به نام خدا

تمرین سری اول

نام درس:

برنامه نویسی پیشرفته

نام استاد:

دکتر جهانشاهی

نگارش :

سجاد قدیری

فهرست **صفحه**

تمرین سری اول

1-1-تابع zeros و ones

در این 2 تابع ابتدا خطاهای احتمالی که ممکن است از طرف کاربر رخ دهد را مورد بررسی قرار دادیم.

اینکه کاربر ابعاد ماتریس را برابر با صفر قرار ندهد و اگر به اشتباه این اتفاق رخ داد یک پیام اخطار برای او بازگردانده شود. اگر این اتفاق رخ نداد ، یک بردار با ابعاد 1*m که با مقادیر اولیه صفر مقداردهی شده اند را ساخته و به کمک این بردار ماتریس با ابعاد n*1 را پر می‌کنیم. در واقعا با توجه به تعریف ماتریسی که در فایل header استفاده کردیم ماتریس های ما نوعی بردارهایی از بردار هستند. (vector of vectors)

```
3  ////////////////////////////////////////////// zeros function ////////////////////////////////////////////////
4  Matrix algebra::zeros(size_t n , size_t m)
5  {    // we creating a 1*m vector with zero initialization and we put them in each n row,
6       // so finally we have a n*m matrix
7       if( (m != 0) && (n != 0) )
8       {
9       Matrix matrix_1 ( n , std::vector<double>(m, 0));
10      return matrix_1;
11      }
12      else
13      {    // we throw back this error message if the user set the dimention of matrix zero
14           throw std::logic_error( "Caution: dimention of matrix can not be zero" );
15      }
16 }
17 ////////////////////////////////////////////// ones function ////////////////////////////////////////////////
18 Matrix algebra::ones(size_t n, size_t m)
19 {    // we creating a 1*m vector with one initialization and we put them in each n row,
20      // so finally we have a n*m matrix
21      if( (m != 0) && (n != 0) )
22      {
23      Matrix matrix_2 ( n , std::vector<double>(m,1));
24      return matrix_2;
25      }
26      else
27      {    // we throw back this error message if the user set the dimention of matrix zero
28           throw std::logic_error( "Caution: dimention of matrix can not be zero" );
29      }
30 }
```

2-1- تابع random و show

در این 2 تابع ابتدا خطاهای احتمالی که ممکن است از طرف کاربر رخ دهد را مورد بررسی قرار دادیم.

اینکه کاربر مقدار min را از max بیشتر مقداردهی نکند(و برعکس)و اگر به اشتباه این اتفاق رخ داد یک پیام اخطار برای او بازگردانده شود.اگر این اتفاق رخ نداد، با استفاده از یک شی از random_device

و با محاسبه اختلاف مقدار min وmax و سپس ایجاد یک ماتریس با ابعاد دلخواه کاربر(n*m) که مقدار آن با همان دستورdist به صورت رندوم انتخاب می‌شود.

```
32  ////////////////////////////////////////////// random function ////////////////////////////////////////////
33  Matrix algebra::random(size_t n, size_t m, double min, double max)
34      // At first we check that min value is less than max ro not with if condition
35  {   if( min <= max )
36      {
37      std::random_device ran_dev ;
38      std::uniform_real_distribution<double> dist(min,max);
39      Matrix matrix_3(n , std::vector<double>(m, dist(ran_dev)));
40      return matrix_3;
41      }
42      else
43      {   // we throw back this error message if the user set the min value greater than max value
44          throw std::logic_error( "Caution: min cannot be greater than max" );
45      }
46  }
47  ////////////////////////////////////////////// show function ////////////////////////////////////////////
48  void algebra::show(const Matrix& matrix)
49  {
50
51  for (size_t i {}; i < matrix.size(); i++) {
52          for (size_t j {}; j < matrix[i].size(); j++) {
53              std::cout << std::setw(15) << matrix[i][j];
54          }
55          std::cout << std::endl;
56      }
57  }
```

همچنین در تابع show با استفاده از دستور set width که در کتابخانه iomanip قرار دارد می‌توانیم خروجی را در موقعیت دلخواهی از صفحه خروجی قرار دهیم که ما بر روی 15 قرار دادیم و به این ترتیب ماتریس هایی به فرم زیر در خروجی چاپ شد:

```
[ RUN      ] HW1Test.RANDOM1
random matrix [-5, 7)
        5.09896        5.09896        5.09896        5.09896
        5.09896        5.09896        5.09896        5.09896
        5.09896        5.09896        5.09896        5.09896
        5.09896        5.09896        5.09896        5.09896
```

3-1- تابع multiply

در این تابع ابتدا یک ماتریس با مقادیر اولیه تماما صفر به کمک تابع zeros که قبلا ساخته ایم می‌سازیم.

