

به نام خدا

گزارش کار تمرین اول AP

محمد مهدی مالوردی

9723079

استاد: امیر جهانشاهی

گروه تدریس یاری: کیان بهزاد و امیر بیات



دانشگاه صنعتی امیر کبیر
(پلی تکنیک تهران)

شرح گزارش

ابتدا داخل فایل هدر ، تمامی توابع را داخل algebra namespace تعریف می کنیم. بعضی از فراخوانی کتابخانه ها را نیز در این فایل انجام می دهیم.

```
hw1.cpp  hw1.h  x
include > C hw1.h
1  #ifndef AP_HW1_H
2  #define AP_HW1_H
3  #include <vector>
4  #include <iomanip>
5
6  using Matrix = std::vector<std::vector<double>>;
7
8  namespace algebra
9  {
10
11     Matrix zeros(size_t n, size_t m);
12     Matrix ones(size_t n, size_t m);
13     Matrix random(size_t n, size_t m, double min, double max);
14     void show(const Matrix &matrix);
15     Matrix multiply(const Matrix &matrix, double c);
16     Matrix multiply(const Matrix &matrix1, const Matrix &matrix2);
17     Matrix sum(const Matrix &matrix, double c);
18     Matrix sum(const Matrix &matrix1, const Matrix &matrix2);
19     Matrix transpose(const Matrix &matrix);
20     Matrix minor(const Matrix &matrix, size_t n, size_t m);
21     double determinant(const Matrix &matrix);
22     Matrix inverse(const Matrix &matrix);
23     Matrix concatenate(const Matrix &matrix1, const Matrix &matrix2, int axis);
24     Matrix ero_swap(const Matrix &matrix, size_t r1, size_t r2);
25     Matrix ero_multiply(const Matrix &matrix, size_t r, double c);
26     Matrix ero_sum(const Matrix &matrix, size_t r1, double c, size_t r2);
27     Matrix upper_triangular(const Matrix &matrix);
28
29 }
30
31 #endif // AP_HW1_H
32
```

hw1.h

حال تعریف توابع را تک به تک در زیر از فایل hw1.cpp استخراج می کنیم:

توابع zeros و ones

این توابع را به وسیله متد push_back، حلقه های تو در تو و ماتریس کمکی op به صورت زیر پیاده سازی می کنیم:

```

G hw1.cpp X C hw1.h
src > G hw1.cpp
1  #include <iostream>
2  #include <random>
3  #include "hw1.h"
4  #include <iomanip>
5  #include <cmath>
6
7  Matrix algebra::zeros(size_t n, size_t m)
8  {
9
10     Matrix output{};
11
12     std::vector<double> op{};
13
14     for (size_t i{}; i < n; i++)
15     {
16
17         for (size_t j{}; j < m; j++)
18         {
19
20             op.push_back(0);
21         }
22
23         output.push_back(op);
24         op.clear();
25     }
26
27     return output;
28 }
29
30 Matrix algebra::ones(size_t n, size_t m)
31 {
32
33     Matrix output{};
34
35     std::vector<double> op{};
36
37     for (size_t i{}; i < n; i++)
38     {
39
40         for (size_t j{}; j < m; j++)
41         {
42
43             op.push_back(1);
44         }
45
46         output.push_back(op);
47         op.clear();
48     }
49
50     return output;
51 }

```

ones, zeros

توابع random و show

در این توابع نیز مانند توابع قبلی عمل کردیم. برای تولید عدد رندوم از روش گفته شده در صورت سوال (کتابخانه رندوم C++11) استفاده کردیم. برای show هم از setw و setprecision استفاده کردیم.

```
std::random_device rd;
std::mt19937 mt(rd());
std::uniform_real_distribution<double> dist(min, max);
```

