به نام خدا

محمد جواد امین – ۹۵۲۳۰۰۸

گزارش کار سری دوم AP

این گزار ش کار در هر مرحله بر اساس صورت مسئله است.

هدف این پروژه مانند پروژه قبل ؛بدست آوردن وزن های مناسب برای تخمین نمره ها ؛ بر اساس ویژگی‌های داده شده هر دانشجو است . روش حل این پروژه با پروژه قبلی متفاوت است.

در ابتدا نیاز به تعریف یک data type داریم پس Matrix class را تعریف میکنیم.

این class شامل member variable و همچنین member function هایی است که در ادامه به تعریف و بررسی تک تک آن ها می پردازیم.

ابتدا member variable` ها به صورت private تعریف می‌شود.

member variables : data که شامل المان های ماتریس است یه صورت یک وکتور دو بعدی تعریف شده و size متغییر بعدی است که تعداد سطر و ستون را نشان می دهد و جنس آن array به طول ۲ است.

در مرحله بعد constructor های class Matrix را تعریف و قابل مشاهده است.

```
class Matrix
{
private :                                       //member variables
    std::vector<std::vector<double>> Data{};    // elements of matrix
    std::array<size_t, 2> size{};               //number of rows and columns
public:                                         //member functions
    Matrix(std::vector<std::vector<double>> data);      // constructors
    Matrix(size_t , size_t, bool ones = true);
    Matrix(const Matrix&);                  // copy constructor
    Matrix(std::unique_ptr<std::unique_ptr<double[]>[]>& data, size_t, size_t);
    Matrix();                                       // defult construactor

    std::array<size_t, 2> getSize();     //size of the matrix
    double det();            //determinant of the matrix
    Matrix inv();           // inverse of the matrix
    Matrix T();             //transpose of the matrix
    void show();             //displays the matrix
    Matrix delCol(size_t );   //removes the given (i-th) column of the matrix
    Matrix col(size_t );    //returns the given (i-th) column of the matrix
    void save(const char*);    //saves the matrix as a csv file
    void load(const char*);   //This function loads a csv file into a matrix.

    Matrix operator+(const Matrix&);    // add matrix
    Matrix operator-(const Matrix&);       //subtraction matrixs
    Matrix operator*(const Matrix&);     //multiply matrix
    std::vector <double>& operator[](int);
};
```

اولین Constructor ؛ ورودی آن یک وکتور دو بعدی است که به صورت زیر تعریف شده.

```
Matrix::Matrix(std::vector<std::vector<double>> data)
{
    Data = data;   // copy each elemnt of data to Matrix
    size[1] = Data[0].size();     //size of matrix
    size[0] = Data.size();
}
```

دومین constructor: ورودی آن تعداد سطر و ستون است و با یک متغییر از جنس bool . این constructor ؛ ماتریسی با المان های صفر یا یک با ابعاد داده شده می‌سازد که تعریف آن به شرح زیر است.

```
Matrix::Matrix(size_t m, size_t n, bool ones)
{
    size[0] = m;   //size of matrix
    size[1] = n;
    for (size_t row = 0; row < size[0]; row++)
    {
        std::vector<double> vector_row{};   //for store row of data

        for (size_t column = 0; column < size[1]; column++)
        {
            if (ones)
                vector_row.push_back(1);   //put one in vector
            else
                vector_row.push_back(0);   //put zero in vector
        }
        Data.push_back(vector_row);    //save row of data
    }
}
```

سومین constructor ؛ورودی آن یک ماتریس است و بصورت reference به آن داده میشود و این باعث عملکرد بهتر و صرفه جویی در حافظه شده . کار این constructor کپی کردن ماتریس و اصطلاحا به آن copy constructor نیزگفته می شود.

```
Matrix::Matrix(const Matrix& mat)   // copy constructor
{
    size[0] = mat.size[0];     //copy size
    size[1] = mat.size[1];
    Data = mat.Data;          // copy matrix
}
```

چهارمین constructor ؛ ورودی آن به ترتیب unique_ptr دو بعدی ؛ تعداد سطر ها و نهایتا ستون های ماتریس است.

```
Matrix::Matrix(std::unique_ptr<std::unique_ptr<double[]>[]>& data, size_t m, size_t n)
{
    size[0] = m;   // size of matrix
    size[1] = n;
    for (size_t row = 0; row < size[0]; row++)   //row
    {
        std::vector<double> vector_row{};   //for store row of data

        for (size_t column = 0; column < size[1]; column++)    //column
            vector_row.push_back(data[row][column]);    //store data
        Data.push_back(vector_row);     //store a row of data
    }
}
```

جواب Qusetion1 : ما نمیتوانیم که unique_ptr را به صورت pass by value برگردانیم چون قابل کپی شدن نیست به همین دلیل constructor آن را حذف می‌کند ؛ پس آن را باید صورت reference به تابع دهیم.