ابعاد این ماتریس را نیز همان ابعاد ماتریسی که در ورودی تابع گرفتیم در نظر می‌گیریم.

سپس با 2 حلقه تو در تو المان به المان درایه های ماتریس را در اسکالر c که در ورودی تابع گرفتیم ضرب می‌کنیم و سپس نتیجه را در خروجی برمی‌گردانیم.

```
59  ////////////////////////////////////////////// multiply scalar function ////////////////////////////////////////////
60  Matrix algebra::multiply(const Matrix& matrix, double c)
61  {
62      //we are Creating a matrix of zeros with same dimention of input matrix
63      Matrix result { algebra::zeros(matrix.size(), matrix[0].size()) };
64
65      //we are multiplying each element of the matrix by the constant with 2 for loop
66      for (size_t i {}; i < matrix.size(); i++)
67      {
68          for (size_t j {}; j < matrix[i].size(); j++) {
69              result[i][j] = matrix[i][j] * c ;
70          }
71      }
72      return result;        // Returning the result
73  }
```

برای ضرب 2 ماتریس نیز ابتدا خطاهای احتمالی که ممکن است از طرف کاربر رخ دهد را مورد بررسی قرار دادیم. اینکه کاربر 2 ماتریس خالی وارد نکرده باشد و یا اینکه اگر یکی از آنها خالی بود پیام اخطاری

از جنس logic_error و یا اینکه از لحاظ ابعاد ماتریس ها، مشکلی در ضرب 2 ماتریس وجود داشت پیام اخطاری برگردانده شود.

```
74  ////////////////////////////////////////////// multiply 2 matrixes function ////////////////////////////////////////////
75  Matrix algebra::multiply(const Matrix& matrix1, const Matrix& matrix2)
76  {
77      if( matrix1.empty() && matrix2.empty() )
78      {
79          return Matrix{};  // return an empty matrix
80      }
81      else if( matrix1.empty() || matrix2.empty() )
82      {
83          throw std::logic_error( "Caution: matrices with wrong dimensions cannot be multiplied" );
84      }
85      if( matrix1[0].size() != matrix2.size())
86      {
87          throw std::logic_error( "Caution: matrices with wrong dimensions cannot be multiplied" );
88      }
89      Matrix temp { algebra::zeros(matrix1.size() ,matrix2[0].size()) };
90      for (size_t i{} ; i < temp.size() ; i++)
91          for(size_t j{} ; j < temp[i].size() ; j++)
92              for(size_t k{} ; k < matrix1[0].size() ; k++)
93                  temp[i][j] += matrix1[i][k] * matrix2[k][j] ;
94
95      return temp;    // Returning the temp
96  }
```

4-1-تابع sum

در این تابع ابتدا خطاهای احتمالی که ممکن است از طرف کاربر رخ دهد را مورد بررسی قرار دادیم. اینکه کاربر ماتریس خالی وارد نکرده باشد و اگر این اتفاق افتاد یک ماتریس خالی برگردانده می‌شود.

سپس یک ماتریس با مقادیر اولیه تماما صفر به کمک تابع zeros که قبلا ساخته ایم می‌سازیم.

ابعاد این ماتریس را نیز همان ابعاد ماتریسی که در ورودی تابع گرفتیم در نظر می‌گیریم.

