Stony Brook University
College of Engineering and Applied Science

ESE 224.L02

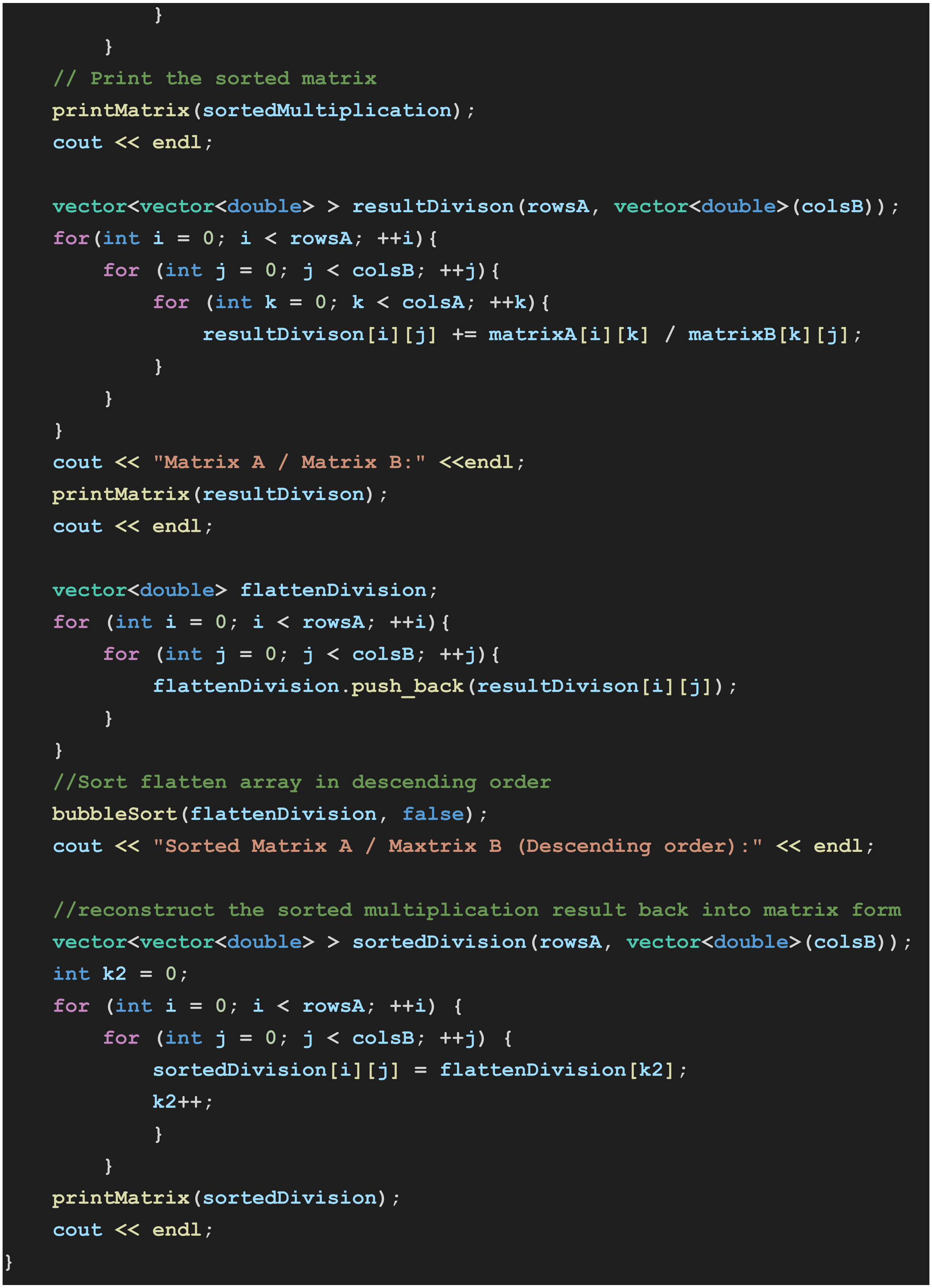
Lab 6

Ryan Lin

Professor: Xin Wang

Task 1
Main.cpp

```
int main(){
    int rowsA, colsA, rowsB, colsB;

    //input matrix A and dimension
    cout << "Enter amount of rows for matrix A: ";
    cin >> rowsA;
    cout << "\nEnter amount of colums for matrix A: ";
    cin >> colsA;

    //input matrix B and dimension
    cout << "\nEnter amount of rows for matrix B: ";
    cin >> rowsB;
    cout << "\nEnter amount of colums for matrix B: " ;
    cin >> colsB;

    if (colsA != rowsB){
        cout << "\nThe matrix does not match, can't perform multiplication" << endl ;
        return 1;
    }
    //input matrix A
    vector<vector<double> > matrixA(rowsA, vector<double>(colsA));
    cout << "\nEnter the elements for matrix A: " <<endl;
    for (int i =0; i < rowsA; ++i){
        for(int j = 0; j < colsA; ++j){
            cin >> matrixA[i][j];
        }
    }
    cout <<"Matrix A is: " << endl;
    printMatrix(matrixA);
    //input matrix B
    vector<vector<double> > matrixB(rowsB, vector<double>(colsB));
    cout << "Enter the elements for matrix B: " <<endl;
    for (int i =0; i < rowsB; ++i){
        for(int j = 0; j < colsB; ++j){
            cin >> matrixB[i][j];
        }
    }
    cout <<"Matrix B is: " << endl;
    printMatrix(matrixB);

    //Perform matrix multiplication
    vector<vector<double> > resultMultiplication(rowsA,
vector<double>(colsB));
    for(int i = 0; i < rowsA; ++i){
        for (int j = 0; j < colsB; ++j){
            for (int k = 0; k < colsA; ++k){
                resultMultiplication[i][j] += matrixA[i][k] *
matrixB[k][j];
            }
        }
    }

    cout << "Matrix A * Matrix B:" <<endl;
    printMatrix(resultMultiplication);
    cout << endl;

    //Flatten the multiplication result into 1d array for bubble sort
    vector<double> flattenMultiplication;