آخرین constructor ؛ ورودی ندارد و به آن default constructor می‌گویند.

```
Matrix::Matrix()  //default constructor
{
    size[0] = 0;        // size of matrix
    size[1] = 0;
    Data = {};      //empty matrix
}
```

حال باید به سراغ member function ها می‌رویم :

اولین تابع؛ getsize است که کار آن برگرداندن تعداد سطر و ستون ماتریس به صورت یک array دو عضوی است.

```
std::array<size_t, 2> Matrix::getSize()    // to show size of matrix
{
    return std::array<size_t, 2>{size[0],size[1]};    //  return size of matrix
}
```

دومین تابع ؛ det است که این تابع باید دترمینان ماتریس را حساب نموده و خروجی را برگرداند و اگر ماتریس مربعی نبود صفر را برگرداند. برای حساب نمودن دترمینان ماتریس با توجه به نحوه حساب کردن آن ؛ الگوریتم محاسبه آن را با استفاده از تابع بازگشتی حساب میکنم که برای این منظور نیاز به تعریف یک تابع به صورت زیر است.

```
double Matrix::det()         //determinant of the matrix
{
    if (size[0] != size[1])              //check  matrix is the squre form or not
    {
        std::cout << "the matrix is not in the square form " << std::endl;
        return 0.0;
    }
    else
        return det_funtion(Data, size[0]);    //return determinant of the matrix
}
```

الگوریتم تابع بازگشتی محاسبه دترمینان را به روش زیر پیاده سازی می‌نماییم.

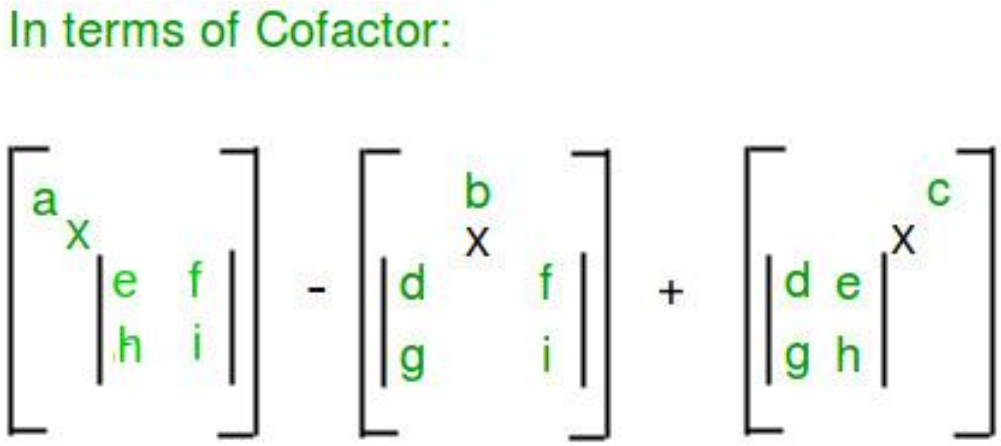
```cpp
double det_funtion(std::vector<std::vector<double>> v, size_t n)     //Recursive function for finding determinant of matrix
{
    double determinant{};

    if (n == 1)
    {
        std::cout << " inavalid matrix" << std::endl;
        return 0;
    }
    else if (n == 2)
        return (v[0][0] * v[1][1]) - (v[0][1] * v[1][0]);       //determinant matrix  2*2
    else
    {
        for (size_t i = 0; i < n; i++)                          // loop to get cofactor for each element of matrix
        {
            std::vector<std::vector<double>> det_vector{};     // to store elemnt of matrix of minors

            for (size_t a = 0; a < n; a++)        //to get matrix of minors   //row
            {
                std::vector<double> det_vector_row{};
                bool flag{ false };                    // for  showing  push back is done or not

                for (size_t b = 0; b < n; b++)    // column
                {
                    if (a != 0 && b != i)     //  Copying into temporary matrix only those element  which are not in given row and column
                    {
                        det_vector_row.push_back(v[a][b]);
                        flag = true;
                    }
                }
                if (flag)
                    det_vector.push_back(det_vector_row);   //To store matrix of minors
            }
            determinant += (std::pow(-1, i) * v[0][i] * det_funtion(det_vector, n - 1));    //calculalte cofactors and  determinant by algorithm recursive determinant
        }
        return determinant;
    }
}
```

الگوریتم ما به این صورت است برای ماتریس ۲در۲ دترمینان به صورت زیر حساب شده:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$|A| = \text{ad - bc}$$

برای ماتریس های با مرتبه بالاتر؛ دترمینان آن به صورت زیر محاسبه شده یعنی cofactor های ماتریس را حساب می‌کنیم.



