گزارش کار

HW1

دانش عبداللهی

9723053

در فایل **hw1.h** تمام کتاب‌خانه‌های مورد نیاز را اضافه می‌کنیم. (البته این اضافه کردن کتاب‌خانه‌ها به مرور با تکمیل شدن کد صورت گرفته است.)

همچنین نوع متغیر **Matrix** را با استفاده از **vector** ها مشخص می‌کنیم.

```cpp
#include <cmath>
#include <iomanip>
#include <iostream>
#include <random>
#include <stdexcept>
#include <string>
#include <vector>

using Matrix = std::vector<std::vector<double>>;
```

همچنین در این فایل **declarations** های تمام توابع را در **namespace algebra** وارد می‌کنیم.

```cpp
namespace algebra {
Matrix zeros(size_t n, size_t m);
Matrix ones(size_t n, size_t m);
Matrix random(size_t n, size_t m, double min, double max);
void show(const Matrix& matrix);
Matrix multiply(const Matrix& matrix, double c);
Matrix multiply(const Matrix& matrix1, const Matrix& matrix2);
Matrix sum(const Matrix& matrix, double c);
Matrix sum(const Matrix& matrix1, const Matrix& matrix2);
Matrix transpose(const Matrix& matrix);
Matrix minor(const Matrix& matrix, size_t n, size_t m);
double determinant(const Matrix& matrix);
Matrix inverse(const Matrix& matrix);
Matrix concatenate(const Matrix& matrix1, const Matrix& matrix2, int axis = 0);
Matrix ero_swap(const Matrix& matrix, size_t r1, size_t r2);
Matrix ero_multiply(const Matrix& matrix, size_t r, double c);
Matrix ero_sum(const Matrix& matrix, size_t r1, double c, size_t r2);
Matrix upper_triangular(const Matrix& matrix);
}
```

در ادامه تعاریف تمام توابع را در فایل **hw1.cpp** می‌نویسیم.

1. تابع zeros

برای ساختن یک ماتریس **n*m** به طوری که تمام درایه‌های آن صفر باشند، **n** تا بردار **m**تائی صفر می‌سازیم و آنرا در متغیر به نوع **Matrix** (که آن در **hw1.h** تعریف کرده بودیم) می‌ریزیم.

ورودی این تابع، دو عدد صحیح نامنفی و خروجی آن یک ماتریس **n*m** با درایه‌های تمام صفر است.

اگر هر کدام از ابعاد ماتریس ورودی صفر باشد ، با **logic error** مواجه می‌شویم.

```
Matrix algebra::zeros(size_t n, size_t m)
{
    // If Size of The Matrix is Zero.
    if (n == 0 || m == 0)
        return Matrix {};
    // Definig The Output Matrix
    Matrix output;
    for (size_t i {}; i < n; i++) {
        std::vector<double> temp(m);
        output.push_back(temp);
    }
    return output;
}
```

2. تابع ones

برای ساختن یک ماتریس **n*m** به طوری که تمام درایه‌های آن یک باشند، **n** تا بردار **m**تائی با درایه‌های تمام یک می‌سازیم و آنرا در متغیر به نوع **Matrix** می‌ریزیم.

ورودی این تابع، دو عدد صحیح نامنفی و خروجی آن یک ماتریس **n*m** با درایه‌های تمام صفر است.

اگر هر کدام از ابعاد ماتریس ورودی صفر باشد ، با **logic error** مواجه می‌شویم.

```cpp
Matrix algebra::ones(size_t n, size_t m)
{
    // If Size of The Matrix is Zero.
    if (n == 0 || m == 0)
        return Matrix {};
    // Definig The Output Matrix
    Matrix output;
    for (size_t i {}; i < n; i++) {
        std::vector<double> temp(m, 1);
        output.push_back(temp);
    }
    return output;
}
```

3. تابع random

برای ساختن یک ماتریس n*m به طوری که تمام درایه‌های آن یک عدد رندوم بین min و max (ورودی‌های تابع) باشند، n تا بردار mتائی با درایه‌های تمام یک می‌سازیم و آنرا در متغیر به نوع Matrix می‌ریزیم.