سپس با 2 حلقه تو در تو المان به المان درایه های ماتریس را با اسکالر c که در ورودی تابع گرفتیم جمع می‌کنیم و سپس نتیجه را در خروجی برمی‌گردانیم.

```
98  ////////////////////////////////////////////// sum scalar with matrix function ////////////////////////////
99  Matrix algebra::sum(const Matrix& matrix, double c)
100 {
101     if(  matrix.empty() )
102         return Matrix{};
103
104     Matrix result { algebra::zeros(matrix.size() ,matrix[0].size()) };
105     for(size_t i{} ; i < matrix.size() ; i++)
106         for(size_t j{} ; j < matrix[0].size() ; j++)
107             result[i][j] = matrix[i][j] + c ;
108
109     return result;
110 }
111 ////////////////////////////////////////////// sum 2 matrixes function ////////////////////////////////////
112 Matrix algebra::sum(const Matrix& matrix1, const Matrix& matrix2)
113 {
114     if( matrix1.empty() && matrix2.empty() )
115         return Matrix{};
116
117     if ( (matrix1.size() != matrix2.size()) || (matrix1[0].size() != matrix2[0].size()) )
118     {
119         throw std::invalid_argument( "Caution: matrices with wrong dimensions cannot be summed" );
120     }
121
122     Matrix result { algebra::zeros(matrix1.size() ,matrix1[0].size()) };
123     for(size_t i{} ; i < matrix1.size() ; i++)
124         for(size_t j{} ; j < matrix2[0].size() ; j++)
125             result[i][j] = matrix1[i][j] + matrix2[i][j];
126
127     return result ;
128 }
```

برای جمع 2 ماتریس نیز به همین شکل ابتدا خالی نبودن مانریس ها را بررسی می‌کنیم. مورد دیگری که بررسی کردیم یکسان بودن ابعاد ماتریس ها برای جمع پذیر بودن است و اگر اینطور نبود یک خطای منطقی برمی‌گردانیم. سپس همانند قبل با 2 حلقه تو در تو المان های متناظر هر ماتریس را با دیگری جمع می‌کنیم.

5-1-تابع Transpose و Minor

برای ماتریس ترنسپوز ابتدا خطای احتمالی را بررسی کردیم. اگر کاربر ماتریس خالی وارد کند یک ماتریس خالی برمی‌گردانیم. سپس یک ماتریس با مقادیر اولیه تماما صفر به کمک تابع zeros که قبلا ساخته ایم می‌سازیم. ابعاد این ماتریس را نیز **برعکس** ابعاد ماتریسی که در ورودی تابع گرفتیم در نظر می‌گیریم.

حال با 2 حلقه تو در تو هر المان از ماتریس ورودی را در المانی با ایندکس برعکس از ماتریس result قرار می‌دهیم.

برای تابع ماینور ابتدا یک ماتریس با همان ابعاد ماتریس ورودی مقداردهی اولیه می‌کنیم. سپس با متد erase می‌توانیم ردیف دلخواهی را از یک ماتریس حذف کنیم . سپس با عمل ترنسپوز توانستیم ستون مورد نظر ماتریس را نیز حذف کنیم.

```cpp
////////////////////////////////////////// Transpose function ////////////////////////////////////////
Matrix algebra::transpose(const Matrix& matrix)
{
    if( matrix.empty() )
    {
        return Matrix{};
    }
    Matrix result { algebra::zeros(matrix[0].size() ,matrix.size()) };
    for(size_t i{} ; i < matrix.size() ; i++)
        for(size_t j{} ; j < matrix[0].size() ; j++)
            result[j][i] = matrix[i][j];

    return result;
}
////////////////////////////////////////// Minor function ////////////////////////////////////////
Matrix algebra::minor(const Matrix& matrix, size_t n, size_t m)
{
    Matrix result {matrix};                 // initialize the result matrix with input matrix
    result.erase(result.begin() + n);       // erase a row with erase method in vector library
    result = algebra::transpose(result);    // we transpose the matrix to use erase method for erasing a row
    result.erase(result.begin() + m);       // erase a column with erase method in vector library
    result = algebra::transpose(result);    // we transpose the final matrix to return
    return result;
}
```

6-1-تابع determinant

برای تابع دترمینان ابتدا چک می‌کنیم که کاربر ماتریس خالی وارد نکرده باشد در غیر اینصورت مقدار 1 را برمی‌گردانیم.(در خود توضیحات گیت هاب گفته شده بود)

سپس این خطای احتمالی را چک می‌کنیم که ماتریس ورودی مربعی باشد در غیر اینصورت یک اخطار منطقی برمی‌گردانیم.

برای محاسبه دترمینان به صورت بازگشتی از خود تابع دترمینان در درون خودش استفاده می‌کنیم و در نهایت وقتی به یک عدد اسکالر رسیدیم آن را برمی‌گردانیم.