    for (int i = 0; i < rowsA; ++i){
        for (int j = 0; j < colsB; ++j){
            flattenMultiplication.push_back(resultMultiplication[i][j]);
        }
    }

    //Sort flatten array in ascending order
    bubbleSort(flattenMultiplication, true);
    cout << "Sorted Matrix A * Maxtrix B (Ascending order):" << endl;

    //reconstruct the sorted multiplication result back into matrix form
    vector<vector<double> > sortedMultiplication(rowsA,
vector<double>(colsB));
    int k1 = 0;
    for (int i = 0; i < rowsA; ++i) {
        for (int j = 0; j < colsB; ++j) {
            sortedMultiplication[i][j] = flattenMultiplication[k1];
            k1++;
```



```
            }
        }
    // Print the sorted matrix
    printMatrix(sortedMultiplication);
    cout << endl;

    vector<vector<double> > resultDivison(rowsA, vector<double>(colsB));
    for(int i = 0; i < rowsA; ++i){
        for (int j = 0; j < colsB; ++j){
            for (int k = 0; k < colsA; ++k){
                resultDivison[i][j] += matrixA[i][k] / matrixB[k][j];
            }
        }
    }
    cout << "Matrix A / Matrix B:" <<endl;
    printMatrix(resultDivison);
    cout << endl;

    vector<double> flattenDivision;
    for (int i = 0; i < rowsA; ++i){
        for (int j = 0; j < colsB; ++j){
            flattenDivision.push_back(resultDivison[i][j]);
        }
    }
    //Sort flatten array in descending order
    bubbleSort(flattenDivision, false);
    cout << "Sorted Matrix A / Maxtrix B (Descending order):" << endl;

    //reconstruct the sorted multiplication result back into matrix form
    vector<vector<double> > sortedDivision(rowsA, vector<double>(colsB));
    int k2 = 0;
    for (int i = 0; i < rowsA; ++i) {
        for (int j = 0; j < colsB; ++j) {
            sortedDivision[i][j] = flattenDivision[k2];
            k2++;
            }
        }
    printMatrix(sortedDivision);
    cout << endl;
}
```