برای ساختن اعداد رندوم در بازهٔ [min : max] ، از توابع کتاب‌خانهٔ random استفاده می‌کنیم.

ابتدا یک عدد به عنوان seed ، random engine می‌سازیم و سپس با استفاده از توزیع احتمال یکنواخت در بازهٔ معلوم ، یک عدد رندوم double در این بازه می‌سازیم.

در این تابع، اگر ورودی min از ورودی max بیشتر باشد یا اگر هرکدام از ابعاد ماتریس ورودی صفر باشد، با logic error مواجه می‌شویم.

```cpp
Matrix algebra::random(size_t n, size_t m, double min, double max)
{
    // If Size of The Matrix is Zero.
    if (n == 0 || m == 0)
        return Matrix {};
    // If The Input Min is bigger than Min
    if (min > max)
        throw std::logic_error("min cannot be greater than max");
    // Definig The Output Matrix
    Matrix output;
    for (size_t i {}; i < n; i++) {
        std::vector<double> temp;
        for (size_t j {}; j < m; j++) {
            std::random_device rd;
            std::default_random_engine eng(rd());
            std::uniform_real_distribution<double> distr(min, max); // Range is Min to Max
            temp.push_back(distr(eng));
        }
        output.push_back(temp);
    }
    return output;
}
```

4. تابع show

در این تابع ، ماتریس ورودی به تابع را با دو حلقهٔ **for** تو در تو چاپ می‌کنیم. (این تابع بدون خروجی است.)

```cpp
void algebra::show(const Matrix& matrix)
{
    // If The Input Matrix is Empty.
    if (matrix.empty())
        throw std::logic_error("Empty Matrix Can't be Shown");

    // Shohwing The Output Matrix
    std::cout << std::endl;
    for (size_t i {}; i < matrix.size(); i++) {
        std::cout << std::fixed << "|";
        for (size_t j {}; j < matrix[0].size(); j++) {
            std::cout << std::setw(7) << std::fixed << std::setprecision(3)
                      << std::showpos << matrix[i][j] << " ";
        }
        std::cout << "|" << std::endl;
    }
}
```

نمونهٔ خروجی این تابع به صورت زیر است :

```
| +5.206  +1.466  +6.907  +2.436 |
| -1.104  +5.847  +1.491  -2.334 |
| -0.146  +1.210  -1.680  +5.882 |
| -4.103  +5.750  -2.031  +6.211 |
```

در این تابع، اگر ماتریس ورودی تهی باشد، با **logic error** مواجه می‌شویم.

5. تابع multiply

این تابع ماتریس ورودی را در عدد ورودی ضرب می‌کند. یعنی تمام درایه‌های ماتریس ورودی را در عدد معلوم ضرب می‌کند. برای اینکار در ابتدای تابع، ماتریس ورودی را در یک متغیر **temp** از نوع **Matrix** تعریف می‌کنیم. زیرا ماتریس ورودی فقط قابل خواندن است.

```
Matrix algebra::multiply(const Matrix& matrix, double c)
{
    // If The Input Matrix is Empty.
    if (matrix.empty())
        return Matrix {};
    // Definig The Output Matrix
    Matrix temp { matrix };
    for (size_t i {}; i < matrix.size(); i++)
        for (size_t j {}; j < matrix[0].size(); j++)
            temp[i][j] *= c;

    return temp;
}
```

در این تابع، اگر ماتریس ورودی تهی باشد، با **logic error** مواجه می‌شویم.

