به نام خدا

گزارش تمرین ۱

محمد برآبادی ۹۷۲۳۰۱۵

در ابتدا با استفاده از فیلم جلسه اول کلاس تدریسیاری تنظیمات اولیه را انجام دادم تا VS Code را آماده کرده و با اتصال به Host 127.0.0.1 شروع به کدزنی کردم.

- فایل hw1.h :

با توجه به توضیحات داده شده در github ابتدا در فایل .h تابع‌ها را در یک namespace تعریف میکنیم.

```h
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);

        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);
};
#endif //AP_HW1_H
```

- فایل hw1.cpp:

در مرحله بعد قصد داریم که هر تابع را پیاده‌سازی کنیم با توجه به نتیجه‌ای که از ما خواسته شده و همچنین unit_test موجود در پوشه.

1. Zeros:

نکته مهمی که در ابتدا باید ذکر شود استفاده از vector میباشد و برای همین برای اضافه کردن هر مقدار باید آن را با دستور push_back اضافه کنیم.

هر سطر را در یک for ساخته و در ماتریس قرار میدهیم.

```
6   ////1//////////////////////////////////////////////////////////////
7   Matrix algebra::zeros(size_t n, size_t m)
8   {
9       Matrix matrix{};
10      std::vector<double> mat{};
11      for(int i{0}; i<n; i++)
12      {
13          for(int j{0}; j<m; j++)
14          {
15              mat.push_back(0);
16          }
17          matrix.push_back(mat);
18          mat.clear();
19      }
20      return matrix;
21  }
```

2. Ones:

مانند تابع قبل میباشد و فقط تنها فرق این میباشد که باید همه ماتریس را با ۱ به جای ۰ پر کنیم.

```
22  ////2//////////////////////////////////////////////////////////////
23  Matrix algebra::ones(size_t n, size_t m)
24  {
25      Matrix matrix{};
26      std::vector<double> mat{};
27      for(int i{0}; i<n; i++)
28      {
29          for(int j{0}; j<m; j++)
30          {
31              mat.push_back(1);
32          }
33          matrix.push_back(mat);
34          mat.clear();
35      }
36      return matrix;
37  }
```

3. Random:

در این تابع قصد داریم ماتریس را به صورت رندوم با اعدادی بین min و max پر کنیم. یکی از نکاتی که باید رعایت شود این میباشد که مقدار min از max بزرگتر نباشد چون به صورت منطقی با مشکل مواجه میشویم.با توجه به صورت تمرین از rand و srand نمیتوانیم استفاده کنیم و از random استفاده میکنیم و دستورات مربوط به آن را از اینترنت پیدا کردیم.

```
38  ////3//////////////////////////////////////////////////////
39  Matrix algebra::random(size_t n, size_t m, double min, double max)
40  {
41      if(min > max)
42          throw std::logic_error("Caution: min cannot be greater than max");
43
44      std::random_device rd;
45      std::default_random_engine generator(rd());
46      std::uniform_real_distribution<double> distribution(min,max);
47
48
49      Matrix matrix{};
50      std::vector<double> mat{};
51      for(int i{0}; i<n; i++)
52      {
53          for(int j{0}; j<m; j++)
54          {
55              double number = distribution(generator);
56              mat.push_back(number);
57          }
58          matrix.push_back(mat);
59          mat.clear();
60      }
61      return matrix;
62  }
```

4. Show:

در این تابع ماتریس را نمایش میدهیم ،با همان ملاحظاتی که گفته شده است

```
63  ////4//////////////////////////////////////////////////////////////
64  void algebra::show(const Matrix &matrix)
65  {
66      for (int i{0}; i < matrix.size(); i++)
67      {
68          for (int j{0}; j < matrix[i].size(); j++)
69          {
70              std::cout << std::setw(3) << matrix[i][j] << " ";
71          }
72          std::cout << std::endl;
73      }
74  }
```