سومین تابع ؛invاست که باید ماتریس معکوس را برگرداند و اگر ماتریس مربعی نبود باید ماتریسی با المان های صفر برگرداند. ابتدا الگوریتم محاسبه ماتریس معکوس بررسی میکنیم

روش بدست آوردن ماتریس معکوس را به چهار مرحله تقسیم می‌کنیم:

مرحله اول: بدست آوردن matrix of minors است یعنی برای هر المان از ماتریس عناصر آن ستون و سطرکه المان در آن قرار دارد را نادیده می‌گریم و دترمینان ماتریس باقی مانده را حساب کرده و ضرب در آن المان میکنیم. برای هر المان از ماتریس این کار را انجام می‌دهیم.برای مثال برای ماتریس زیر داریم:

$$A = \begin{bmatrix} 3 & 0 & 2 \\ 2 & 0 & -2 \\ 0 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \bullet & 0 & 2 \\ 2 & 0 & -2 \\ 0 & 1 & 1 \end{bmatrix} \quad 0\times1 - (-2)\times1 = 2$$

$$\begin{bmatrix} 3 & \bullet & 2 \\ 2 & 0 & -2 \\ 0 & 1 & 1 \end{bmatrix} \quad 2\times1 - (-2)\times0 = 2$$

$$\begin{bmatrix} 3 & 0 & 2 \\ 2 & 0 & -2 \\ 0 & \bullet & 1 \end{bmatrix} \quad 3\times-2 - 2\times2 = -10$$

$$\begin{bmatrix} 3 & 0 & 2 \\ 2 & 0 & -2 \\ 0 & 1 & \bullet \end{bmatrix} \quad 3\times0 - 0\times2 = 0$$

و در نتیجه داریم:

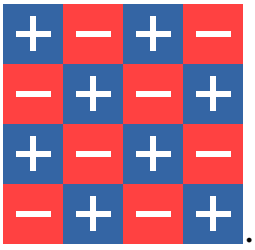
$$\begin{bmatrix} 0\times1 - (-2)\times1 & 2\times1 - (-2)\times0 & 2\times1 - 0\times0 \\ 0\times1 - 2\times1 & 3\times1 - 2\times0 & 3\times1 - 0\times0 \\ 0\times(-2) - 2\times0 & 3\times(-2) - 2\times2 & 3\times0 - 0\times2 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 2 \\ -2 & 3 & 3 \\ 0 & -10 & 0 \end{bmatrix}$$

Matrix of Minors

مرحله دوم بدست آوردن cofactors است که به صورت زیر قابل انجام است

$$\begin{bmatrix} 2 & 2 & 2 \\ -2 & 3 & 3 \\ 0 & -10 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} + & - & + \\ - & + & - \\ + & - & + \end{bmatrix} \rightarrow \begin{bmatrix} 2 & -2 & 2 \\ +2 & 3 & -3 \\ 0 & +10 & 0 \end{bmatrix}$$

Matrix of Minors *Matrix of CoFactors*



مرحله سوم بدست آوردن adjugate ماتریس است که برای آن باید تمام المان ها را transpose کرد

مرحله چهارم بدست آوردن دترمینان ماتریس اصلی است (در قسمت قبل بدست آوردن آن توضیح داده شده است) و سپس آن را در تک تک المان های ماتریس adjugate ضرب میکنیم.

$$A^{-1} = \frac{1}{10}\begin{bmatrix} 2 & 2 & 0 \\ -2 & 3 & 10 \\ 2 & -3 & 0 \end{bmatrix} = \begin{bmatrix} 0.2 & 0.2 & 0 \\ -0.2 & 0.3 & 1 \\ 0.2 & -0.3 & 0 \end{bmatrix}$$

Adjugate Inverse

حال به بررسی کد پیاده سازی شده این قسمت می پردازیم. توضیحات مربوط به پیاده‌سازی صورت کامنت آورده شده است.

```
Matrix Matrix::inv()                    // inverse of the matrix
{
    if (size[0] != size[1])          //check  matrix is the squre form or not
    {
        Matrix inverse{ size[0], size[1], false };
        std::cout << "the matrix is not in the square form " << std::endl;
        return  inverse;                    // if it's not square form return a martix with zero elemnt
    }

    else if (size[0] == 2)                // inverse of matrix 2*2
    {
        Matrix inverse{ 2,2,false };
        Matrix data_inverse{ Data };

        inverse[0][0] = data_inverse[1][1] / data_inverse.det();
        inverse[0][1] = -data_inverse[0][1] / data_inverse.det();
        inverse[1][0] = -data_inverse[1][0] / data_inverse.det();
        inverse[1][1] = data_inverse[0][0] / data_inverse.det();

        return inverse;
    }
```

```
else     // inverse of matrix n*n, n>2
{
    std::vector<std::vector<double>> inverse_vector{};  // to store of inverse matrix

    for (size_t i = 0; i < size[0]; i++)    //row of matrix
    {
        std::vector<double> inverse_vector_row{};          // for give each row to 2d vector inverse

        for (size_t j = 0; j < size[1]; j++)     //column of matrix
        {
            std::vector<std::vector<double>> temp_vector{};     // for compute det each element

            for (size_t k = 0; k < size[0]; k++)      // to get matrix of minors   //row
            {
                std::vector<double> temp_vector_row{};        // for give each row for compute det
                bool flag{ false };                // for  showing  push back is done or not

                for (size_t l = 0; l < size[1]; l++)   // to get matrix of minors   //column
                {
                    if (k != i && l != j)  //  Copying into temporary matrix only those element  which are not in given row and column
                    {
                        temp_vector_row.push_back(Data[l][k]);     // data must be transpose(adjugate matrix)
                        flag = true;
                    }
                }
                if (flag)
                    temp_vector.push_back(temp_vector_row);       //to store matrix of minors
            }
            Matrix temp_2{ temp_vector };

            inverse_vector_row.push_back(temp_2.det() * std::pow(-1, i + j) / Matrix{ Data }.det());  // claculate cofators and multiply by 1/determinat
        }
        inverse_vector.push_back(inverse_vector_row);          // to store inverse matrix
    }

    return Matrix { inverse_vector };   //return matrix inverse
}
}
```

