

**DSA (Data Structure and Algorithms) Lab**

**LAB REPORT # 7**

**Semester**: 3rdSemester

**Section**: C

**Submitted To:**

**Abdullah Shahrose**

**Submitted By:**

**Name**: Abdul Ahad

**Roll No**: 22-CS-071

**Task 1:**

#include <iostream>

using namespace std;

int binary\_search(int arr[], int low, int high, int target)

{

    if (low > high)

        return -1;

    int mid = (low + high) / 2;

    if (arr[mid] == target)

        return mid;

    else if (arr[mid] > target)

        return binary\_search(arr, low, mid - 1, target);

    else

        return binary\_search(arr, mid + 1, high, target);

}

int main()

{

    int arr[] = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19};

    int target = 7;

    int index = binary\_search(arr, 0, sizeof(arr) / sizeof(arr[0]) - 1, target);

    cout << "\n\n";

    if (index != -1)

        cout << "The target element is found at index " << index << endl;

    else

        cout << "The target element is not found in the array" << endl;

    cout << "\n\n";

    return 0;

}

**Output:**



**Task 2:**

#include <iostream>

#include <vector>

using namespace std;

int main()

{

    vector<int> numbers = {5, 2, 9, 1, 7, 3};

    int max = numbers[0];

    int min = numbers[0];

    for (int i = 1; i < numbers.size(); i++)

    {

        if (numbers[i] > max)

        {

            max = numbers[i];

        }

        if (numbers[i] < min)

        {

            min = numbers[i];

        }

    }

    cout << "\n\n";

    cout << "Maximum value: " << max << endl;

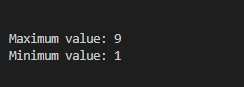
    cout << "Minimum value: " << min << endl;

    cout << "\n\n";

    return 0;

}

**Output:**



**Task 3:**

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

int countOnes(int \*arr, int size)

{

    int count = 0;

    int low = 0;

    int high = size - 1;

    while (low <= high)

    {

        int mid = low + (high - low) / 2;

        if (arr[mid] == 1)

        {

            count = size - mid;

            high = mid - 1;

        }

        else

        {

            low = mid + 1;

        }

    }

    return count;

}

int main()

{

    srand(time(0));

    const int SIZE = 15;

    int \*arr = new int[SIZE];

    for (int i = 0; i < SIZE; i++)

    {

        arr[i] = rand() % 2;

    }

    int onesCount = countOnes(arr, SIZE);

    cout << "\n\n";

    cout << "Number of 1's in the array: " << onesCount << endl;

    cout << "\n\n";

    return 0;

}

**Output:**

****

**Task 4:**

#include <iostream>

using namespace std;

class Matrix

{

private:

    int \*\*A;

    int size;

    string name;

public:

    Matrix(int size = 3)

    {

        this->name = name;

        this->size = size;

        A = new int \*[size];

        for (int i = 0; i < size; i++)

        {

            A[i] = new int[size];

        }

        for (int i = 0; i < size; i++)

        {

            for (int j = 0; j < size; j++)

            {

                A[j][i] = 0;

            }

        }

    }

    void setdata(int i, int j, int value)

    {

        A[i][j] = value;

    }

    void getdata()

    {

        cout << "Enter the values for the matrix : " << endl;

        for (int i = 0; i < this->size; i++)

        {

            cout << "Column " << i + 1 << " : " << endl;

            for (int j = 0; j < this->size; j++)

            {

                cout << "Row " << j + 1 << " : ";

                std::cin >> A[j][i];

            }

            cout << endl;

        }

    }

    void display()

    {

        int i = 0, j = 0;

        cout << "Values for the matrix : \n"

             << endl;

        for (int i = 0; i < this->size; i++)

        {

            cout << "  |";

            for (int j = 0; j < this->size; j++)

            {

                cout << "  ";

                cout << A[i][j];

            }

            cout << "  |";

            cout << endl;

        }

    }

    friend std::ostream &operator<<(std::ostream &o, Matrix const &M)

    {

        int i = 0, j = 0;

        o << "Values for the matrix : \n"