5. Multiply:

ما در این تابع قصد داریم یک عدد را در مانریس ضرب کنیم.مانند تابع ۱و۲ که مقدار صفر و یک در ماتریس قرار میدادیم اینبار هر درایه ماتریس را ضرب میکنیم در عدد وارد شده و در ماتریس قرار میدهیم.

```
75  ////5/////////////////////////////////////////////////////////
76  Matrix algebra::multiply(const Matrix &matrix, double c)
77  {
78      Matrix matrix_1{};
79      std::vector<double> mat{};
80      for (int i{0}; i < matrix.size(); i++)
81      {
82          for (int j{0}; j < matrix[i].size(); j++)
83          {
84              mat.push_back(matrix[i][j] * c);
85          }
86          matrix_1.push_back(mat);
87          mat.clear();
88      }
89      return matrix_1;
90  }
```

6. Multiply:

این تابع ضرب دو ماتریس را داریم ولی باید در ابتدا بررسی شود که ماتریس ها خالی نباشند و همچنین امکان ضرب آن‌ها وجود داشته باشد یعنی اینکه تعداد ستون‌های ماتریس اول با تعداد سطرهای ماتریس دوم برابر باشد.

```
91  ////6/////////////////////////////////////////////////////////
92  Matrix algebra::multiply(const Matrix &matrix1, const Matrix &matrix2)
93  {
94      size_t row1{matrix1.size()};
95      size_t row2{matrix2.size()};
96      size_t column1{};
97      size_t column2{};
98
99      if (row1 > 0)
100         column1 = matrix1[0].size();
101     else
102         column1 = 0;
103
104     if (row2 > 0)
105         column2 = matrix2[0].size();
106     else
107         column2 = 0;
108
109     std::vector<double> mat{};
110     Matrix matrix{};
111     double sum{0};
112
```

اگر این شرایط وجود داشته باشد سپس ضرب درایه‌ها را انجام میدهیم.

```
113     if (column1 != row2)
114     {
115         throw std::logic_error("Caution: matrices with wrong dimensions cannot be multiplied");
116     }
117 
118     for (int i{0}; i < row1; i++)
119     {
120         for (int j{0}; j < column2; j++)
121         {
122             for (int k{0}; k < column1; k++)
123             {
124                 sum += matrix1[i][k] * matrix2[k][j];
125             }
126             mat.push_back(sum);
127             sum = 0;
128         }
129         matrix.push_back(mat);
130         mat.clear();
131     }
132     return matrix;
133 }
```

7. Sum:

مانند تابع ضرب با عدد اسکالر، هر درایه با یک عدد جمع میشوند.

```
134 ////7/////////////////////////////////////////////////////////////
135 Matrix algebra::sum(const Matrix &matrix, double c)
136 {
137     std::vector<double> mat{};
138     Matrix matrix_1{};
139     for (int i{0}; i < matrix.size(); i++)
140     {
141         for (int j{0}; j < matrix[i].size(); j++)
142         {
143             mat.push_back(matrix[i][j] + c);
144         }
145         matrix_1.push_back(mat);
146         mat.clear();
147     }
148     return matrix_1;
149 }
```

8. Sum:

باز باید بررسی شود که ماتریس‌ها تهی نباشند و برای انجام جمع باید ستون‌ها و سطرها با هم برابر باشند و در نهایت هم درایه‌های متناظر با هم جمع میشوند و در ماتریس جدید قرار میگیرند.

```
150   ////8/////////////////////////////////////////////////////////
151   Matrix algebra::sum(const Matrix &matrix1, const Matrix &matrix2)
152   {
153       size_t row1{matrix1.size()};
154       size_t row2{matrix2.size()};
155       size_t column1{};
156       size_t column2{};
157
158       if (row1 > 0)
159           column1 = matrix1[0].size();
160       else
161           column1 = 0;
162
163       if (row2 > 0)
164           column2 = matrix2[0].size();
165       else
166           column2 = 0;
167
168       std::vector<double> mat{};
169       Matrix matrix{};
170       double sum{0};
171
```

```
172       if (row1 != row2 || column1 != column2)
173       {
174           throw std::logic_error("Caution: matrices with wrong dimensions cannot be multiplied");
175       }
176
177       for (int i{0}; i < row1; i++)
178       {
179           for (int j{0}; j < column1; j++)
180           {
181               sum = matrix1[i][j] + matrix2[i][j];
182               mat.push_back(sum);
183           }
184           matrix.push_back(mat);
185           mat.clear();
186       }
187       return matrix;
188   }
```