چهارمین تابع ؛ T است که کار آن برگرداندن ماتریس transpose است . برای بدست آوردن آن کافی است که جای سطر و ستون را عوض کنیم. در قسمت قبل نمونه آن را دیدم.

```
Matrix Matrix::T()     // change row and column eith each other
{
    std::vector<std::vector<double>> transpose_vector{};    // to store transpose element

    for (size_t i = 0; i < size[1]; i++)      // column
    {
        std::vector<double> transpose_vector_row{};

        for (size_t j = 0; j < size[0]; j++)      // row
            transpose_vector_row.push_back(Data[j][i]);         //changing row and column of each element and store it
        transpose_vector.push_back(transpose_vector_row);
    }

    return Matrix{ transpose_vector };   // return taranspose matrix
}
```

پنجمین تابع ؛ show است که کار آن نمایش دادن ماتریس است که پیاده سازی آن به صورت زیر است.

```
void Matrix::show()    // showing matrix
{
    for (size_t i = 0; i < size[0]; i++)    //row
    {
        for (size_t j = 0; j < size[1]; j++)     //column
            std::cout << std::setw(20) << Data[i][j];    // print each element
        std::cout << std::endl;
    }
}
```

ششمین تابع ؛ decol است که کار آن حذف کردن ستون iام است که به عنوان ورودی به آن داده می‌شود و نکته قابل توجه اینکه این تابع باید ماتریس باقی مانده پس از حذف ستون مورد نظر ذخیره و همچنان آن را به عنوان ورودی برگرداند. پیاده سازی آن به صورت زیر است.

```
Matrix Matrix::delCol(size_t k)    //remove ith column
{
    std::vector<std::vector<double>> remain_vector{};  // to store remain elements

    for (size_t i = 0; i < size[0]; i++)    //row
    {
        std::vector<double> remain_vector_row{};

        for (size_t j = 0; j < size[1]; j++)   //column
            if (j != k)      // for removes element which has i-th column
                remain_vector_row.push_back(Data[i][j]);  // store  remaining element
        remain_vector.push_back(remain_vector_row);
    }
    Data = remain_vector;     // store ramain elements of matrix
    size[0] = Data.size();     // store size of remain matrix
    size[1] = Data[0].size();

    return Matrix(Data);      // return remaining matrix
}
```

هفتمین تابع ؛ col است که کار آن ذخیره ستون l ام ماتریس بوده و برگرداندن آن به صورت ماتریسی دیگر است. پیاده سازی آن به صورت زیر قابل انجام است.

```
Matrix Matrix::col(size_t k)   //return i-th column
{
    std::vector<std::vector<double>> column{};  // to store elements of i-th column

    for (size_t i = 0; i < size[0]; i++)      //row
    {
        std::vector<double> temp{};
        temp.push_back(Data[i][k]);   // store elements if i-th column
        column.push_back(temp);
    }

    return Matrix{ column };         // return column i-th as a new matrix
}
```

هشتمین تابع؛ save است که کار آن ذخیره کردن ماتریس در یک فایل csv. است که اسم فایل به صورت ورودی به آن داده می شود. پیاده‌سازی آن به صورت زیر است.

```
void Matrix::save(const char* name)    // save matrix as a .csv file
{
    std::ofstream out{ name };    // .csv file

    for (size_t i = 0; i < size[0]; i++)   // rows
    {
        for (size_t j = 0; j < size[1]; j++)  // columns
            out << Data[i][j]<<',';             // print each elements of matrix
        out << std::endl;
    }
    out.close();   // close file
}
```

نهمین تابع؛load است که کار آن گرفتن ماتریس از فایل csv. است. که ورودی تابع اسم فایل است. نوشتن الگوریتم تابع به این صورت است که چون تعداد سطر ها و ستون های آن مشخص نیست باید یک حلقه بگذاریم که شرط آن eof است ولی ممکن است که در آخر فایل چند سطر خالی موجود باشد و نباید ما آن ها را در نظر بگیریم که این حالت را با یک if چک میکنم بعد از آن یک خط را داخل یکstring ذخیره میکنم و سپس شروع به گرفتن اعداد داخل هر خط می‌کنیم و این کار را تا به پایان فایل به صورت خط به خط انجام میدهم و در آخر اعداد بدست آمده در داخل ماتریس و اندازه ماتریس را نیز ذخیره میکنم.