برای تولید عدد رندوم

```
53 Matrix algebra::random(size_t n, size_t m, double min, double max)
54 {
55     Matrix output{};
56     std::vector<double> op{};
57
58     if (min > max)
59     {
60         throw std::logic_error("Caution: min cannot be greater than max");
61     }
62
63     std::random_device rd;
64     std::mt19937 mt(rd());
65     std::uniform_real_distribution<double> dist(min, max);
66
67     for (size_t i{}; i < n; i++)
68     {
69         for (size_t j{}; j < m; j++)
70         {
71             op.push_back(dist(mt));
72         }
73         output.push_back(op);
74         op.clear();
75     }
76     return output;
77 }
78
79 void algebra::show(const Matrix &matrix)
80 {
81     for (size_t i{}; i < matrix.size(); i++)
82     {
83         for (size_t j{}; j < matrix[0].size(); j++)
84         {
85             std::cout << std::setw(3) << std::setprecision(3) << matrix[i][j] << " ";
86         }
87         std::cout << std::endl;
88     }
89 }
```

random, show

توابع multiply

```
98 Matrix algebra::multiply(const Matrix &matrix, double c)
99 {
100     std::vector<double> op{};
101     Matrix output{};
102     if (matrix.size() == 0)
103     {
104         return output;
105     }
106
107     for (size_t i{}; i < matrix.size(); i++)
108     {
109         for (size_t j{}; j < matrix[0].size(); j++)
110         {
111             op.push_back(c * matrix[i][j]);
112         }
113         output.push_back(op);
114         op.clear();
115     }
116
117     return output;
118 }
119
120
121 }
```

ضرب عدد ثابت در تک تک درایه های ماتریس

در این جا خیلی راحت و به وسیله دو حلقه تو در تو، عدد ثابت C را در تک تک درایه های ماتریس ضرب می کنیم. باید دقت کنیم که اگر سائز ماتریس ما صفر بود، همان ماتریس تهی رو باید برگردانیم.

```

122
123 Matrix algebra::multiply(const Matrix &matrix1, const Matrix &matrix2)
124 {
125     std::vector<double> op{};
126     Matrix output{};
127
128     if (matrix1.size() == 0 && matrix2.size() == 0)
129     {
130         return output;
131     }
132     size_t n(matrix1.size());
133     size_t m(matrix1[0].size());
134     size_t q(matrix2.size());
135     size_t p(matrix2[0].size());
136
137     double sum{};
138
139     if (n == 0 || m == 0 || q == 0 || p == 0)
140     {
141
142         throw std::logic_error("Caution: multiplication of 2 empty matrix");
143     }
144
145     if (m != q)
146     {
147
148         throw std::logic_error("Caution: matrices with wrong dimensions cannot be multiplied");
149     }
150     else
151     {
152
153         for (size_t i{}; i < n; i++)
154         {
155             for (size_t j{}; j < p; j++)
156             {
157                 for (size_t k{}; k < m; k++)
158                 {
159                     sum += (matrix1[i][k] * matrix2[k][j]);
160                 }
161                 op.push_back(sum);
162                 sum = 0;
163             }
164
165             output.push_back(op);
166             op.clear();
167         }
168     }
169
170     return output;
171 }
172

```

ضرب دو ماتریس $n*m$

در اینجا نیز علاوه بر بررسی سایز ماتریس که در ضرب عدد ثابت هم انجام دادیم، به وسیله سه حلقه تو در تو که هر کدام از داخلی ترین به بیرونی ترین به ترتیب تعداد جمع های مورد نیاز، تعداد ستون ها و تعداد سطر ها را کنترل می کند. باید حواسمان باشد که در هر دور `sum` و `op` را `clear` و صفر کنیم تا هربار دچار مشکل نشویم.

توابع sum

```
173 Matrix algebra::sum(const Matrix &matrix, double c)
174 {
175     std::vector<double> op{};
176     Matrix output{};
177
178     if (matrix.size() == 0)
179     {
180         return output;
181     }
182
183     for (size_t i{}; i < matrix.size(); i++)
184     {
185         for (size_t j{}; j < matrix[0].size(); j++)
186         {
187             op.push_back(c + matrix[i][j]);
188         }
189
190         output.push_back(op);
191         op.clear();
192     }
193
194     return output;
195 }
196
197
198
199
```