9. Transpose:

در ابتدا بررسی میشود که آیا ماتریس تهی است یا خیر و اگر تهی باشد، ترانهاده آن همان ماتریس تهی میباشد. و در با حرکت در ستون ماتریس ، سطر ماتریس جدید را میسازیم و به این شکل ماتریس ترانهاده ایجاد میشود.

```
189   ////9//////////////////////////////////////////////////////////////
190   Matrix algebra::transpose(const Matrix& matrix)
191   {
192       if(matrix.empty())
193           return matrix;
194       size_t row{matrix.size()};
195       size_t column{matrix[0].size()};
196       Matrix matrix_1{};
197       std::vector<double> mat{};
198       for (int i{0}; i < column; i++)
199       {
200           for (int j{0}; j < row; j++)
201           {
202               mat.push_back(matrix[j][i]);
203           }
204           matrix_1.push_back(mat);
205           mat.clear();
206       }
207       return matrix_1;
208   }
```

10.Minor:

ماینور n,m به این معنا است که سطر n و ستونm را حذف کنیم.

```
209   ////10//////////////////////////////////////////////////////////////
210   Matrix algebra::minor(const Matrix& matrix, size_t n, size_t m)
211   {
212       size_t row{matrix.size()};
213       size_t column{matrix[0].size()};
214       Matrix matrix_1{};
215       std::vector<double> mat{};
216       for (int i{0}; i < row; i++)
217       {
218           if(i != n)
219           {
220               for (int j{0}; j < column; j++)
221               {
222                   if(j != m)
223                       mat.push_back(matrix[i][j]);
224               }
225               matrix_1.push_back(mat);
226               mat.clear();
227           }
228       }
229       return matrix_1;
230   }
```

11.Determinant:

ماتریس اگر تهی باشد دترمینان ۱ را خروجی میدهیم و اگر ماتریس مربعی نباشد اصلا دترمینانی وجود ندارد و در غیر این دو صورت میتوانیم دترمینان را بدست آوریم.

از روش معمولی محاسبه دترمینان استفاده میکنیم به این صورت که یک سطر را درنظر میگیریم و ماینورهای هر درایه را بدست آورده و در آن ضرب میکنیم.(از الگوریتم بازگشتی استفاده میکنیم چون نیاز داریم به مایتور ۲*۲ برسیم)

```
231  ////11//////////////////////////////////////////////////////////
232  double algebra::determinant(const Matrix& matrix)
233  {
234      if(matrix.empty())
235          return 1;
236      double det{0};
237      size_t row{matrix.size()};
238      size_t column{matrix[0].size()};
239      Matrix matrix_1{};
240      std::vector<double> mat{};
241      if(row != column)
242          throw std::logic_error("Caution: non-square matrices have no determinant");
243      if(row == 2)
244          return matrix[0][0] * matrix[1][1] - matrix[1][0] * matrix[0][1];
245      for (int j{0}; j < column; j++)
246      {
247          det += matrix[0][j] * determinant(minor(matrix,0,j)) * pow(-1, j);
248
249      }
250      return det;
251  }
```

12.Inverse:

برای بدست آوردن ماتریس معکوس ابتدا نیاز داریم ماتریس کهاد را بدست آوریم و با اعمال مثبت و منفی یکی در میان به ماتریس همسازه میرسیم و با ترانهاده کردن آن به ماتریس الحاقی میرسیم و در معکوس دترمینان ضرب میکنیم و به ماتریس وارون میرسیم.