پیاده سازی آن به صورت زیر است.

```
void Matrix::load(const char* name)    // loading matrix from .csv file
{
    std::ifstream input_file{ name };    //  read .csv file

    while (!input_file.eof())           // while until end of file
    {
        std::string line{};               // for store each line
        std::string data{};               // for store each data
        std::vector<double> row{};      // for store each row

        std::getline(input_file, line);   // store each line
        if (static_cast<int>(line[0]) == 0)   // if line is empty break from loop
            break;
        for (size_t i = 0; i < line.length(); i++)    // to get data from a line
        {
            if (line[i] != ',')   // continue to get a data
                data += line[i];
            else
            {
                row.push_back(std::stod(data));       // store data
                data = "";
            }
        }
        row.push_back(std::stod(data));
        Data.push_back(row);              // store a row of data in a Dta vector
    }
    ;
    size[0] = Data.size();             // store matrix size
    size[1] = Data[0].size();

    input_file.close();    //close matrix
```

حال به سراغ تعریف چند operator می‌رویم .

اولین + operator است که برای جمع ماتریسی استفاده می‌شود و به صورت زیر استفاده می‌شود.

```
Matrix Matrix::operator+(const Matrix& A)       // for add matrix
{
    if (size[0] != A.size[0] || size[1] != A.size[1])    // two matrix must have same dimension
    {
        throw std::invalid_argument(" dimensions not correct");
        return Matrix{};
    }

    std::vector<std::vector<double>> sum_vector{};       // for store sum of elements of eacg matrix

    for (size_t i = 0; i < size[0]; i++)           // row
    {
        std::vector<double> sum_row{};

        for (size_t j = 0; j < size[1]; j++)   //column
            sum_row.push_back(Data[i][j] + A.Data[i][j]);    //store sum of elements
        sum_vector.push_back(sum_row);
    }

    return Matrix(sum_vector);    //return sum matrix
}
```

نکته قابل توجه این است که ورودی این operator به صورت reference است که برای بالا بردن performance و جلوگیری از هدر رفتن بیهوده حافظه است و برای جلوگیری از تغییر در متغییر اصلی

که به صورت reference به ورودی operatorداده شده است آن را به صورت const می دهیم و این کار را برای همه operator هایی که تعریف میکنم انجام می‌دهیم.

دومین – operator است که برای تفریق دو ماتریس از هم استفاده می‌شود. که به صورت زیر پیاده می‌شود.

```
Matrix Matrix::operator-(const Matrix& A)  // for subtraction matrixs
{
    if (size[0] != A.size[0] || size[1] != A.size[1])  // two matrix must have same dimension
    {
        throw std::invalid_argument(" dimensions not correct");
        return Matrix{};
    }

    std::vector<std::vector<double>> subtraction_vector{};    // for store subtraction of elements of each matrix

    for (size_t i = 0; i < size[0]; i++)   //row
    {
        std::vector<double> sum_row{}; // for store row of data

        for (size_t j = 0; j < size[1]; j++)    //column
            sum_row.push_back(Data[i][j] - A.Data[i][j]);  //store subtraction of elements
        subtraction_vector.push_back(sum_row);       //store row vector
    }

    return Matrix(subtraction_vector);   //return subtraction matrix
}
```

سومین:* operator است که برای ضرب کردن دو ماتریس باید آن را تعریف کنیم. الگوریتم ضرب ماترسی به صورت زیر است

$$c_{11}=a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{31}+a_{14}b_{41}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix}$$

$2 \times 4 \qquad 4 \times 3 \qquad 2 \times 3$

$$c_{22}=a_{21}b_{12}+a_{22}b_{22}+a_{23}b_{32}+a_{24}b_{42}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix}$$

حال به سراغ پیاده سازی آن می‌رویم.

```
Matrix Matrix::operator*(const Matrix& B)    //matrix multiplication
{
    if (size[1] != B.size[0] )  // A  has dimensions mxn and B has dimensions nxp
    {
        throw std::invalid_argument(" dimensions not correct");
        return Matrix{};
    }

    std::vector<std::vector<double>> multiply_vector{};    // for store  data of multiply matrix

    for (size_t i = 0; i < size[0]; i++)   // row matrix A
    {
        std::vector<double> multiply_vector_row{};        // for store  row of data
        for (size_t j = 0; j < B.size[1]; j++)     // column martix B
        {
            double temp{};

            for (size_t k = 0; k < size[1]; k++)
                temp += Data[i][k] * B.Data[k][j];    //multiplying row  i  of  A  by column  j  of  B
            multiply_vector_row.push_back(temp);       //sotre a row data
        }
        multiply_vector.push_back(multiply_vector_row);   //store data of matrix
    }

    return Matrix{multiply_vector};    //return of multiply matrix
}
```