6. تابع multiply

این تابع، دو ماتریس ورودی را در هم ضرب ماتریسی می‌کند و به عنوان خروجی تحویل می‌دهد.

```
Matrix algebra::multiply(const Matrix& matrix1, const Matrix& matrix2)
{
    // If The Input Matrices is Empty.
    if (matrix1.empty() && matrix2.empty()) {
        return Matrix {};
        // If One Of The Input Matrices is Empty.
    } else if (matrix1.empty() || matrix2.empty())
        throw std::logic_error("matrices with wrong dimensions cannot be multiplied");

    // If Matrices Dimesnsions doesn't Match !
    if (matrix1[0].size() != matrix2.size())
        throw std::logic_error("matrices with wrong dimensions cannot be multiplied");

    else {
        // Definig The Output Matrix
        Matrix Output { algebra::zeros(matrix1.size(), matrix2[0].size()) };
        for (size_t i {}; i < matrix1.size(); i++)
            for (size_t j {}; j < matrix2[0].size(); j++)
                for (size_t k {}; k < matrix2.size(); k++)
                    Output[i][j] += matrix1[i][k] * matrix2[k][j];
        return Output;
    }
}
```

در این تابع، اگرهر دو ماتریس ورودی تهی باشند، خروجی تابع ماتریس تهی است اما اگر فقط یکی از ماتریس‌های ورودی تهی باشد، یا ابعاد ماتریس‌های ورودی برای ضرب ماتریسی با هم ، هم‌خوانی نداشته باشند ، با **logic error** مواجه می‌شویم.

7. تابع sum

این تابع ماتریس ورودی را با عدد ورودی جمع می‌کند. یعنی تمام درایه‌های ماتریس ورودی را با عدد معلوم جمع می‌کند. برای اینکار در ابتدای تابع، ماتریس ورودی را در یک متغیر **temp** از نوع **Matrix** تعریف می‌کنیم. زیرا ماتریس ورودی فقط قابل خواندن است.

```cpp
Matrix algebra::sum(const Matrix& matrix, double c)
{
    // If The Input Matrix is Empty.
    if (matrix.empty())
        return Matrix {};

    // Definig The Output Matrix
    Matrix temp { matrix };
    for (size_t i {}; i < matrix.size(); i++)
        for (size_t j {}; j < matrix[0].size(); j++)
            temp[i][j] += c;

    return temp;
}
```

در این تابع، اگر ماتریس ورودی تهی باشد، خروجی تابع ماتریس تهی است .

8. ماتریس sum

این تابع دو ماتریس ورودی را با هم جمع ماتریسی می‌کند. یعنی درایه‌های نظیر به نظیر هر دو ماتریس را با هم جمع می‌کند.

```cpp
Matrix algebra::sum(const Matrix& matrix1, const Matrix& matrix2)
{
    // If The Input Matrices is Empty.
    if (matrix1.empty() && matrix2.empty()) {
        return Matrix {};
        // If One Of The Input Matrices is Empty.
    } else if (matrix1.empty() || matrix2.empty())
        throw std::logic_error("matrices with wrong dimensions cannot be Summed");
    // Definig The Output Matrix
    Matrix Summ { algebra::zeros(matrix1.size(), matrix1[0].size()) };
    if (matrix1.size() == matrix2.size()
        && matrix1[0].size() == matrix2[0].size()) {
        for (size_t i {}; i < matrix1.size(); i++)
            for (size_t j {}; j < matrix1[0].size(); j++)
                Summ[i][j] += matrix1[i][j] + matrix2[i][j];
    } // If Matrices Dimesnsions doesn't Match !
    else
        throw std::logic_error("matrices with wrong dimensions cannot be Summed");

    return Summ;
}
```

در این تابع، اگرهر دو ماتریس ورودی تهی باشند، خروجی تابع ماتریس تهی است اما اگر فقط یکی از ماتریس‌های ورودی تهی باشد، یا ابعاد ماتریس‌های ورودی برای جمع ماتریسی با هم ، هم‌خوانی نداشته باشند ، با **logic error** مواجه می‌شویم.

9. تابع transpose

خروجی این تابع، **transpose** شدهٔ ماتریس ورودی است. اگر ماتریس ورودی **n*m** باشد ، ماتریس خروجی **m*n** خواهد بود.

```
Matrix algebra::transpose(const Matrix& matrix)
{
    // If The Input Matrix is Empty.
    if (matrix.empty())
        return Matrix {};
    // Definig The Output Matrix
    Matrix Trans { algebra::zeros(matrix[0].size(), matrix.size()) };
    for (size_t i {}; i < matrix.size(); i++)
        for (size_t j {}; j < matrix[0].size(); j++)
            Trans[j][i] = matrix[i][j];
    return Trans;
}
```

در این تابع ، اگر ماتریس ورودی تهی باشد، ماتریس خروجی تهی خواهد بود.