اما چند نکته مهم وجود دارد که اگر ماتریس تهی باشد، وارون آن هم تهی است و اگر ماتریس مربعی نباشد دیگر وارونی وجود ندارد و اگر دترمینان هم صفر شود باز وارونی وجود ندارد.

```cpp
////12/////////////////////////////////////////////////////////////
Matrix algebra::inverse(const Matrix& matrix)
{
    if(matrix.empty())
        return matrix;
    size_t row{matrix.size()};
    size_t column{matrix[0].size()};
    Matrix matrix_1{};
    double det;
    std::vector<double> mat{};
    if(row != column)
        throw std::logic_error("Caution: non-square matrices have no inverse");
    det = determinant(matrix);
    if(det == 0)
        throw std::logic_error("Caution: singular matrices have no inverse");
    for (int i{0}; i < row; i++)
    {
        for (int j{0}; j < column; j++)
        {
            mat.push_back(determinant(minor(matrix,i,j)) * pow(-1, i+j));
        }
        matrix_1.push_back(mat);
        mat.clear();
    }
    matrix_1 = transpose(matrix_1);
    matrix_1 = multiply(matrix_1, 1/det);
    return matrix_1;
}
```

13.Concatenate:

باز هم بررسی میکنیم که ماتریس‌ها تهی نباشند و سپس با توجه به اینکه کاربر axis =0 or 1 وارد کرده است دو ماتریس را به هم وصل میکنیم.

```cpp
////13/////////////////////////////////////////////////////////////
Matrix algebra::concatenate(const Matrix& matrix1, const Matrix& matrix2, int axis=0)
{
    size_t row1{matrix1.size()};
    size_t row2{matrix2.size()};
    size_t column1{};
    size_t column2{};
    Matrix matrix_1{};
    std::vector<double> mat{};
    if (row1 > 0)
        column1 = matrix1[0].size();
    else
        column1 = 0;

    if (row2 > 0)
        column2 = matrix2[0].size();
    else
        column2 = 0;
    if(axis == 0)
    {
        if(column1 != column2)
            throw std::logic_error("Caution: matrices with wrong dimensions cannot be concatenated");
        for (int i{0}; i < row1 + row2; i++)
        {
            for (int j{0}; j < column1; j++)
            {
                if(i < row1)
                    mat.push_back(matrix1[i][j]);
```

وقتی دو ماتریس را پشت هم وصل کنیم پس تعداد سطرها افزایش میابد و باید ستون‌های دو ماتریس برابر باشند و وقتی دو ماتریس کنار هم وصل کنیم پس تعداد ستون‌ها افزایش میابد و باید سطرهای دو ماتریس با هم برابر باشند تا این عمل اتصال انجام شود.

```
306                 if(i < row1)
307                     mat.push_back(matrix1[i][j]);
308                 else
309                     mat.push_back(matrix2[i - row1][j]);
310             }
311             matrix_1.push_back(mat);
312             mat.clear();
313         }
314     }
315     else
316     {
317         if(row1 != row2)
318             throw std::logic_error("Caution: matrices with wrong dimensions cannot be concatenated");
319         for (int i{0}; i < row1; i++)
320         {
321             for (int j{0}; j < column1 + column2; j++)
322             {
323                 if(j < column1)
324                     mat.push_back(matrix1[i][j]);
325                 else
326                     mat.push_back(matrix2[i][j - column1]);
327             }
328             matrix_1.push_back(mat);
329             mat.clear();
330         }
331     }
332     return matrix_1;
333 }
```

14.Ero_swap:

در این تابع دو سطر را با هم جابه‌جا میکنیم فقط باید بررسی شود که عدد سطری که وارد میشود از تعداد سطرهای ماتریس بیشتر نباشد و هنگامی که ماتریس ورودی را در ماتریس دیگری میریزیم اگر به آن سطرهای موردنظر برسیم باید جابه‌جا این عملیات انجام شود.

```
334 ////14//////////////////////////////////////////////////////////
335 Matrix algebra::ero_swap(const Matrix& matrix, size_t r1, size_t r2)
336 {
337     size_t row{matrix.size()};
338     size_t column{matrix[0].size()};
339     Matrix matrix_1{};
340     std::vector<double> mat{};
341     if(r1 >= row or r2 >= row)
342         throw std::logic_error("Caution: r1 or r2 inputs are out of range");
343     for(int i{0}; i<row; i++)
344     {
345         for(int j{0}; j<column; j++)
346         {
347             if(i == r1)
348                 mat.push_back(matrix[r2][j]);
349             else if(i == r2)
350                 mat.push_back(matrix[r1][j]);
351             else
352                 mat.push_back(matrix[i][j]);
353         }
354         matrix_1.push_back(mat);
355         mat.clear();
356     }
357     return matrix_1;
358 }
```