جمع تک به تک درایه ها با عددی ثابت

مشابه ضرب می باشد، منتهی با فرق این که درایه ها را با عدد ثابت C جمع می کنیم.

```

200 Matrix algebra::sum(const Matrix &matrix1, const Matrix &matrix2)
201 {
202
203     std::vector<double> op{};
204     Matrix output{};
205     if (matrix1.empty() && matrix2.empty())
206     {
207         return output;
208     }
209     size_t height_of_matrix1{matrix1.size()};
210
211     size_t height_of_matrix2{matrix2.size()};
212
213     if (height_of_matrix1 != height_of_matrix2)
214     {
215
216         throw std::logic_error("Caution: matrices with wrong dimensions cannot be summed");
217     }
218
219     size_t length_of_matrix1{matrix1[0].size()};
220     size_t length_of_matrix2{matrix2[0].size()};
221     size_t n{height_of_matrix1}, m{length_of_matrix1};
222
223     if (length_of_matrix1 != length_of_matrix2)
224     {
225         throw std::logic_error("Caution: matrices with wrong dimensions cannot be summed");
226     }
227
228     for (size_t i{}; i < n; i++)
229     {
230         for (size_t j{}; j < m; j++)
231         {
232
233             op.push_back(matrix1[i][j] + matrix2[i][j]);
234         }
235
236         output.push_back(op);
237         op.clear();
238     }
239
240     return output;
241 }
242

```

جمع دو ماتریس با یکدیگر

به وسیله دو حلقه تو در تو تک به تک درایه ها را در هر موقعیت با یکدیگر جمع کرده و ماتریس خروجی را تشکی می دهیم.

توابع transpose و minor

```
242
243 Matrix algebra::transpose(const Matrix &matrix)
244 {
245     std::vector<double> op{};
246     Matrix output{};
247
248     if (matrix.empty())
249     {
250         return output;
251     }
252
253     for (size_t i{}; i < matrix[0].size(); i++)
254     {
255         for (size_t j{}; j < matrix.size(); j++)
256         {
257             op.push_back(matrix[j][i]);
258         }
259
260         output.push_back(op);
261         op.clear();
262     }
263
264     return output;
265 }
266
267
268
269 Matrix algebra::minor(const Matrix &matrix, size_t n, size_t m)
270 {
271     std::vector<double> op{};
272     Matrix output{};
273
274     for (size_t i{}; i < matrix.size(); i++)
275     {
276         if (i != n)
277         {
278             for (size_t j{}; j < matrix[0].size(); j++)
279             {
280                 if (j != m)
281                 {
282                     op.push_back(matrix[i][j]);
283                 }
284             }
285
286             output.push_back(op);
287             op.clear();
288         }
289     }
290
291     return output;
292 }
293
294
295
296 }
```

ماتریس transpose و minor

در ماتریس ترانهاده، به کمک دو حلقه تو در تو جای سطر و ستون را عوض می کنیم.

در تابع محاسبه ماینور ها نیز به وسیله ماتریس های کمکی `op` و `output`، درایه هایی که اندیس های سطر و ستون آن ها با اندیس های اعلان شده متفاوت است را در قالب ماتریس خروجی در می آوریم.

تابع determinant

```
297
298 double algebra::determinant(const Matrix &matrix)
299 {
300     if (matrix.empty())
301     {
302         return 1.0;
303     }
304
305     double sum{};
306
307     if (matrix.size() != matrix[0].size())
308     {
309
310         throw std::logic_error("Caution: non-square matrices have no determinant");
311     }
312
313     if (matrix.size() == 2 && matrix[0].size() == 2)
314     {
315
316         sum = matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0];
317     }
318     else if (matrix.size() == 1 && matrix[0].size() == 1)
319     {
320         sum = matrix[0][0];
321     }
322     else
323     {
324         for (size_t j{}; j < matrix[0].size(); j++)
325         {
326
327             sum += matrix[0][j] * pow(-1, 0 + j) * determinant(minor(matrix, 0, j));
328         }
329     }
330     return sum;
331 }
332
```

تابع محاسبه دترمینان

ابتدا باید دقت کنیم که فقط ماتریس های مربعی دترمینان دارند. همچنین دترمینان ماتریس های 2×2 و 1×1 را جدا گانه محاسبه می نماییم. دترمینان ماتریس `empty` را نیز برابر یک در نظر می گیریم. حال دترمینان سایر ابعاد را به کمک تابع `minor`، تابعی بازگشتی از `determinant` و روش یاد گرفته شده در جبر خطی محاسبه می نماییم.

