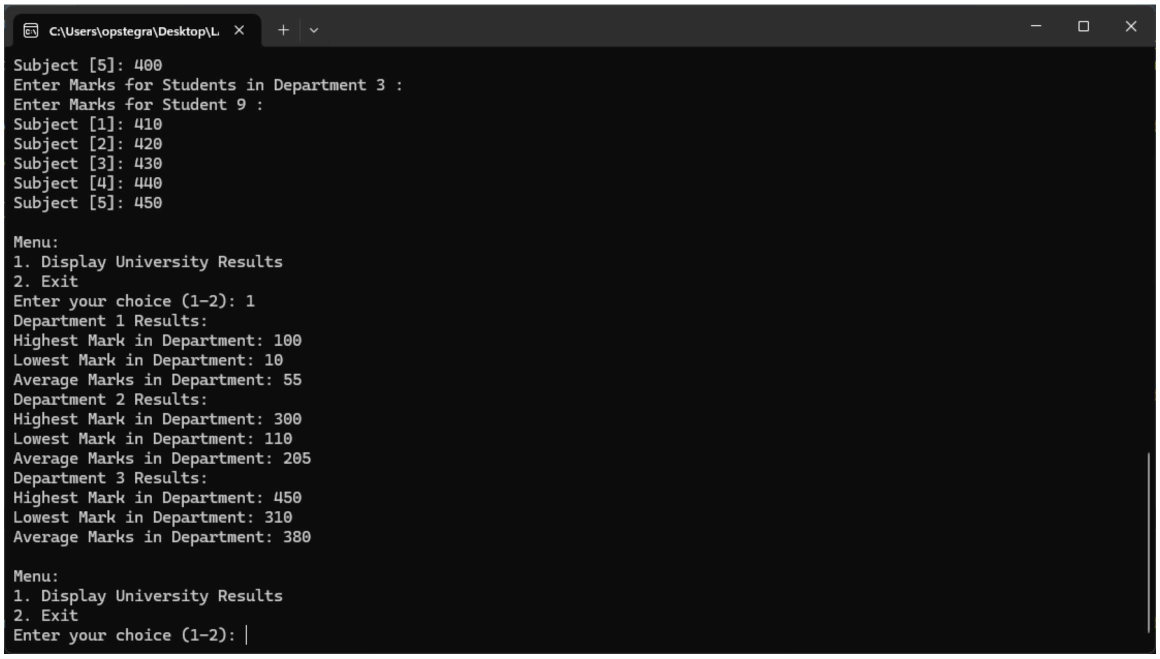
```
                                                            // Defining the University
class

private:

   Department* departments;

                                          // Pointer to hold multiple departments in the
university

   int numDepartments;

                                               // Number of departments in the university


public:

   // Constructor to initialize the University with departments

   University(int numDepts) : numDepartments(numDepts) {
                                                                   // Initialize the
number of departments

      departments = new Department[numDepts];
                                                                          // Dynamically
allocate memory for departments

      for (int i = 0; i < numDepts; i++) {
                                                                   // Loop through all
departments

         cout << "Enter Students for Department " << i + 1 << " : " << endl;
                                          // Prompt user for students in each department

         departments[i].setMarksForAllStudents();
                                                                   // Set marks for all
students in the department

      }

   }


   // Destructor to deallocate memory

   ~University() {
```

```
                                                    // Destructor to free memory for
departments
    delete[] departments;
                                                // Delete the dynamically allocated departments
array
  }

  // Function to calculate and display results for all departments
  void displayUniversityResults() {
                                                                                                    //
Display results of all departments in the university
    for (int i = 0; i < numDepartments; i++) {
                                                                    // Loop through all
departments
      departments[i].printDepartmentResults();
                                                                    // Print results for
each department
    }
  }
};

int main() {
                                                                    // Main function
where the menu and program logic resides
  int numDepartments;
                                                    // Variable to store the number of
departments
  cout << "Enter number of departments: ";
                                                                            // Prompt user
for number of departments
```

```
    cin >> numDepartments;
                                                              // Get the number of departments

    University university(numDepartments);
                                                                                                   //
Create a University object with the specified number of departments

    // Menu loop for operations
    int choice;
    do {
        cout << endl<<"Menu:"<<endl;
        cout << "1. Display University Results"<<endl;
                                                                             // Option to display results of
the university
        cout << "2. Exit"<<endl;
                                                         // Option to exit the program
        cout << "Enter your choice (1-2): ";
        cin >> choice;

        switch (choice) {
                                                              // Switch case for menu options
            case 1:
                university.displayUniversityResults();
                                                                             // Display results of all
departments in the university
                break;

            case 2:
```

```
            cout << "Exiting program..."<<endl;
                                                                    // Exit the program

            break;


         default:

            cout << "Invalid choice! Please try again."<<endl;
                                                       // Invalid menu choice

            break;

      }

   } while (choice != 2);

                                                       // Continue until the user chooses to exit


   return 0;

                                                                          // End of program

}
```

Q3 . Screenshots



C:\Users\opstegra\Desktop\L

```
Subject [5]: 400
Enter Marks for Students in Department 3 :
Enter Marks for Student 9 :
Subject [1]: 410
Subject [2]: 420
Subject [3]: 430
Subject [4]: 440
Subject [5]: 450

Menu:
1. Display University Results
2. Exit
Enter your choice (1-2): 1
Department 1 Results:
Highest Mark in Department: 100
Lowest Mark in Department: 10
Average Marks in Department: 55
Department 2 Results:
Highest Mark in Department: 300
Lowest Mark in Department: 110
Average Marks in Department: 205
Department 3 Results:
Highest Mark in Department: 450
Lowest Mark in Department: 310
Average Marks in Department: 380

Menu:
1. Display University Results
2. Exit
Enter your choice (1-2):
```