



Section 1: 作业内容概述

Section 2: Structed Binding

```
1 #include <iostream>
2 #include <map>
3 #include <string>
4 #include <functional>
5 template<typename Key, typename Value, typename F>
6
7
8 void update(std::map<Key, Value> &m, F foo)
9 {
10     for(auto &[x, y] :m) y = foo(x);
11
12 }
13 int main(){
14     std::map<std::string, long long int> m{
15         { "a", 1 },
16         { "b", 2 },
17         { "c", 3 }
18     };
19     update(m, [](auto x){ return x.size(); });
20     for (auto &&[key, value]: m)
21         std::cout << key << ":" << value << std::endl;
22 }
```

Section 3: References

```
1 #include <iostream>
2 using namespace std;
3 void swap(int &a, int &b)
4 {
5     int temp = a;
6     a = b;
```

```
7     b = temp;
8 }
9 int main()
10 {
11     int a = 10, b = 100;
12     swap(a, b);
13     cout<<a<<" "<<b<<endl;
14     return 0;
15 }
```

Section 4: Streams

```
1 #include <iostream>
2 #include <string>
3 #include <fstream>
4 using namespace std;
5 class Student
6 {
7 public:
8     Student(string &name, int &score)
9     {
10         this->name = name;
11         this->score = score;
12     }
13     void setName(string &name)
14     {
15         this->name = name;
16     }
17     void setScore(int score)
18     {
19         this->score = score;
20     }
21     int getScore()
22     {
23         return score;
24     }
25     string getName()
26     {
27         return name;
28     }
29 private:
30     string name;
31     int score;
32 };
```

```

33  int main()
34  {
35      string name;
36      int score;
37      Student stu(name, score);
38      ofstream out("stud.dat");
39      if(out.is_open())
40      {
41          while(cin >> name >> score)
42          {
43              stu.setName(name);
44              stu.setScore(score);
45              out << stu.getName() << " " << stu.getScore() << endl;
46          }
47      }
48      else
49      {
50          cout << "open file failed" << endl;
51      }
52
53      ifstream in("stud.dat");
54      if(in.is_open())
55      {
56          string line;
57          while(getline(in, line))
58          {
59              cout << line << endl;
60          }
61          in.close();
62      }
63      else
64      {
65          cout << "open file failed" << endl;
66      }
67      return 0;
68  }

```

Section 5: STL(Containers)

```

1  #include <iostream>
2  #include <vector>
3  using namespace std;
4  void Traversal(vector<int> v)
5  {

```

```

6     for(vector<int>::iterator it = v.begin(); it != v.end(); it++)
7     {
8         cout<<*it<<' ';
9     }
10    cout<<endl;
11 }
12 void rTraversal(vector<int> v)
13 {
14     for(vector<int>::reverse_iterator it = v.rbegin(); it != v.rend(); it++)
15     {
16         cout<<*it<<' ';
17     }
18    cout<<endl;
19 }
20 void test()
21 {
22     vector<int> v;
23     int num;
24     for(int i = 0; i < 5; i++)
25     {
26         cin>>num;
27         v.push_back(num);
28     }
29     Traversal(v);
30     rTraversal(v);
31 }
32 int main()
33 {
34     test();
35     return 0;
36 }

```

Section 6: Linear Algebra Library

Code

linearalgebra.h

```

1 #include <iostream>
2 #include <vector>
3 #include <cmath>
4 #include <algorithm>
5 #include <random>
6 #include <iomanip>
7