          << endl;

        for (int i = 0; i < M.size; i++)

        {

            o << "  |";

            for (int j = 0; j < M.size; j++)

            {

                o << "  ";

                o << M.A[i][j];

            }

            o << "  |";

            o << endl;

        }

        return o;

    }

    Matrix operator\*(Matrix &M)

    {

        Matrix m;

        int val;

        for (int i = 0; i < size; i++)

        {

            for (int j = 0; j < size; j++)

            {

                val = 0;

                for (int k = 0; k < size; k++)

                {

                    val += this->A[i][k] \* M.A[k][j];

                }

                m.setdata(j, i, val);

            }

        }

        return m;

    }

    Matrix &operator=(const Matrix &other)

    {

        if (this == &other)

        {

            return \*this;

        }

        // Deallocate the memory for the current matrix

        for (int i = 0; i < size; i++)

        {

            delete[] A[i];

        }

        delete[] A;

        // Allocate memory for the new matrix

        size = other.size;

        A = new int \*[size];

        for (int i = 0; i < size; i++)

        {

            A[i] = new int[size];

        }

        // Copy the values from the other matrix

        for (int i = 0; i < size; i++)

        {

            for (int j = 0; j < size; j++)

            {

                A[i][j] = other.A[i][j];

            }

        }

        return \*this;

    }

    Matrix operator!()

    {

        Matrix mM;

        for (int i = 0; i < size; i++)

        {

            for (int j = 0; j < size; j++)

            {

                mM.setdata(j, i, this->A[i][j]);

            }

        }

        return mM;

    }

    Matrix operator-(Matrix &M)

    {

        Matrix m;

        for (int i = 0; i < size; i++)

        {

            for (int j = 0; j < size; j++)

            {

                m.setdata(j, i, (this->A[j][i] - M.A[j][i]));

            }

        }

        return m;

    }

    Matrix operator+(Matrix &M)

    {

        Matrix m;

        for (int i = 0; i < size; i++)

        {

            for (int j = 0; j < size; j++)

                m.setdata(j, i, (this->A[j][i] + M.A[j][i]));

        }

        return m;

    }

    ~Matrix()

    {

        for (int i = 0; i < size; i++)

            delete[] A[i];

        delete[] A;

    }

};

int main()

{

    system("cls");

    Matrix M1, M2;

    int choice = -1;

    while (true)

    {

    check:

        system("cls");

        cout << "\t\t\t  MAIN MENU\n\t\t\t~~~~~~~~~~~~\n";

        cout << "1 - Read Matrix - A" << endl;

        cout << "2 - Read Matrix - B" << endl;

        cout << "3 - Addition of Matrix - A and Matrix - B" << endl;

        cout << "4 - Difference of Matrix - A and Matrix - B" << endl;

        cout << "5 - Product of Matrix - A and Matrix - B" << endl;

        cout << "6 - Transpose of Matrix - A or Matrix - B" << endl;

        cout << "7 - Exit Program";

        cout << "\n\nYour Choice ---->  ";

        std::cin >> choice;

        switch (choice){

        case 1:

            M1.display();

            cout << "\nIf want to change the matrix values press 1 else 0 : ";

            std::cin >> choice;

            switch (choice)

            {

            case 1:

                M1.getdata();

                break;

            default:

                break;

            }

            break;

        case 2:

            M2.display();

            cout << "\nIf want to change the matrix values press 1 else 0 : ";

            std::cin >> choice;

            switch (choice)

            {

            case 1:

                M2.getdata();

                break;

            }

            break;

        case 3:

            cout << (M1 + M2) << endl<< endl;

            break;

        case 4:

            cout << (M1 - M2) << endl<< endl;

            break;

        case 5:

            cout << (M1 \* M2) << endl<<endl;

            break;

        case 6:

            cout << "Enter 1 for Matrix A and 2 for Matrix B : ";

            std::cin >> choice;

            switch (choice)