10. تابع minor

در این تابع، ابتدا ماتریس ورودی را در یک متغیر **temp** از نوع **Matrix** میریزیم.

سپس با استفاده از تابع **erase** از کتابخانهٔ **vector** ، سطر **n**ام ورودی را پاک می‌کنیم. سپس **temp** را **transpose** می‌کنیم و سطر **m**ام را پاک می‌کنیم و در مرحلهٔ آخر دوباره ماتریس **temp** ، را **transpose** می‌کنیم و به عنوان خروجی تحویل می‌دهیم.

در این تابع، اگر ماتریس ورودی تهی باشد، خروجی تابع ماتریس تهی است .

```cpp
Matrix algebra::minor(const Matrix& matrix, size_t n, size_t m)
{
    // If The Input Matrix is Empty.
    if (matrix.empty())
        return Matrix {};
    // Using erase Func. To Generate The Minor Matrix From Input Matrix.
    // Definig The Output Matrix
    Matrix Temp { matrix };
    Temp.erase(Temp.begin() + n);
    Temp = algebra::transpose(Temp);
    Temp.erase(Temp.begin() + m);
    return algebra::transpose(Temp);
}
```

11. تابع determinant

این تابع دترمینان ماتریس ورودی را به عنوان خروجی تحویل می‌دهد. در تعریف این تابع، خود تابع را فرا می‌خوانیم. (تابع بازگشتی است.) برای اینکه تابع بازگشتی بنویسیم ، یک شرط پایانی برای تعریف تابع می‌نویسیم. (دترمینان ماتریس 1*1 برابر با خودش است.)

```cpp
double algebra::determinant(const Matrix& matrix)
{
    // If The Input Matrix is Empty.
    if (matrix.empty())
        return 1;
    // If The Input Matrix isn't Square.
    if (matrix.size() != matrix[0].size())
        throw std::logic_error("non-square matrices have no determinant");
    // Defining The Initial Condition Cause We Using This Func. Recursively
    if (matrix.size() == 1 && matrix[0].size() == 1)
        return matrix[0][0];

    Matrix Temp { matrix };
    // Definig The Output Number.
    double Det {};
    for (size_t i {}; i < Temp[0].size(); i++)
        Det += std::pow(-1, 1 + i + 1) * Temp[0][i]
            * algebra::determinant(algebra::minor(Temp, 0, i));

    return Det;
}
```

در این تابع اگر ماتریس ورودی تهی باشد، خروجی برابر با عدد **1** است. همچنین اگر ماتریس ورودی مربعی نباشد، با **logic error** مواجه می‌شویم.

12. تابع inverse

خروجی این تابع ، **inverse** ماتریس ورودی است. در این تابع، از تابع‌های قبلی (**determinant** و **minor**) استفاده می‌کنیم.

```
Matrix algebra::inverse(const Matrix& matrix)
{
    // If The Input Matrix is Empty.
    if (matrix.empty())
        return Matrix {};
    // If The Input Matrix isn't Square.
    if (matrix.size() != matrix[0].size())
        throw std::logic_error("non-square matrices have no inverse");
    // If The Matrix Determinant is 0 .
    if (algebra::determinant(matrix) == 0)
        throw std::logic_error("singular matrices have no inverse");
    // Generating The Inverse Matrix
    Matrix output { algebra::zeros(matrix.size(), matrix[0].size()) };
    for (size_t i {}; i < matrix.size(); i++)
        for (size_t j {}; j < matrix[0].size(); j++)
            output[i][j] = std::pow(-1, i + j + 2)
                * algebra::determinant(algebra::minor(matrix, i, j));

    output = algebra::transpose(output);
    output = algebra::multiply(output, (1 / algebra::determinant(matrix)));
    return output;
}
```

در این تابع، اگر ماتریس ورودی مربعی نباشد یا دترمینان ماتریس ورودی **0** باشد ، با **logic error** مواجه می‌شویم. همچنین اگر ماتریس ورودی تهی باشد ، خروجی ماتریس تهی خواهد بود.