در درون حلقه نیز ابتدا ماینور را محاسبه کرده و سپس ماتریس الحاقی را محاسبه می‌کنیم.

```
154 ////////////////////////////////////////// Determinant function ////////////////////////////////////
155 double algebra::determinant(const Matrix& matrix)
156 {
157     if( matrix.empty() )
158     {
159         return 1;
160     }
161 
162     if( matrix.size() != matrix[0].size())
163     {
164         throw std::logic_error( "Caution: non-square matrices have no determinant" );
165     }
166 
167     Matrix temp {matrix};
168     double det{};
169     if( temp.size() == 1)
170     {
171         return temp[0][0];  // final determinant which is a number
172     }
173 
174     for(size_t i{} ; i < matrix.size() ; i++ )
175     {
176         temp = algebra::minor(matrix,i,0);      // calculate minor
177         det += pow(-1,i) * matrix[i][0] * algebra::determinant(temp);    //calculate adjoint matrix
178     }
179 return det;
180 }
181 
```

7-1-تابع Inverse

مانند قبل همه ملاحظات گفته شده برای خطاهای احتمالی را بررسی کردیم.

سپس دترمینان ماتریس اصلی را در متغییرmain_det مقداردهی اولیه کردیم.

سپس دو ماتریس با مقادیر اولیه تماما صفر به کمک تابع zeros که قبلا ساخته ایم می‌سازیم. ابعاد ماتریس اول یکی کمتر از ماتریس اصلی برای ماینور و دومی هم ابعاد ماتریس اصلی می‌باشد.

سپس با 2 حلقه تو در تو ابتدا ماینور را محاسبه می‌کنیم و سپس با محاسبه ی دترمینان ماینور ها ماتریس الحاقی را می‌سازیم و سپس منفی یک به توان جمع اندیس درایه ها را اعمال می‌کنیم.

در نهایت ماتریس را ترنسپوز کرده و بر دترمینان ماتریس اصلی تقسیم می‌کنیم.

```cpp
////////////////////////////////////////// Inverse function ////////////////////////////////////
Matrix algebra::inverse(const Matrix& matrix)
{
    if( matrix.empty() )
    {
        return Matrix{};
    }
    if( matrix.size() != matrix[0].size())
    {
        throw std::logic_error( "Caution: non-square matrices have no inverse" );
    }
    double main_det {algebra::determinant(matrix)};
    if( main_det == 0 )
    {
        throw std::logic_error( "Caution: singular matrices have no inverse" );
    }
    Matrix temp1 { algebra::zeros(matrix.size() -1,matrix[0].size() - 1) };
    Matrix result { algebra::zeros(matrix.size() ,matrix[0].size() ) };

    for (size_t i{} ; i < matrix.size() ; i++)
        for(size_t j{} ; j < matrix[0].size() ; j++)
            {
                temp1 = algebra::minor(matrix,i,j);
                //we are creating adjoint matrix with calculating determinant of minors
                double det = algebra::determinant(temp1);
                result[i][j] = pow(-1 , i+j) * det ;
            }
    result = algebra::transpose(result);
    return algebra::multiply(result , (1/main_det)) ;
}
```

8-1-تابع concatenate

با توجه به ورودی axis که صفر است یا یک به 2 قسمت تقسیم می‌کنیم.

در هر حالت خطا های احتمالی را بررسی کردیم و هم چنین با توجه به نوع axis با دستور push_back سطر به سطر ردیف های ماتریس دومی را به ماتریس اولی اضافه می‌کنیم.

```cpp
////////////////////////////////////////// concatenate function ////////////////////////////////////
Matrix algebra::concatenate(const Matrix& matrix1, const Matrix& matrix2 , int axis)
{
    Matrix concatenated_matrix_row {matrix1};
    Matrix concatenated_matrix_column {algebra::transpose(matrix1)};
    if( axis == 0 )          //for axis = 0
    {
        if( (matrix1[0].size() != matrix2[0].size()) )
        {
        throw std::logic_error( "Caution: matrices with wrong dimensions cannot be concatenated" );
        }
        // we add rows of matrix2 at the end of matrix1
        for(size_t i{} ; i < matrix2.size() ; i++)
            concatenated_matrix_row.push_back(matrix2[i]);

        return concatenated_matrix_row ;
    }

    else if( axis == 1 )
    {
        if( (matrix1.size() != matrix2.size()) )
        {
        throw std::logic_error( "Caution: matrices with wrong dimensions cannot be concatenated" );
        }

        for(size_t i{} ; i < matrix2[0].size() ; i++)
            // first i transpose the whole matrix and then i choose rows 1 by 1
            concatenated_matrix_column.push_back(algebra::transpose(matrix2)[i]);
        return algebra::transpose(concatenated_matrix_column);
    }
    else
        return Matrix {};
}
```