            {

            case 1:

                cout << "The matrix is : ";

                cout << M1 << endl;

                cout << "The transpose matrix is : ";

                cout << !M1 << endl;

                break;

            case 2:

                cout << "The matrix is : ";

                cout << M2 << endl;

                cout << "The transpose matrix is : ";

                cout << !M2 << endl;

                break;

            }

            break;

        case 7:

            exit(0);

            break;

        default:

            cout << "Wrong Input : ";

            goto check;

        }

        std::cin.ignore();

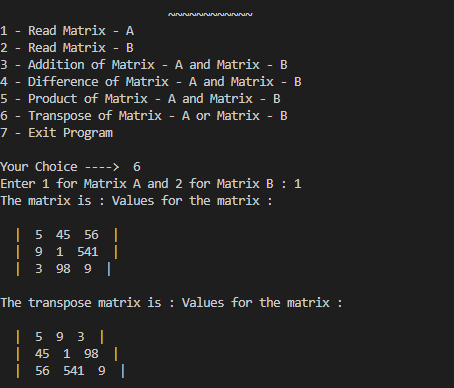
        std::cin.ignore();

    }

    return 0;

}

**Output:**



**Task 5:**

#include <iostream>

using namespace std;

class Matrix

{

private:

    int \*\*A;

    int size;

    string name;

public:

    Matrix(int size = 3)

    {

        this->name = name;

        this->size = size;

        A = new int \*[size];

        for (int i = 0; i < size; i++)

        {

            A[i] = new int[size];

        }

        for (int i = 0; i < size; i++)

        {

            for (int j = 0; j < size; j++)

            {

                A[j][i] = 0;

            }

        }

    }

    void setdata(int i, int j, int value)

    {

        A[i][j] = value;

    }

    int linearSearch()

    {

        int maxVal = A[0][0];

        for (int i = 0; i < size; i++)

        {

            for (int j = 0; j < size; j++)

            {

                if (A[i][j] > maxVal)

                {

                    maxVal = A[i][j];

                }

            }

        }

        return maxVal;

    }

    void getdata()

    {

        cout << "Enter the values for the matrix : " << endl;

        for (int i = 0; i < this->size; i++)

        {

            cout << "Column " << i + 1 << " : " << endl;

            for (int j = 0; j < this->size; j++)

            {

                cout << "Row " << j + 1 << " : ";

                std::cin >> A[j][i];

            }

            cout << endl;

        }

    }

    void display()

    {

        int i = 0, j = 0;

        cout << "Values for the matrix : \n"

             << endl;

        for (int i = 0; i < this->size; i++)

        {

            cout << "  |";

            for (int j = 0; j < this->size; j++)

            {

                cout << "  ";

                cout << A[i][j];

            }

            cout << "  |";

            cout << endl;

        }

    }

    friend std::ostream &operator<<(std::ostream &o, Matrix const &M)

    {

        int i = 0, j = 0;

        o << "Values for the matrix : \n"

          << endl;

        for (int i = 0; i < M.size; i++)

        {

            o << "  |";

            for (int j = 0; j < M.size; j++)

            {

                o << "  ";

                o << M.A[i][j];

            }

            o << "  |";

            o << endl;

        }

        return o;

    }

    Matrix operator\*(Matrix &M)

    {

        Matrix m;

        int val;

        for (int i = 0; i < size; i++)

        {

            for (int j = 0; j < size; j++)

            {

                val = 0;

                for (int k = 0; k < size; k++)

                {

                    val += this->A[i][k] \* M.A[k][j];

                }

                m.setdata(j, i, val);

            }

        }

        return m;

    }

    Matrix &operator=(const Matrix &other)

    {

        if (this == &other)

        {

            return \*this;

        }

        // Deallocate the memory for the current matrix

        for (int i = 0; i < size; i++)

        {

            delete[] A[i];

        }

        delete[] A;

        // Allocate memory for the new matrix

        size = other.size;

        A = new int \*[size];

        for (int i = 0; i < size; i++)

        {

            A[i] = new int[size];

        }

        // Copy the values from the other matrix

        for (int i = 0; i < size; i++)

        {

            for (int j = 0; j < size; j++)

            {

                A[i][j] = other.A[i][j];

            }

        }

        return \*this;