چهارمین ;[] operator است؛ کار این operator این است که یک reference به object می‌باشدکه بیانگر سطر است که خود آن object خود یک operator[] دارد.

```
std::vector<double>& Matrix::operator[](int row)  //overload operator[] to return a reference to an object that represents the row
{                                                 // and with itself has an appropriate operator[]
    return Data[row];
}
```

حال بعد از نوشتن class matrix مراحل بعدی پروژه را اجرا و ضمن تعریف چند تابع که برای حل مسئله لازم است .

توجه شود که prototype ها را در داخل فایلaphw2.h مینویسم که در زیر تمام آن ها قابل مشاهده است.

```
std::optional<double> det(Matrix&);     // return determinant of the given matrix
std::optional<Matrix> inv(Matrix&);      //return inverse of the given matrix
std::optional<Matrix> transpose(Matrix&);   //return the transpose of the matrix.
std::vector<std::vector<double>> getData(const char* filename, bool add_bias=true);  //to get data
size_t findMinNoOfMultiplications(std::vector<Matrix>& v);  // minimum number of required scalar multiplications
Matrix findWeights(const char*);              // return the proper weights
Matrix predict(const char*, Matrix& w, bool disp = false );    //return the estimated outputs
```

اولین تابع ؛ det می‌باشدکه ورودی آن یک ماتریس که به صورت reference به آن داده شده (دلیل اینکه ما ورودی را به صورت refernce می‌دهیم این است که باعث افزایش performance و صرفه‌جویی در حافظه می‌شود) کار این تابع حساب کردن مقدار دترمینان ماتریس است و این کار را به سادگی با توابعی که در class matrix تعریف شده به صورت زیر قابل پیاده سازی است.

```
std::optional<double> det(Matrix& A )   // return determinant of the given matrix
{
    return std::optional<double> {A.det()};
}
```

جواب Qusetion2 : اگر ورودی به &const Matrix یابد در class matrix در تابع det که قبلا تعریف شده یک const به خط اول اضافه می‌گرددبه صورت زیر:

```
double Matrix::det()  const      //determinant of the matrix
{
    if (size[0] != size[1])              //check  matrix is the squre form or not
    {
        std::cout << "the matrix is not in the square form " << std::endl;
        return 0.0;
    }
    else
        return det_funtion(Data, size[0]);    //return determinant of the matrix
}
```

دومین تابع؛inv است ؛ ورودی آن یک ماتریس می‌باشدکه به صورت reference به آن داده شده (دلایل این کار را در بالا شرح داده‌ایم) کار این تابع حساب کردن و برگرداندن ماتریس معکوس و این کار را به سادگی با توابعی که در class matrix تعریف شده انجام می‌دهیم و اگر ماتریس مربعی نبود باید چیزی برنگرداند .

```
std::optional<Matrix> inv(Matrix& A)            //return inverse of the given matrix
{
    std::array<size_t, 2> size_matrix{A.getSize()};
    if (size_matrix[0] != size_matrix[1])     //check  matrix is the squre form or not
        return std::nullopt;      // same as return { };

    return std::optional<Matrix>{A.inv()};
}
```

سومین تابع؛transpose می‌باشدکه ورودی آن یک ماتریس که به صورت reference به آن داده شده (دلایل این کار را در بالا شرح داده‌ایم) کار این تابع حساب کردن و برگرداندن ماتریس انتقال است که این کار را به سادگی با توابعی که در class matrix تعریف شده به صورت زیر قابل پیاده سازی است.

```
std::optional<Matrix> transpose(Matrix& A)    //return the transpose of the matrix
{
    return std::optional<Matrix>{A.T()};
}
```

چهارمین تابع؛getData است که دو ورودی دارد ؛یکی اسم فایل csv. و دیگری یک متغییر از جنس bool که مشخص میکندکه باید ستون bias باید باشد یا نه . الگوریتم این تابع دقیقا مشابه تابعload است که در class matrix شده و تنها تفاوت آن با تابع load این است که در آن جا تابع باید ماتریس برگرداند ولی این تابع باید یک وکتور دو بعدی برمی‌گرداند.

```
std::vector<std::vector<double>> getData(const char* filename, bool add_bias)   //get data from a .csv file
{
    std::ifstream input_file{ filename };   //  read .csv file
    std::vector<std::vector<double>> data_vector{};   //for store data

    while (!input_file.eof())    // while until end of file
    {
        std::string line{};       // for store each line
        std::string data{};       // for store each data
        std::vector<double> row{};  // for store each row

        std::getline(input_file, line);  // store each line

        if (static_cast<int>(line[0]) == 0)  // if line is empty break from loop
            break;
        if (add_bias)      // add bias or not
            row.push_back(1.);
        for (size_t i = 0; i < line.length(); i++)  // to get data from a line
        {
            if (line[i] != ',')
                data += line[i];
            else
            {
                row.push_back(std::stod(data));   // store data
                data = "";
            }
        }
        row.push_back(std::stod(data));   // store a row of data in a Data vector
        data_vector.push_back(row);
    }
    input_file.close();  //close file

    return data_vector;  //return data vector
}
```