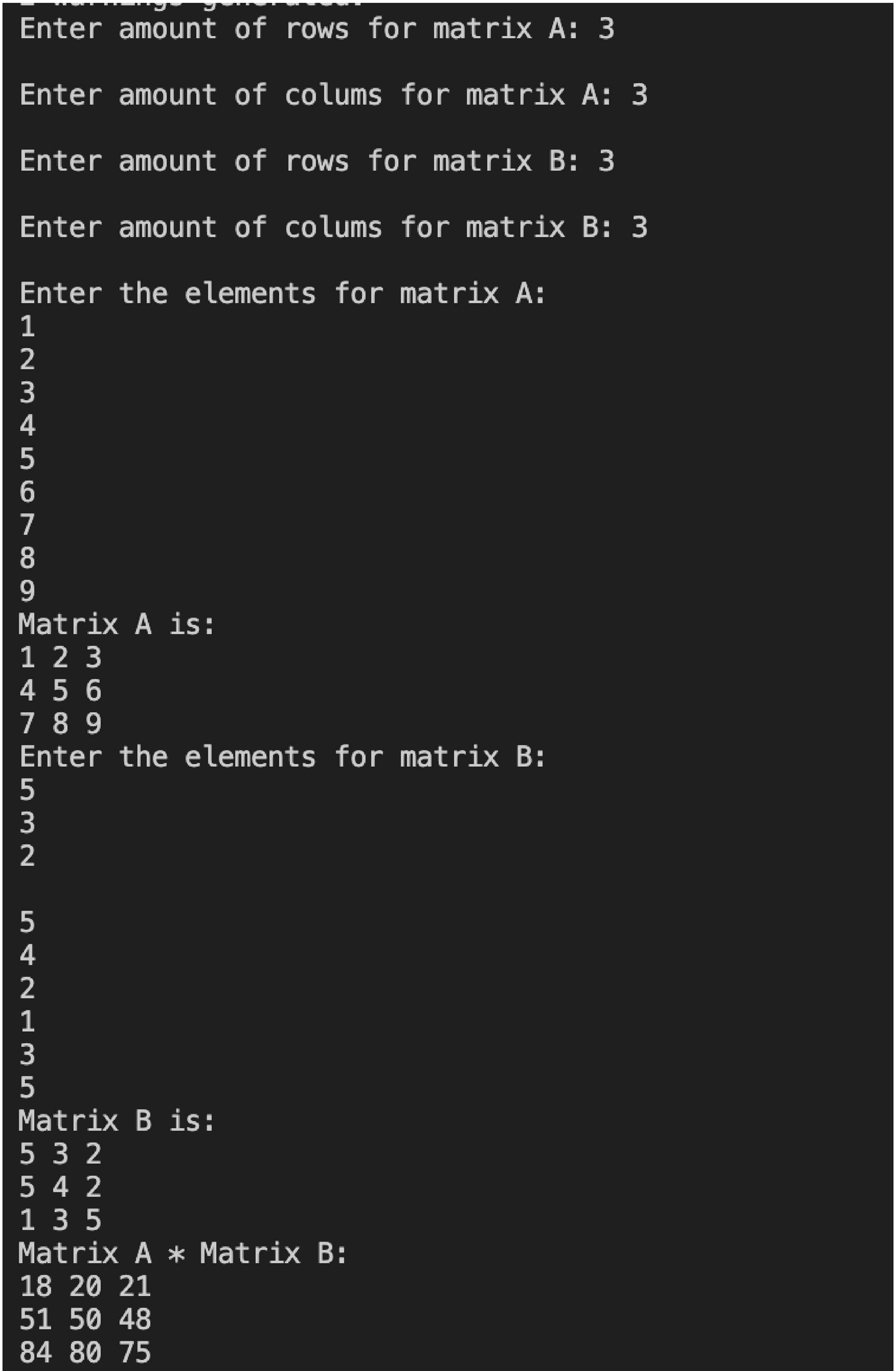
Matrix.cpp

```
void bubbleSort(vector<double>& arr, bool ascending)
{
    int i, j;
    bool swapped;
    if(ascending ==1){
        for (i=0; i<arr.size() - 1; i++){
            swapped = false;
            for(j=0; j<arr.size()- i - 1; j++){
                if(arr[j] > arr[j+1]){
                    swap(arr[j],arr[j+1]);
                    swapped = true;
                }
            }
            if (swapped == false)
            break;
        }
    }
    else{
        for (i=0; i <arr.size() - 1; i++){
            swapped = false;
            for(j=0; j<arr.size() - i - 1; j++){
                if(arr[j] < arr[j+1]){
                    swap(arr[j], arr[j+1]);
                    swapped = true;
                }
            }
            if (swapped == false)
            break;
        }
    }
}

vector<vector<double> > transposeMatrix(vector<vector<double> > matrix){
    int rows = matrix.size();
    int cols = matrix[0].size();

     // Create a new matrix to store the transpose
    vector<vector<double> > transposed(cols, vector<double>(rows,0));
     // Transpose the matrix
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            transposed[j][i] = matrix[i][j];
        }
    }
    return transposed;
}

void printMatrix(vector<vector<double> >& matrix){
    for (vector<double> row : matrix){
        for (double num : row){
            cout << setprecision(3) << num << " ";
        }
        cout << endl;
    }
}
```