    }

    Matrix operator!()

    {

        Matrix mM;

        for (int i = 0; i < size; i++)

        {

            for (int j = 0; j < size; j++)

            {

                mM.setdata(j, i, this->A[i][j]);

            }

        }

        return mM;

    }

    Matrix operator-(Matrix &M)

    {

        Matrix m;

        for (int i = 0; i < size; i++)

        {

            for (int j = 0; j < size; j++)

            {

                m.setdata(j, i, (this->A[j][i] - M.A[j][i]));

            }

        }

        return m;

    }

    Matrix operator+(Matrix &M)

    {

        Matrix m;

        for (int i = 0; i < size; i++)

        {

            for (int j = 0; j < size; j++)

            {

                m.setdata(j, i, (this->A[j][i] + M.A[j][i]));

            }

        }

        return m;

    }

    ~Matrix()

    {

        for (int i = 0; i < size; i++)

        {

            delete[] A[i];

        }

        delete[] A;

    }

};

int main()

{

    system("cls");

    Matrix M1, M2;

    int choice = -1;

    while (true)

    {

    check:

        system("cls");

        cout << "\t\t\t  MAIN MENU\n\t\t\t~~~~~~~~~~~~\n";

        cout << "1 - Read Matrix - A" << endl;

        cout << "2 - Read Matrix - B" << endl;

        cout << "3 - Addition of Matrix - A and Matrix - B" << endl;

        cout << "4 - Difference of Matrix - A and Matrix - B" << endl;

        cout << "5 - Product of Matrix - A and Matrix - B" << endl;

        cout << "6 - Transpose of Matrix - A or Matrix - B" << endl;

        cout << "7 - Difference of Matrix - A and Matrix - B" << endl;

        cout << "8 - Exit Program";

        cout << "\n\nYour Choice ---->  ";

        std::cin >> choice;

        switch (choice)

        {

        case 1:

            M1.display();

            cout << "\nIf want to change the matrix values press 1 else 0 : ";

            std::cin >> choice;

            switch (choice)

            {

            case 1:

                M1.getdata();

                break;

            default:

                break;

            }

            break;

        case 2:

            M2.display();

            cout << "\nIf want to change the matrix values press 1 else 0 : ";

            std::cin >> choice;

            switch (choice)

            {

            case 1:

                M2.getdata();

                break;

            }

            break;

        case 3:

            cout << (M1 + M2) << endl

                 << endl;

            break;

        case 4:

            cout << (M1 - M2) << endl

                 << endl;

            break;

        case 5:

            cout << (M1 \* M2) << endl

                 << endl;

            break;

        case 6:

            cout << "Enter 1 for Matrix A and 2 for Matrix B : ";

            std::cin >> choice;

            switch (choice)

            {

            case 1:

                cout << "The matrix is : \n";

                cout << M1 << endl;

                cout << "The transpose matrix is : \n";

                cout << !M1 << endl;

                break;

            case 2:

                cout << "The matrix is : \n";

                cout << M2 << endl;

                cout << "The transpose matrix is : \n";

                cout << !M2 << endl;

                break;

            }

            break;

        case 7:

            cout << "The Subtraction of the Matrices is : \n";

            cout << M1 - M2;

            cout << "The Max value of the subtracted matrix is : ";

            cout << (M1 - M2).linearSearch();

            cout << "\n\n";

            break;

        case 8:

            exit(0);

            break;

        default:

            cout << "Wrong Input : ";

            goto check;

        }

        std::cin.ignore();

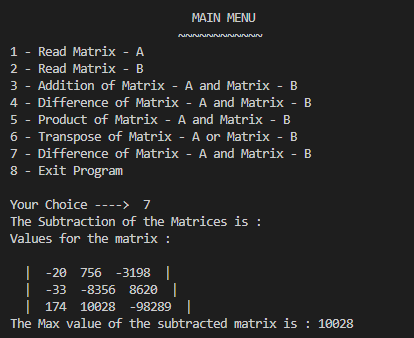
        std::cin.ignore();

    }

    return 0;

}

**Output:**

****