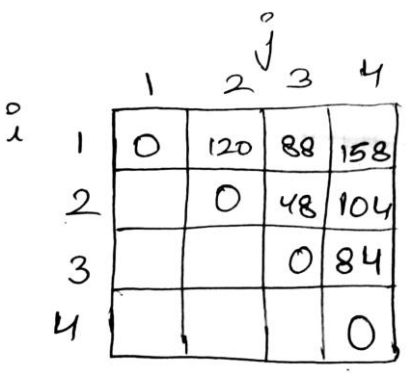
پنجمین تابع؛findminNoOfMultiplication است ؛کارش پیدا کردن حداقل تعداد ضرب ها در ضرب چند ماتریس است که به صورت ورودی به آن داده شده است . الگوریتم این تابع در کلاس تدریسیاری به طور کامل بررسی شده و اینجا فقط به ذکر چند نکته و روش پیاده سازی آن اکتفا میکنیم. برای حل این مسئله یک جدول فرض شده که المان های آن ؛ حداقل ضرب ها که جدولش به صورت زیر است.

در جدول i وj شماره ماتریس و آن را مطابق شکل بر اساس فرمول زیر خانه های آن را حساب و پر می‌کنیم

$$m[i,j]=\min(m[i,k]+m[k+1,j]+d(i-1)*d(k)*d(j)$$

در فرمول d سایز ماتریس است.

باید توجه کرد که وقتی i=j است حتما m[i,j]=0 است .



الگوریتم بالا به صورت زیر پیاده‌سازی شده.

```
size_t findMinNoOfMultiplications(std::vector<Matrix>& v)  // minimum number of required scalar multiplications
{
    std::vector<std::vector<size_t>> table{};     // for store m[i,j]
    int step{};

    for (size_t i = 0; i < v.size(); i++)   //make a table with elements of zero
    {
        std::vector<size_t> row(v.size());   // for store a row elements
        table.push_back(row);                // store a row elements of zero
    }
    while (step != v.size()-1)       //continue until fill the table
    {
        for (size_t i = 0; i < v.size(); i++)
        {
            for (size_t j = i + step + 1; j < v.size(); j++)  //step 1
            {
                if (j - i == 1)
                {
                    table[i][j] = v[i].getSize()[0] * v[i].getSize()[1] * v[j].getSize()[1];    //comput the m[i,j]: j-i=1
                    break;   // got next step
                }
                else  //j-i>1
                {
                    size_t min{};  // min multipy
                    bool  ful{ false };  //for initial min
                    for (size_t k = i; k < j; k++)
                    {
                        size_t temp{};
                        temp = table[i][k] + table[k + 1][j] + (v[i].getSize()[0]) * v[k].getSize()[1] * v[j].getSize()[1];   //compute the m[i,j]
                        if (!ful)  //initial min
                        {
                            ful = true;   //show it is first temp or not
                            min = temp;  // first temp is first min
                        }
                        min = min > temp ? temp : min;   //find min
                    }
                    table[i][j] = min;   //fill the table with min
                    break;  // go to next step
                }
            }
        }
        step++;
    }
    return table[0][v.size() - 1]; // return minimum number of multiplications
}
```

اکنون با کمک قابلیت های بدست آمده از توابع ذکر شده که برای کار با ماتریس است به سراغ هدف اصلی پروژه یعنی پیدا کردن وزن های مناسب از روی data هایی که در اختیار داریم می‌رویم .برای این منظور تابع findweight که ورودی آن اسم فایلی است که در آن data ها قرار دارد را تعریف میکنم .

طبق فرمول زیر می‌توان وزن ها را به سادگی حساب نمود.

$$Xw = y$$

$$w = [w_0, w_1 \ldots, w_6]^T$$

$$w = (X^T X)^{-1} X^T y$$

در این فرمول ها X ماتریس ویژگی ها و w ماتریس وزن و y ماتریس نمره های واقعی است که این ماتریس ها را به کمک , توابع ذکر شده و همچنین توابع موجود در class matrix به راحتی بدست

می‌آید. روش پیاده سازی این فرمول ها و نحوه به دست آوردن این ماتریس ها درکد به صورت کامنت ذکر شده است .

```
Matrix findWeights(const char* name_file)  //find weights
{
    Matrix data{getData(name_file,true)}; // get data from .csv file as a matrix
    Matrix Y{ data.col(7) };  // get real grade as a matrix
    Matrix X{ data.delCol(7) };  // get feature as a matrix
    Matrix W{ 7,1,false };  //define and initial weights matrix
    W = Matrix{ X.T() * X}.inv()*X.T() * Y;  //calculate weights

    return W; //return weights as a matrix
}
```