9-1-تابع ero_swap و ero_multiply

بررسی ملاحظات خطا های احتمالی را انجام دادیم. سپس با دستور swap که در کتابخانه vector است این عمل را انجا می‌دهیم. 😊

```
244 ////////////////////////////////////////////// ero_swap function ////////////////////////
245 Matrix algebra::ero_swap(const Matrix& matrix, size_t r1, size_t r2)
246 {
247     if( matrix.empty() )
248     {
249         return Matrix{};
250     }
251     if( r1 >= matrix.size() || r2 >= matrix.size() || r1 < 0  || r2 < 0)
252     {
253         throw std::logic_error( "Caution: r1 or r2 inputs are out of range" );
254     }
255     Matrix swaped_matrix { matrix } ;
256     swaped_matrix[r2].swap(swaped_matrix[r1]);
257     return swaped_matrix ;
258 }
259 ////////////////////////////////////////////// ero_multiply function ////////////////////
260 Matrix algebra::ero_multiply(const Matrix& matrix, size_t r, double c)
261 {
262     if( matrix.empty() )
263     {
264         return Matrix{};
265     }
266     Matrix multipied_matrix { matrix } ;
267     Matrix temp {algebra::multiply(matrix , c)} ;
268
269     for(size_t i{} ; i < matrix.size() ; i++)
270         if(i == r)
271         {
272             multipied_matrix[r] = temp[r];
273         }
274 return multipied_matrix;
275 }
```

10-1-تابع ero_sum

در این تابع همه ملاحظات قبلی انجام شده و بررسی می‌شود. سپس با استفاده از توابع multiply و Ero_swap که در قبل تعریف و مشخص کردیم ، یک سطر از ماتریس ورودی را در یک اسکالر که از ورودی دریافت کردی ضرب و سپس با یک سطر دیگر جمع می‌کنیم و در همان سطر دومی میریزیم.

```
276 ////////////////////////////////////////// ero_sum function ////////////////////////////////////
277 Matrix algebra::ero_sum(const Matrix& matrix, size_t r1, double c, size_t r2)
278 {
279
280     if( matrix.empty() )
281     {
282         return Matrix{};
283     }
284
285     if(r1 >= matrix.size()  || r2 >= matrix.size())
286     {
287         throw std::logic_error("Caution: r1 or r2 inputs are out of range");
288     }
289
290
291     Matrix sumed_matrix { matrix };
292     Matrix temp {algebra::multiply(matrix , c)};
293     temp = algebra::ero_swap(temp , r1, r2);
294     temp = algebra::sum(matrix , temp);
295     sumed_matrix[r2] = temp[r2];
296     return sumed_matrix;
297
298 }
```

11-1-تابع upper_triangular

در این تابع ابتدا ملاحظات قبلی را بررسی می‌کنیم .

سپس چک می‌کنیم که اگر درایه صفری روی قطر اصلی باشد آن را با استفاده از تابع ero_swap سطر ها را جابهجا می‌کنیم تا به حالت عادی بازگردیم . سپس با 2 حلقه تو در تو و شروع حلقه داخلی از درایه ای که در رو روی قطر قرار دارد ، با استفاده از تابع ero_sum ضریبی از سطر اصلی را با سطر های پایین اش جمع می‌کنیم تا ماتریس را به صورت مثلثی در بیاوریم.

```
263 ////////////////////////////////////////// upper_triangular function ////////////////////////////////////
264 Matrix algebra::upper_triangular(const Matrix& matrix)
265 {
266     Matrix temp { matrix };
267     if (matrix.empty()) {
268         return Matrix {};
269     }
270     else if (matrix.size() != matrix[0].size())
271     {
272         throw std::logic_error("Caution: non-square matrices have no upper triangular form");
273     }
274      else
275      {
276           for(size_t i{} ; i < matrix.size() ; i++)
277               if(matrix[i][i] == 0)
278               {
279                   temp = algebra::ero_swap(temp , i , i+1);
280               }
281           for (size_t i {}; i < matrix.size(); i++)
282                       for (size_t j { i + 1 }; j < matrix.size(); j++)
283                           temp = algebra::ero_sum(temp, i, (-temp[j][i] / temp[i][i]), j);
284      }
285     return temp;
286 }
```