```

```

8  using Matrix = std::vector<std::vector<double>>>;
9
10 Matrix construct_matrix(size_t m, size_t n);
11
12 Matrix zeros(size_t m, size_t n);
13
14 Matrix ones(size_t m, size_t n);
15
16 Matrix random(size_t m, size_t n, int min, int max);
17
18 Matrix show(const Matrix &A);
19
20 Matrix multiply(const Matrix &A, double c);
21
22 Matrix multiply(const Matrix &A, const Matrix &B);
23
24 Matrix sum(const Matrix &A, double c);
25
26 Matrix sum(const Matrix &A, const Matrix &B);
27
28 Matrix transpose(const Matrix &A);
29
30 Matrix minor(const Matrix &A, size_t x, size_t y);
31
32 double determinant(const Matrix &A);
33
34 Matrix inverse(const Matrix &A);
35
36 Matrix unit_matrix(size_t n);
37
38 Matrix ero_swap(const Matrix &A, size_t x, size_t y);
39
40 Matrix ero_multiply(Matrix A, size_t x, double c);
41
42 Matrix ero_sum(Matrix A, size_t x, double c, size_t y);
43
44 Matrix concatenate(const Matrix &A, const Matrix &B, int axis);
45
46 Matrix upper_triangular(const Matrix &matrix);

```

linearalgebra.cpp

```

1  Matrix construct_matrix(size_t m, size_t n) {
2      if (m == 0 || n == 0) {
3          std::cout << "Matrix is empty" << std::endl;
4          return Matrix();
5      }

```

```

6     Matrix A(m, std::vector<double>(n, 0.0));
7     for (size_t i = 0; i < m; i++) {
8         for (size_t j = 0; j < n; j++) {
9             std::cin >> A[i][j];
10        }
11    }
12    return A;
13 }
14
15 Matrix zeros(size_t m, size_t n) {
16     Matrix A(m, std::vector<double>(n, 0.0));
17     return A;
18 }
19
20 Matrix ones(size_t m, size_t n) {
21     Matrix A(m, std::vector<double>(n, 1.0));
22     return A;
23 }
24
25 Matrix random(size_t m, size_t n, int min, int max) {
26     if (min > max) {
27         throw (std::logic_error(""));
28         return {};
29     }
30     std::random_device rd;
31     std::mt19937 gen(rd());
32     std::uniform_real_distribution<> dis(min, max);
33     Matrix A(m, std::vector<double>(n, 0.0));
34     for (size_t i = 0; i < m; i++) {
35         for (size_t j = 0; j < n; j++) {
36             A[i][j] = dis(gen);
37         }
38     }
39     return A;
40 }
41
42 Matrix show(const Matrix &A) {
43     if (A.empty()) {
44         std::cout << "Matrix is empty" << std::endl;
45         return A;
46     }
47     for (size_t i = 0; i < A.size(); i++) {
48         for (size_t j = 0; j < A[i].size(); j++) {
49             printf("%-8.3f", A[i][j]);
50         }
51         std::cout << std::endl;

```

```

52     }
53     return A;
54 }
55
56 Matrix multiply(const Matrix &A, double c) {
57     Matrix B = zeros(A.size(), A[0].size());
58     for (size_t i = 0; i < A.size(); i++) {
59         for (size_t j = 0; j < A[i].size(); j++) {
60             B[i][j] = A[i][j] * c;
61         }
62     }
63     return B;
64 }
65
66 Matrix multiply(const Matrix &A, const Matrix &B) {
67     if (A.empty() || B.empty()) {
68         return {};
69     }
70     if (A[0].size() != B.size()) {
71         throw std::logic_error("");
72     }
73     Matrix C = zeros(A.size(), B[0].size());
74     for (size_t i = 0; i < A.size(); i++) {
75         for (size_t j = 0; j < B[0].size(); j++) {
76             for (size_t k = 0; k < A[0].size(); k++) {
77                 C[i][j] += A[i][k] * B[k][j];
78             }
79         }
80     }
81     return C;
82 }
83
84 Matrix sum(const Matrix &A, double c) {
85     if (A.empty()) {
86         return {};
87     }
88     Matrix B = zeros(A.size(), A[0].size());
89     for (size_t i = 0; i < A.size(); i++) {
90         for (size_t j = 0; j < A[i].size(); j++) {
91             B[i][j] = A[i][j] + c;
92         }
93     }
94     return B;
95 }
96
97 Matrix sum(const Matrix &A, const Matrix &B) {