حال به سراغ تخمین نمره ها براساس وزن های بدست آمده در مرحله قبل رفته و برای این منظور تابع predict را تعریف میکنیم که ورودی آن نام فایلی است که data ها در آن قرار دارند و یک متغییر از جنس bool است که نشان می‌دهد آیا باید نمرات تخمینی و واقعی نمایش داده شود یا نه و خروجی تابع نمرات تخمینی به صورت یک ماتریس است . نحوه پیاده سازی این تابع به صورت زیر است.

```
Matrix predict(const char* name_file, Matrix& W, bool disp) // calculate predict grade and show them
{
    Matrix data{ getData(name_file,true) };    //get data as a matrix
    Matrix Y{ data.col(7) };   // get real grade as a matrix
    Matrix X{ data.delCol(7) };  // get feature as a matrix
    Matrix pridict_grade{ X * W };  //predict grade and save as a matrix

    if (disp)  //to display predict grade and real grade
    {
        std::cout << "No" << std::setw(28) << "Real Grade" << std::setw(25) << "Estimated Grade" << std::endl;
        for (size_t i = 0; i < 28; i++)
            std::cout << "**";
        std::cout << std::endl;

        for (size_t i = 0; i < pridict_grade.getSize()[0]; i++)
        {
            for (size_t j = 0; j < 3; j++)
            {
                if (j == 0)  //number of studen
                    std::cout << std::setiosflags(std::ios::left) << std::setw(20) << i + 1;
                if (j == 1)
                    std::cout << std::setw(20) << Y[i][0]; //real grade
                if (j == 2)
                    std::cout << std::setw(20) << pridict_grade[i][0];  //predict garde
            }
            std::cout << std::endl;
        }
    }

    return pridict_grade; //return predict grade as a matrix
}
```

توجه: توضیحات مربوط به پیاده سازی الگوریتم ها در داخل کد به صورت کامنت نوشته شده است و در این گزارش کار سعی شده در مورد الگوریتم توابع توضیح داده شود.

توجه: این پروژه همانطورکه در عکس ها مشخص است ابتدا در visual studio 2019 نوشته شده و سپس در visual studio code به وسیله docker اجرا شده و نتایج صفحه بعد بدست آمده است.

نتایج google test به شرح زیر است.

```
C:\Users\mjami\Documents\vscode\ap1398-2-hw2-master>docker run --rm ap1398/hw2
RUNNING TESTS ...
[==========] Running 9 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 9 tests from APHW2Test
[ RUN      ] APHW2Test.AddressingTest1
[       OK ] APHW2Test.AddressingTest1 (0 ms)
[ RUN      ] APHW2Test.SumTest
[       OK ] APHW2Test.SumTest (0 ms)
[ RUN      ] APHW2Test.MultiplicationTest
[       OK ] APHW2Test.MultiplicationTest (0 ms)
[ RUN      ] APHW2Test.TransposeTest
[       OK ] APHW2Test.TransposeTest (0 ms)
[ RUN      ] APHW2Test.InversionTest
[       OK ] APHW2Test.InversionTest (0 ms)
[ RUN      ] APHW2Test.DeterminantTest
[       OK ] APHW2Test.DeterminantTest (0 ms)
[ RUN      ] APHW2Test.MultiplicationTest2
[       OK ] APHW2Test.MultiplicationTest2 (1 ms)
[ RUN      ] APHW2Test.getDataTest
[       OK ] APHW2Test.getDataTest (3 ms)
[ RUN      ] APHW2Test.estimationTest
[       OK ] APHW2Test.estimationTest (1063 ms)
[----------] 9 tests from APHW2Test (1071 ms total)

[----------] Global test environment tear-down
[==========] 9 tests from 1 test suite ran. (1072 ms total)
[  PASSED  ] 9 tests.
Here!
<<<SUCCESS>>>
```

همانطورکه قابل مشاهده است نتایج google test موفقیت آمیز بوده همچین خروجی تابع predict به صورت زیر است.

No	Real Grade	Estimated Grade
1	14.23	14.1858
2	15.76	16.6979
3	9.99	9.58491
4	13.39	13.0088
5	11.26	11.945
6	12.74	13.0889
7	10.11	10.2861
8	14.7	15.1742
9	13.34	12.7991
10	12.92	12.93
11	17.72	17.3995
12	12.81	13.4341
13	16.03	16.3821
14	14.98	15.4855
15	9.51	9.10332
16	17.95	17.4907
17	7.83	6.92275
18	20	21.0807
19	8.91	9.47516
20	10.27	10.7996
21	18.62	18.5335
22	13.31	12.839
23	8.1	8.72082
24	12.34	12.5558
25	19.48	19.0568
26	9.27	9.82346
27	8.49	8.51551
28	13.74	13.9633

تمام