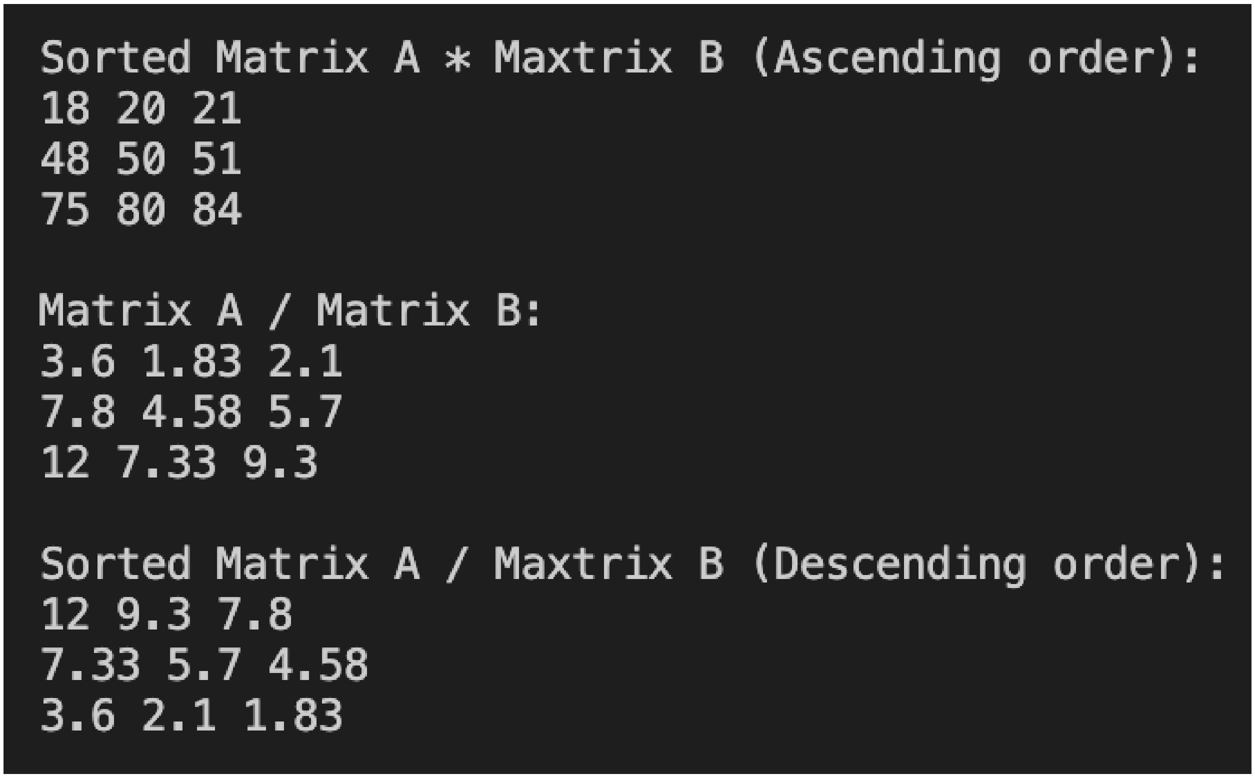
Matrix.h

```
#ifndef MATRIX_H
#define MATRIX_H
#include <vector>
using namespace std;


void bubbleSort(vector<double>& arr, bool ascending);
vector<vector<double> > transposeMatrix(vector<vector<double> > matrix);
void printMatrix(vector<vector<double> >& matrix);
#endif
```

Output:



```
2 warnings generated.
Enter amount of rows for matrix A: 3

Enter amount of colums for matrix A: 3

Enter amount of rows for matrix B: 3

Enter amount of colums for matrix B: 3

Enter the elements for matrix A:
1
2
3
4
5
6
7
8
9
Matrix A is:
1 2 3
4 5 6
7 8 9
Enter the elements for matrix B:
5
3
2

5
4
2
1
3
5
Matrix B is:
5 3 2
5 4 2
1 3 5
Matrix A * Matrix B:
18 20 21
51 50 48
84 80 75
```



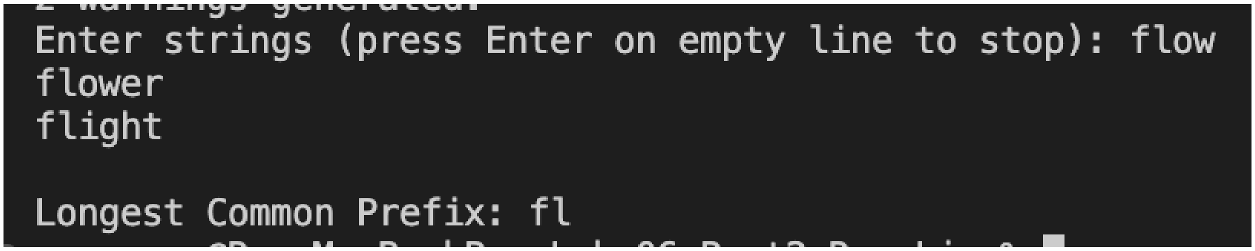
```
Sorted Matrix A * Maxtrix B (Ascending order):
18 20 21
48 50 51
75 80 84

Matrix A / Matrix B:
3.6 1.83 2.1
7.8 4.58 5.7
12 7.33 9.3

Sorted Matrix A / Maxtrix B (Descending order):
12 9.3 7.8
7.33 5.7 4.58
3.6 2.1 1.83
```

Task 2:

Task2.cpp

```
string longestCommonPrefix(vector<string>& strs){
    if (strs.empty()){
        return "";
    }
    int minLen = INT_MAX;
    for(const string& str : strs){
        minLen = min(minLen, static_cast<int>(str.size()));
    }
    for (int i =0; i < minLen; ++i){
        char sameChar = strs[0][i];
        for (const string& str : strs){
            if ( str[i] != sameChar){
                return str.substr(0,i);
            }
        }
    }
    return strs[0].substr(0,minLen);
}

int main(){
    vector<string> strs;
    string str;
    cout << "Enter strings (press Enter on empty line to stop): ";
    while (true){
        getline(cin, str);
        if(str.empty()){
            break;
        }
        strs.push_back(str);
    }
    if(strs.empty()){
        cout << "No input strings provided." <<endl;
    }
    else{
        string commonPrefix = longestCommonPrefix(strs);
        cout << "Longest Common Prefix: " << commonPrefix << endl;
    }
    return 0;Task2.cpp
```

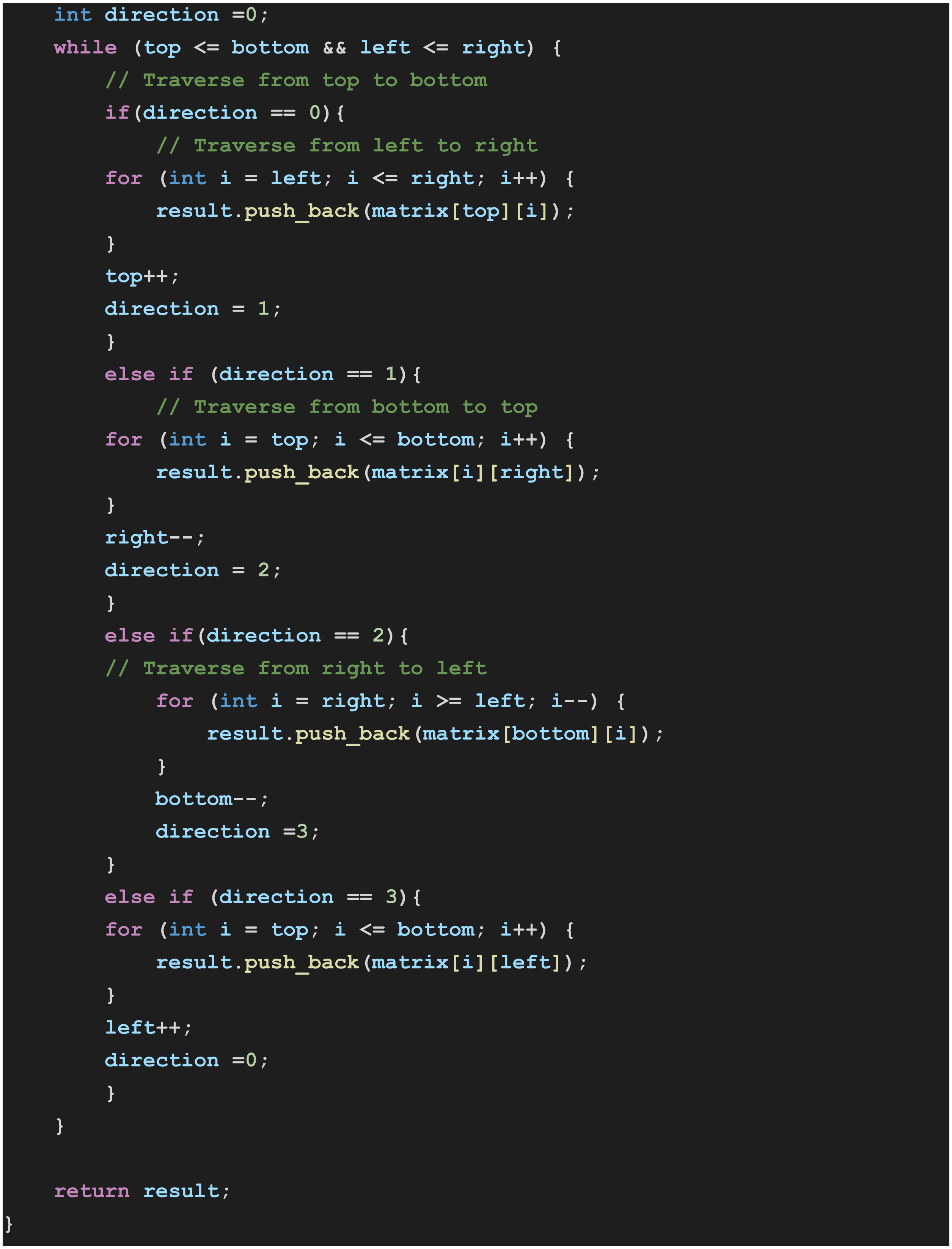
Task2 output:



```
2 warnings generated.
Enter strings (press Enter on empty line to stop): flow
flower
flight

Longest Common Prefix: fl
```