13. تابع concatenate

این تابع ، دو ماتریس ورودی را از کنار یا از زیر، به هم دیگر می‌چسباند.اگر ورودی **axis** تابع ، **0** باشد ، ماتریس**2** را از راست به ماتریس**1** می‌چسباند و اگر ورودی **axis** تابع ، **1** باشد ، ماتریس**2** را از پایین به ماتریس**1** می‌چسباند.

```
Matrix algebra::concatenate(const Matrix& matrix1, const Matrix& matrix2, int axis)
{
    // If The Input Matrices is Empty.
    if (matrix1.empty() && matrix2.empty()) {
        return Matrix {};
        // If One Of The Input Matrices is Empty.
    } else if (matrix1.empty() || matrix2.empty())
        throw std::logic_error("matrices with wrong dimensions cannot be Summed");

    // If Matrices Dimesnsions doesn't Match !
    if (axis == 0 && matrix1[0].size() != matrix2[0].size())
        throw std::logic_error("matrices with wrong dimensions cannot be concatenated");
    if (axis == 1 && matrix1.size() != matrix2.size())
        throw std::logic_error("matrices with wrong dimensions cannot be concatenated");

    Matrix Temp { matrix1 };
    // If axis = 0
    if (axis == 0) {
        // Definig The Output Matrix
        for (size_t i {}; i < matrix2.size(); i++)
            Temp.push_back(matrix2[i]);
        return Temp;
    }
    // If axis = 1
    else {
        // Definig The Output Matrix
        Temp = algebra::transpose(Temp);
        for (size_t i {}; i < matrix2[0].size(); i++)
            Temp.push_back(algebra::transpose(matrix2)[i]);

        return algebra::transpose(Temp);
    }
}
```

در این تابع، اگرهر دو ماتریس ورودی تهی باشند، خروجی تابع ماتریس تهی است اما اگر فقط یکی از ماتریس‌های ورودی تهی باشد، یا ابعاد ماتریس‌های ورودی برای به هم چسباندن در راستای افقی یا عمودی ، هم‌خوانی نداشته باشند ، با **logic error** مواجه می‌شویم.

14. تابع ero_swap

در این تابع با استفاده از تابع **swap** از کتاب‌خانهٔ **vector** ، دو سطر **r1**ام و سطر **r2**ام را با هم عوض می‌کند.

```
Matrix algebra::ero_swap(const Matrix& matrix, size_t r1, size_t r2)
{
    // If The Input Matrix is Empty.
    if (matrix.empty())
        return Matrix {};

    // If r1 or r2 are Out Of Range.
    if (r1 >= matrix.size() || r2 >= matrix.size())
        throw std::logic_error("r1 or r2 inputs are out of range");
    // Definig The Output Matrix
    Matrix output { matrix };
    output[r1].swap(output[r2]);
    return output;
}
```

در این تابع اگر ماتریس ورودی تهی باشد، خروجی تابع ماتریس تهی است. همچنین اگر ورودی **r1** و **r2** از سایز ابعاد ماتریس (تعداد سطرهای ماتریس) بیشتر باشد، با **logic error** مواجه می‌شویم.

15. تابع ero_multiply

در این تابع ، فقط سطر **r**ام ماتریس ورودی را در عدد **c** ضرب می‌کنیم.

```
Matrix algebra::ero_multiply(const Matrix& matrix, size_t r, double c)
{
    // If The Input Matrix is Empty.
    if (matrix.empty())
        return Matrix {};

    // If r is Out Of Range.
    if (r >= matrix.size())
        throw std::logic_error("r is out of range");
    // Definig The Output Matrix
    Matrix output { matrix };
    output[r].swap(algebra::multiply(matrix, c)[r]);
    return output;
}
```

ابتدا ماتریس ورودی را در یک متغیر **temp** از نوع **Matrix** می‌ریزیم و ماتریس ورودی را در عدد **c** ضرب می‌کنیم. سپس با استفاده از تابع **swap**، سطر **r**ام ماتریس ضرب شده را در سطر **r**ام متغیر **temp** می‌ریزیم.