تابع inverse

```
332
333 Matrix algebra::inverse(const Matrix &matrix)
334 {
335     std::vector<double> op{};
336     Matrix adj{}, adjT{}, inv{};
337     double det{determinant(matrix)};
338     if (matrix.empty())
339     {
340         return adj;
341     }
342
343     if (matrix.size() != matrix[0].size())
344     {
345         throw std::logic_error("Caution: non-square matrices have no inverse");
346     }
347
348     if (det == 0)
349     {
350         throw std::logic_error("Caution: singular matrices have no inverse");
351     }
352
353     for (size_t i{}; i < matrix.size(); i++)
354     {
355         for (size_t j{}; j < matrix[0].size(); j++)
356         {
357             op.push_back(pow(-1, i + j) * determinant(minor(matrix, i, j)));
358         }
359         adj.push_back(op);
360         op.clear();
361     }
362
363     adjT = transpose(adj);
364     inv = multiply(adjT, 1 / det);
365
366     return inv;
367 }
368
369
370
371
372 }
```

تابع محاسبه معکوس ماتریس $n \times m$

به کمک ماینور ها و دترمینان، معکوس ماتریس را محاسبه می کنیم. باید دقت کنیم که ماتریس های غیر مربعی و سینگولار (دترمینان برابر با صفر)، معکوس ندارند.

تابع concatenate

```
373 Matrix algebra::concatenate(const Matrix &matrix1, const Matrix &matrix2, int axis = 0)
374 {
375
376     std::vector<double> op{};
377     Matrix output{};
378
379     if (axis == 0)
380     {
381         if (matrix1[0].size() != matrix2[0].size())
382         {
383             throw std::logic_error("Caution: matrices with wrong dimensions cannot be concatenated");
384         }
385         for (size_t i{}; i < matrix1.size() + matrix2.size(); i++)
386         {
387             for (size_t j{}; j < matrix1[0].size(); j++)
388             {
389                 if (i < matrix1.size())
390                 {
391                     op.push_back(matrix1[i][j]);
392                 }
393                 else if (i >= matrix1.size() && i < matrix1.size() + matrix2.size())
394                 {
395                     op.push_back(matrix2[i - matrix1.size()][j]);
396                 }
397             }
398             output.push_back(op);
399             op.clear();
400         }
401     }
402     else if (axis == 1)
403     {
404         if (matrix1.size() != matrix2.size())
405         {
406             throw std::logic_error("Caution: matrices with wrong dimensions cannot be concatenated");
407         }
408         for (size_t i{}; i < matrix1.size(); i++)
409         {
410             for (size_t j{}; j < matrix1[0].size() + matrix2[0].size(); j++)
411             {
412                 if (j < matrix1[0].size())
413                 {
414                     op.push_back(matrix1[i][j]);
415                 }
416                 else if (j >= matrix1[0].size() && j < matrix1[0].size() + matrix2[0].size())
417                 {
418                     op.push_back(matrix2[i][j - matrix1[0].size()]);
419                 }
420             }
421             output.push_back(op);
422             op.clear();
423         }
424     }
425     return output;
426 }
427 }
```

تابع بهم چسباندن ماتریس ها به صورت عمودی یا افقی

برای این کار نیز متناسب با سایز ماتریس ها و این که به صورت عمودی یا افقی قرار است آن ها را داخل یک ماتریس واحد قرار دهیم، خروجی را در output قرار دادیم.