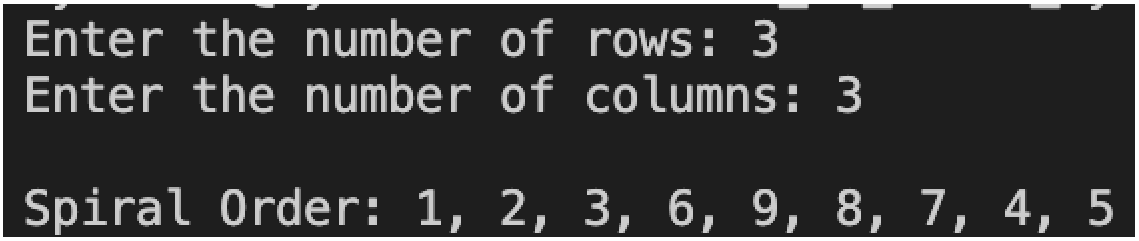
Task3
task3.cpp

```
int main() {
    int m, n;
    cout << "Enter the number of rows: ";
    cin >> m;
    cout << "Enter the number of columns: ";
    cin >> n;


    // populate the 2d matrix
    vector<vector<int> > matrix;
    int count  = 1;
    for (int i = 0; i < m; i++) {
        vector<int> temp;
        for (int j = 0; j < n; j++) {
            temp.push_back(count++);
        }
        matrix.push_back(temp);
    }
vector<int> result = createMatrixAndGetSpiralOrder(matrix);

    cout << "\nSpiral Order: ";
    for (int i = 0; i < result.size(); i++) {
        cout << result[i];
        if (i < result.size() - 1) {
            cout << ", ";
        }
    }
    cout << endl;

    return 0;
}

vector<int> createMatrixAndGetSpiralOrder(vector<vector<int> > &matrix) {
    vector<int> result;

    // Extract the elements in the desired spiral order
    int left = 0, right = matrix[0].size() - 1, top = 0, bottom =
matrix.size() - 1;
```



```
    int direction =0;
    while (top <= bottom && left <= right) {
        // Traverse from top to bottom
        if(direction == 0){
            // Traverse from left to right
        for (int i = left; i <= right; i++) {
            result.push_back(matrix[top][i]);
        }
        top++;
        direction = 1;
        }
        else if (direction == 1){
            // Traverse from bottom to top
        for (int i = top; i <= bottom; i++) {
            result.push_back(matrix[i][right]);
        }
        right--;
        direction = 2;
        }
        else if(direction == 2){
        // Traverse from right to left
            for (int i = right; i >= left; i--) {
                result.push_back(matrix[bottom][i]);
            }
            bottom--;
            direction =3;
        }
        else if (direction == 3){
        for (int i = top; i <= bottom; i++) {
            result.push_back(matrix[i][left]);
        }
        left++;
        direction =0;
        }
    }

    return result;
}
```

Task 3 output

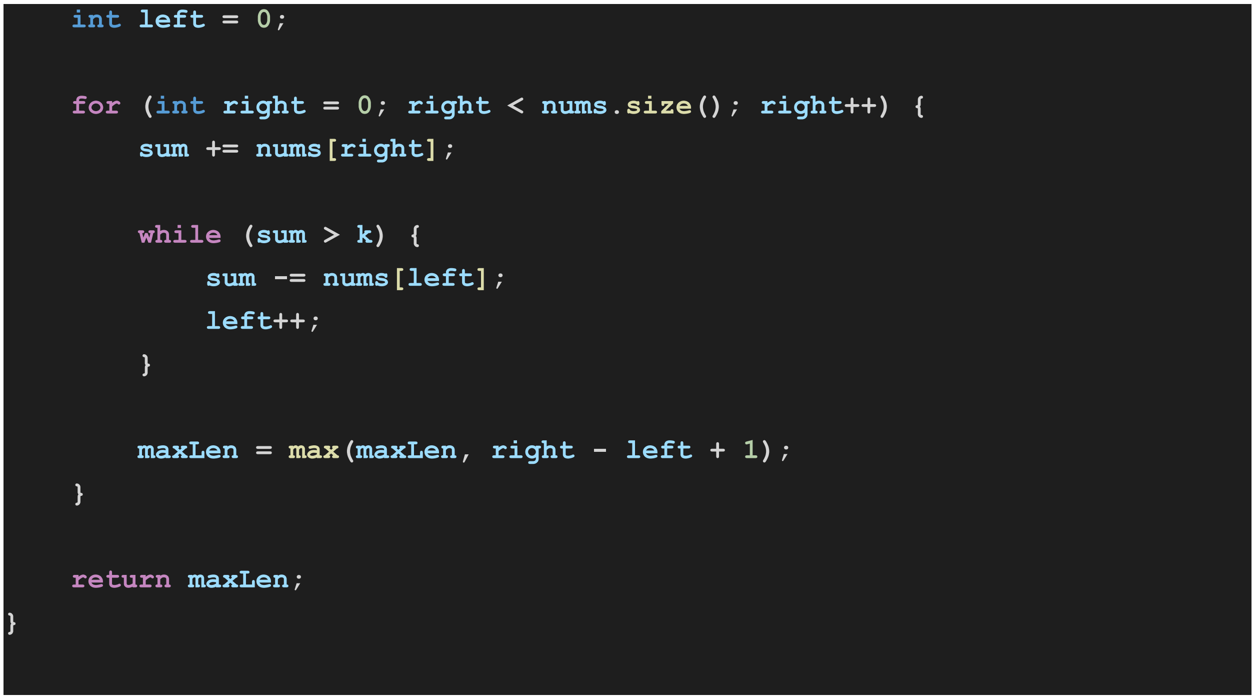


```
Enter the number of rows: 3
Enter the number of columns: 3

Spiral Order: 1, 2, 3, 6, 9, 8, 7, 4, 5
```