در این تابع اگر ماتریس ورودی تهی باشد، خروجی تابع ماتریس تهی است. همچنین اگر ورودی **r** از سایز ابعاد ماتریس (تعداد سطرهای ماتریس) بیشتر باشد، با **logic error** مواجه می‌شویم.

16. تابع ero_sum

این تابع، سطر **r1**ام ماتریس ورودی را در عدد **c** ضرب می‌کند و آنرا با سطر **r2**ام ماتریس جمع می‌کند.ابتدا با استفاده از تابع **ero_multiply** سطر **r1**ام ماتریس را در عدد **c** ضرب می‌کنیم. سپس سطر **r1**ام آنرا با سطر **r2**ام جابه‌جا می‌کنیم و بعد ماتریس بدست آمده را با ماتریس ورودی تابع جمع می‌کنیم. در نهایت سطر **r2**ام ماتریس بدست آمده را در سطر **r2**ام ماتریس ورودی جایگزین می‌کنیم.

```
Matrix algebra::ero_sum(const Matrix& matrix, size_t r1, double c, size_t r2)
{
    // If The Input Matrix is Empty.
    if (matrix.empty())
        return Matrix {};
    // If r1 or r2 are Out Of Range.
    if (r1 >= matrix.size() || r2 >= matrix.size())
        throw std::logic_error("r1 or r2 inputs are out of range");
    // Definig The Output Matrix
    Matrix output { matrix };
    Matrix temp { algebra::ero_multiply(matrix, r1, c) };
    temp[r2].swap(temp[r1]);
    temp = algebra::sum(temp, matrix);
    output[r2].swap(temp[r2]);
    return output;
}
```

در این تابع اگر ماتریس ورودی تهی باشد، خروجی تابع ماتریس تهی است. همچنین اگر ورودی **r1** و **r2** از سایز ابعاد ماتریس (تعداد سطرهای ماتریس) بیشتر باشد، با **logic error** مواجه می‌شویم.

17. تابع upper_triangular

در این تابع ابتدا با جابه‌جا کردن سطرهای ماتریس ورودی، ماتریس ورودی را به ماتریس تبدیل می-کنیم که هیچ کدام از درایه‌های قطر اصلی آن صفر نباشد. (اگر نیاز باشد!)

سپس با اعمال عملیات‌های سطری مقدماتی ، ماتریس بالا مثلثی را می‌سازیم.

```
Matrix algebra::upper_triangular(const Matrix& matrix)
{
    // If The Input Matrix is Empty.
    if (matrix.empty())
        return Matrix {};
    // If The Input Matrix isn't Square.
    if (matrix.size() != matrix[0].size())
        throw std::logic_error("non-square matrices have no upper triangular form");

    Matrix output { matrix };
    std::vector<size_t> r {};
    // Sortting Matrix's Rows To Avoid Having Zero On Main Diagonal
    for (size_t j {}; j < matrix[0].size(); j++)
        for (size_t i { j }; i < matrix.size(); i++)
            if (matrix[i][j] != 0) {
                output[j].swap(output[i]);
                break;
            }
    // Generating The Upper_Trianglura Matrix
    for (size_t k {}; k < matrix.size(); k++)
        for (size_t z { k + 1 }; z < matrix.size(); z++)
            output = algebra::ero_sum(output, k, -output[z][k] / output[k][k], z);

    return output;
}
```

در این تابع اگر ماتریس ورودی تهی باشد، خروجی ماتریس تهی است. همچنین اگر ماتریس ورودی مربعی نباشد، با **logic error** مواجه می‌شویم.

در آخر هم برنامه را اجرا می‌کنیم و خروجی به صورت زیر است :


[----------] Global test environment tear-down
[==========] 24 tests from 1 test suite ran. (7 ms total)
[PASSED] 24 tests.
<<<SUCCESS>>>

- لینک git