```

```

98     if (A.empty() && B.empty()) {
99         return {};
100     }
101     if ((A.empty() && !B.empty()) || (!A.empty() && B.empty())) {
102         throw std::logic_error("");
103     }
104     Matrix C = zeros(A.size(), A[0].size());
105     for (size_t i = 0; i < A.size(); i++) {
106         for (size_t j = 0; j < A[i].size(); j++) {
107             C[i][j] = A[i][j] + B[i][j];
108         }
109     }
110     return C;
111 }
112
113 Matrix transpose(const Matrix &A) {
114     if (A.empty()) {
115         return {};
116     }
117     Matrix B = zeros(A[0].size(), A.size());
118     for (auto i = 0; i < A[0].size(); i++) {
119         for (auto j = 0; j < A.size(); j++) {
120             B[i][j] = A[j][i];
121         }
122     }
123     return B;
124 }
125
126 Matrix minor(const Matrix &A, size_t x, size_t y) {
127     Matrix B = zeros(A.size(), A[0].size());
128     for (auto i = 0; i < A.size(); i++) {
129         for (auto j = 0; j < A[0].size(); j++) {
130             B[i][j] = A[i][j];
131         }
132     }
133     B.erase(B.begin() + x);
134     for (auto i = 0; i < B.size(); i++) {
135         B[i].erase(B[i].begin() + y);
136     }
137     return B;
138 }
139
140 double determinant(const Matrix &A) {
141     if (A.empty()) {
142         return 1;
143     }

```



```

144     if (A.size() != A[0].size()) {
145         throw std::logic_error("");
146     }
147     if (A.size() == 1) {
148         return A[0][0];
149     }
150     if (A.size() == 2) {
151         return A[0][0] * A[1][1] - A[0][1] * A[1][0];
152     }
153     if (A.size() > 2) {
154         double det = 0;
155         for (auto i = 0; i < A.size(); i++) {
156             det += A[0][i] * pow(-1, i) * determinant(minor(A, 0, i));
157         }
158         return det;
159     }
160 }
161
162 Matrix ero_swap(const Matrix &A, size_t x, size_t y) {
163     Matrix ret = A;
164     if (x >= A.size() || y >= A.size()) {
165         throw std::logic_error("");
166     }
167     for (auto i = 0; i < A[0].size(); i++) {
168         std::swap(ret[x][i], ret[y][i]);
169     }
170     return ret;
171 }
172
173 Matrix ero_multiply(Matrix A, size_t x, double c) {
174     if (x >= A.size()) {
175         throw std::logic_error("");
176     }
177     for (auto i = 0; i < A[0].size(); i++) {
178         A[x][i] *= c;
179     }
180     return A;
181 }
182
183 Matrix ero_sum(Matrix A, size_t x, double c, size_t y) {
184     for (auto i = 0; i < A[0].size(); i++) {
185         A[y][i] += c * A[x][i];
186     }
187     return A;
188 }
189

```

```

190 Matrix unit_matrix(size_t n) {
191     Matrix A = zeros(n, n);
192     for (auto i = 0; i < n; i++) {
193         A[i][i] = 1;
194     }
195     return A;
196 }
197
198 Matrix inverse(const Matrix &A) {
199     if (A.empty()) {
200         return {};
201     }
202     if (determinant(A) == 0 || A.size() != A[0].size()) {
203         throw std::logic_error("");
204     }
205     Matrix B = zeros(A.size(), A[0].size());
206     for (auto i = 0; i < A.size(); i++) {
207         for (auto j = 0; j < A[0].size(); j++) {
208             B[i][j] = A[i][j];
209         }
210     }
211     Matrix C = unit_matrix(A.size());
212     for (auto i = 0; i < A.size(); i++) {
213         if (B[i][i] == 0) {
214             for (auto j = i + 1; j < A.size(); j++) {
215                 if (B[j][i] != 0) {
216                     C = ero_swap(C, i, j);
217                     B = ero_swap(B, i, j);
218
219                     break;
220                 }
221             }
222         }
223         C = ero_multiply(C, i, 1 / B[i][i]);
224         B = ero_multiply(B, i, 1 / B[i][i]);
225
226         for (auto j = 0; j < A.size(); j++) {
227             if (j != i && B[j][i] != 0) {
228                 C = ero_sum(C, i, -B[j][i], j);
229                 B = ero_sum(B, i, -B[j][i], j);
230
231             }
232         }
233     }
234     return C;
235 }