Task 4

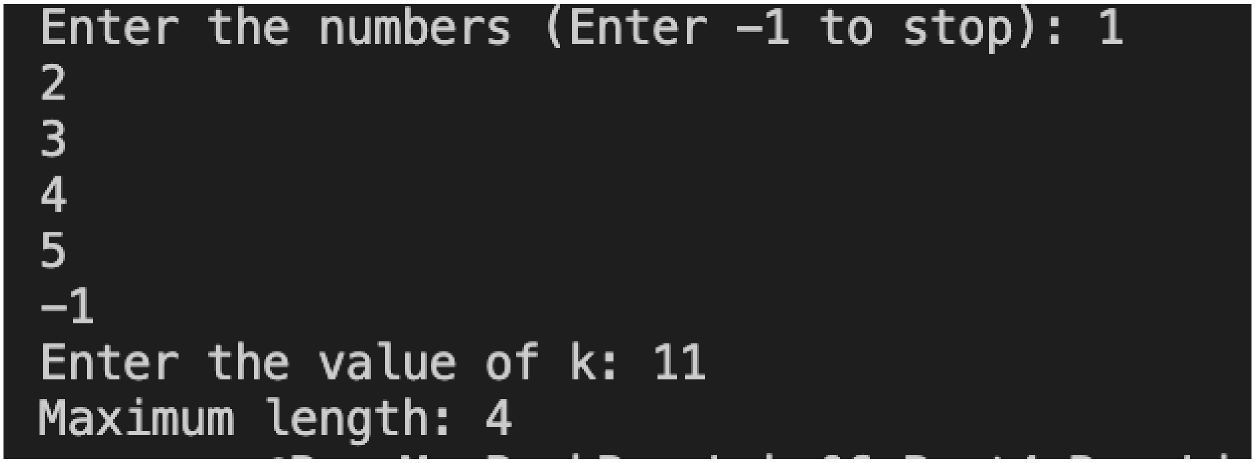
task4.cpp

```
int main() {
    int k;
    vector<int> nums;
    int num;

    cout << "Enter the numbers (Enter -1 to stop): ";

    while (true) {
        cin >> num;
        if (num == -1) {
            break;
        }
        if (num > NUM_MAX) {
            cout << "The number is too big. Please enter a smaller
number." << endl;
        } else {
            nums.push_back(num);
        }
    }

    cout << "Enter the value of k: ";
    cin >> k;

    if (k > NUM_MAX) {
        cout << "The value of k is too big." << endl;
        return 1; // You might want to return a non-zero value to indicate
an error.
    }

    int result = maxSubArrayLength(nums, k);
    cout << "Maximum length: " << result << endl;

    return 0;
}

int maxSubArrayLength(vector<int>& nums, int k) {
    int maxLen = 0;
    int sum = 0;
```



```
    int left = 0;

    for (int right = 0; right < nums.size(); right++) {
        sum += nums[right];

        while (sum > k) {
            sum -= nums[left];
            left++;
        }

        maxLen = max(maxLen, right - left + 1);
    }

    return maxLen;
}
```

Task 4 output



```
Enter the numbers (Enter -1 to stop): 1
2
3
4
5
-1
Enter the value of k: 11
Maximum length: 4
```

Task 5

task5.cpp

```
int main() {
    int rows, cols;
    cout << "Enter the number of rows: ";
    cin >> rows;
    cout << "Enter the number of columns: ";
    cin >> cols;

    vector<vector<int>> nums(rows, vector<int>(cols));

    cout << "Input the values of the matrix:" << endl;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cin >> nums[i][j];
        }
    }
     vector<int> result = rowProduct(nums);

    for (int i = 0; i < result.size(); i++) {
        cout << result[i] << " ";
    }
    cout << endl;


    int maxSubarrayProduct = maxProduct(result);

    cout << "Maximum product of a subarray: " << maxSubarrayProduct << endl;
    return 0;
}
```