توابع ero_swap, ero_multiply, ero_sum

```
439 Matrix algebra::ero_swap(const Matrix &matrix, size_t r1, size_t r2)
440 {
441     std::vector<double> op{};
442     Matrix output(matrix);
443
444     if (r1 >= output.size() || r2 >= output.size())
445     {
446         throw std::logic_error("Caution: r1 or r2 inputs are out of range");
447     }
448
449     for (size_t j{}; j < output[0].size(); j++)
450     {
451         op.push_back(output[r1][j]);
452     }
453     output[r1].clear();
454     for (size_t j{}; j < output[0].size(); j++)
455     {
456         output[r1].push_back(output[r2][j]);
457     }
458     output[r2].clear();
459     for (size_t j{}; j < output[0].size(); j++)
460     {
461         output[r2].push_back(op[j]);
462     }
463
464     return output;
465 }
466
467
468
469
470
```

تابع ero_swap

در این تابع، پس از این که مطمئن شدیم که $r1$, $r2$ در بازه سطر های موجود هستند، به کمک ماتریس های کمکی op و $output$ و یک حلقه for ، این دو سطر را با یکدیگر جابجا می کنیم.

```

471 Matrix algebra::ero_multiply(const Matrix &matrix, size_t r, double c)
472 {
473     std::vector<double> op{};
474     Matrix output{};
475
476     if (r >= matrix.size())
477     {
478         throw std::logic_error("Caution: r1 or r2 inputs are out of range");
479     }
480
481     for (size_t i{}; i < matrix.size(); i++)
482     {
483         if (i == r)
484         {
485             for (size_t j{}; j < matrix[0].size(); j++)
486             {
487                 op.push_back(matrix[i][j] * c);
488             }
489
490             output.push_back(op);
491             op.clear();
492         }
493         else
494         {
495             for (size_t j{}; j < matrix[0].size(); j++)
496             {
497                 op.push_back(matrix[i][j]);
498             }
499
500             output.push_back(op);
501             op.clear();
502         }
503     }
504
505     return output;
506 }
507
508
509
510
511

```

تابع ero_multiply

در این تابع، پس از رسیدن به سطر مورد نظر r ، عدد ثابت c را در آن سطر ضرب کرده و جایگزین آن سطر می‌کنیم. بقیه سطرها را همانگونه که هستند باقی می‌گذاریم. در نهایت ماتریس نتیجه را در `output` می‌ریزیم.

```

Matrix algebra::ero_sum(const Matrix &matrix, size_t r1, double c, size_t r2)
{
    Matrix output(matrix);
    std::vector<double> op;

    if (r1 >= output.size() || r2 >= output.size())
    {
        throw std::logic_error("Caution: r1 or r2 inputs are out of range");
    }

    for (size_t j{0}; j < output[0].size(); j++)
    {
        op.push_back(c * output[r1][j]);
    }

    for (size_t j{0}; j < output[0].size(); j++)
    {
        output[r2][j] += op[j];
    }

    return output;
}

```

تابع `ero_sum`

در این تابع، ابتدا از ماتریس کمکی `output` استفاده کرده و مقدار اولیه آن را برابر با ماتریس ورودی قرار می دهیم. همچنین عدد ثابت `c` را در سطر `r1` ام ماتریس کمکی ضرب می کنیم. در نهایت تغییرات و جابجایی مورد نظر را روی `output` اعمال کرده و آن را بر میگردانیم.

ماتریس upper_tiangular

```
539 Matrix algebra::upper_triangular(const Matrix &matrix)
540 {
541     Matrix output(matrix);
542     if (output.empty())
543     {
544         return output;
545     }
546     if (output.size() != output[0].size())
547     {
548         throw std::logic_error("Caution: non-square matrices have no upper triangular form");
549     }
550 }
551
552 for (size_t i{}; i < output.size(); i++)
553 {
554     for (size_t j{i}; j < output.size() - 1; j++)
555     {
556         {
557             output = ero_sum(output, i, -1 * output[j + 1][i] / output[i][i], j + 1);
558         }
559     }
560 }
561
562 return output;
563 }
```

تابع تبدیل ماتریس ورودی به ماتریس بالامثلثی

ابتدا باید توجه کنیم که فقط ماتریس های مربعی را می توان بالا مثلثی کرد. در این تابع نیز به کمک ماتریس کمکی output و تابع ero_sum عملیات بالا مثلثی کردن ماتریس ورودی را درون دو حلقه تو در تو و به وسیله فرمول به دست آمده روی کاغذ که روبری output داخل حلقه درونی نوشته شده است، انجام دادیم.

در نهایت 23 تست از 24 تست پاس شدند که تست 24 امتیازی بود.