15.Ero_multiply:

اگر به سطر موردنظر رسیده باشیم تمام درایه‌های آن سطر را در یک عدد اسکالر ضرب میکنیم.

```
359  ////15//////////////////////////////////////////////////////////
360  Matrix algebra::ero_multiply(const Matrix& matrix, size_t r, double c)
361  {
362      size_t row{matrix.size()};
363      size_t column{matrix[0].size()};
364      Matrix matrix_1{};
365      std::vector<double> mat{};
366      for(int i{0}; i<row; i++)
367      {
368          for(int j{0}; j<column; j++)
369          {
370              if(i == r)
371                  mat.push_back(matrix[i][j] * c);
372              else
373                  mat.push_back(matrix[i][j]);
374          }
375          matrix_1.push_back(mat);
376          mat.clear();
377      }
378      return matrix_1;
379  }
```

16.Ero_sum:

یک سطر را در عدد اسکالری ضرب کرده و به سزری دیگر اضافه میکنیم.در کد هم به همین روش انجام دادم و باقی درایه‌ها خودشان قرار میگیرند و فقط اگر به سطر موردنظر برسیم این الگوریتم را پیاده سازی میکنیم.

```
359  ////15//////////////////////////////////////////////////////////
360  Matrix algebra::ero_multiply(const Matrix& matrix, size_t r, double c)
361  {
362      size_t row{matrix.size()};
363      size_t column{matrix[0].size()};
364      Matrix matrix_1{};
365      std::vector<double> mat{};
366      for(int i{0}; i<row; i++)
367      {
368          for(int j{0}; j<column; j++)
369          {
370              if(i == r)
371                  mat.push_back(matrix[i][j] * c);
372              else
373                  mat.push_back(matrix[i][j]);
374          }
375          matrix_1.push_back(mat);
376          mat.clear();
377      }
378      return matrix_1;
379  }
```

17.Upper_triangular:

اگر ماتریس مربعی نباشد نمیتوان ماتریس بالا مثلثی بدست آورد و از تابع ero_sum استفاده میکنیم تا درایه‌های موردنظر را صفر کنیم. سطر اول را در ضریبی از درایه اول سطر اول و درایه اول سطرهای بعدی ضرب میکنیم و با همان سطر جمع میکنیم.

نکته مهم این است که اگر یکی از درایه‌های روی قطر صفر باشد آنگاه مخرج ضریب صفر میشود و برای این که به این مشکل برخورد نکنیم این سطر را با سطر زیر آن جابه‌جا میکنیم.

```
401  ////17//////////////////////////////////////////////////////////
402  Matrix algebra::upper_triangular(const Matrix& matrix)
403  {
404      if(matrix.empty())
405          return matrix;
406      size_t row{matrix.size()};
407      size_t column{matrix[0].size()};
408      size_t i{0};
409      if(row != column)
410          throw std::logic_error("Caution: non-square matrices have no upper triangular form");
411      Matrix matrix_1{matrix};
412      while(i < row)
413      {
414          if(matrix_1[i][i] == 0)
415              matrix_1 = ero_swap(matrix_1, i, i + 1);
416          i++;
417      }
418      for(int i{0}; i<row; i++)
419      {
420          for(int j{i+1}; j<row; j++)
421          {
422              matrix_1 = ero_sum(matrix_1, i, -matrix_1[j][i]/matrix_1[i][i], j);
423          }
424      }
425      return matrix_1;
426  }
```

https://github.com/MBW0lf/AP_HW1

با تشکر