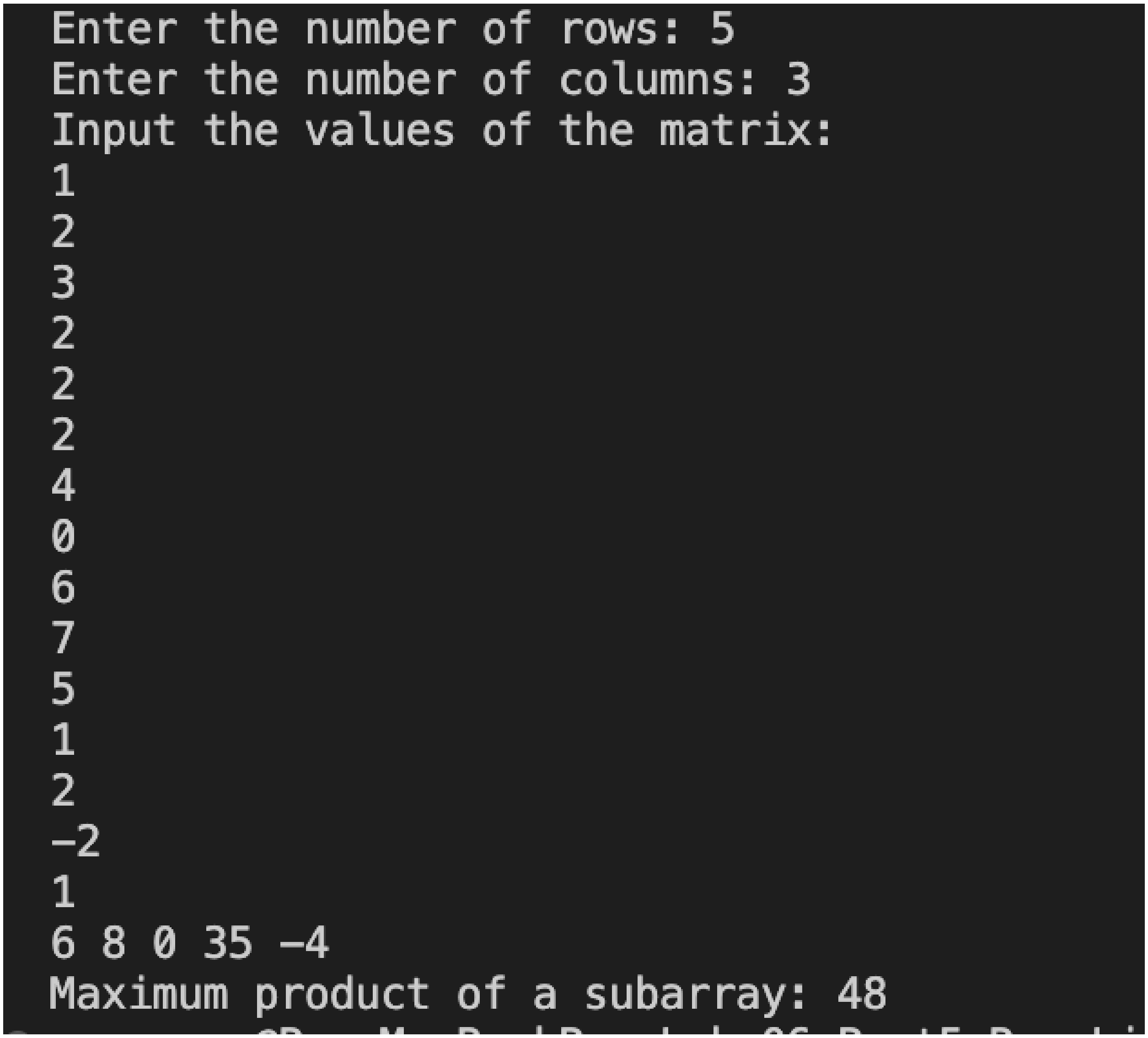
matrix_product.cpp

```
vector<int> rowProduct(vector<vector<int>>& nums) {
    vector<int> productRow;
    int rows = nums.size();
    int cols = nums[0].size();

    for (int i = 0; i < rows; i++) {
        int product = 1;
        for (int j = 0; j < cols; j++) {
            product *= nums[i][j];
        }
        productRow.push_back(product);
    }

    return productRow;
}
int maxProduct(vector<int>& nums) {
    int maxProduct = nums[0];
    int minProduct = nums[0];
    int result = nums[0];

    for (int i = 1; i < nums.size(); i++) {
        if (nums[i] < 0) {
            swap(maxProduct, minProduct);
        }

        maxProduct = max(nums[i], maxProduct * nums[i]);
        minProduct = min(nums[i], minProduct * nums[i]);

        result = max(result, maxProduct);
    }

    return result;
}
```

Matrix_product.h

```
vector<int> rowProduct(vector<vector<int>>& nums);
int maxProduct(vector<int>& nums);
```

Task 5 output:



```
Enter the number of rows: 5
Enter the number of columns: 3
Input the values of the matrix:
1
2
3
2
2
2
4
0
6
7
5
1
2
-2
1
6 8 0 35 -4
Maximum product of a subarray: 48
```