```

```

236
237 Matrix concatenate(const Matrix &A, const Matrix &B, int axis) {
238     if (axis == 0) {
239         if (A[0].size() != B[0].size()) {
240             throw std::logic_error("");
241         }
242         Matrix C = zeros(A.size() + B.size(), A[0].size());
243         for (auto i = 0; i < A.size(); i++) {
244             for (auto j = 0; j < A[0].size(); j++) {
245                 C[i][j] = A[i][j];
246             }
247         }
248         for (auto i = 0; i < B.size(); i++) {
249             for (auto j = 0; j < B[0].size(); j++) {
250                 C[i + A.size()][j] = B[i][j];
251             }
252         }
253         return C;
254     }
255     if (axis == 1) {
256         if (A.size() != B.size()) {
257             throw std::logic_error("");
258         }
259         Matrix C = zeros(A.size(), A[0].size() + B[0].size());
260         for (auto i = 0; i < A.size(); i++) {
261             for (auto j = 0; j < A[0].size(); j++) {
262                 C[i][j] = A[i][j];
263             }
264         }
265         for (auto i = 0; i < B.size(); i++) {
266             for (auto j = 0; j < B[0].size(); j++) {
267                 C[i][j + A[0].size()] = B[i][j];
268             }
269         }
270         return C;
271     }
272 }
273
274 Matrix upper_triangular(const Matrix &matrix)
275 {
276     if (matrix.empty()) {
277         return {};
278     }
279     if (matrix.size() != matrix[0].size()) {
280         throw std::logic_error("");
281     }

```

```

282 Matrix A = zeros(matrix.size(), matrix[0].size());
283 for (auto i = 0; i < matrix.size(); i++) {
284     for (auto j = 0; j < matrix[0].size(); j++) {
285         A[i][j] = matrix[i][j];
286     }
287 }
288 for(auto i = 0; i < A.size(); i++)
289 {
290     for(auto j = 0; j < A.size(); j++)
291     {
292         if(A[i][i] == 0)
293         {
294             for(auto k = i + 1; k < A.size(); k++)
295             {
296                 if(A[k][i] != 0)
297                 {
298                     A = ero_swap(A, i, k);
299                     break;
300                 }
301             }
302         }
303         if(j > i)
304         {
305             A = ero_sum(A, i, -A[j][i] / A[i][i], j);
306         }
307     }
308 }
309 return A;
310 }

```

```
Activities Terminal Mar 28 15:13
wyb@ubuntu: ~/Documents/LinearAlgebra/build

wyb@ubuntu:~/Documents/LinearAlgebra/build$ ./main
RUNNING TESTS ...
----- Running 24 tests from 1 test suite.
----- Global test environment set-up.
----- 24 tests from LinearAlgebraTest
[ RUN      ] LinearAlgebraTest.ZEROS (0 ms)
[ RUN      ] LinearAlgebraTest.ONES (0 ms)
[ RUN      ] LinearAlgebraTest.RANDOM1
random matrix [-5, 7)
0.834 6.171 -3.879 -4.256
1.306 -4.586 2.902 2.293
-3.063 5.234 -1.362 6.229
3.946 4.852 -2.849 5.490
[ RUN      ] LinearAlgebraTest.RANDOM1 (0 ms)
[ RUN      ] LinearAlgebraTest.RANDOM2 (0 ms)
[ RUN      ] LinearAlgebraTest.MULTIPLY1 (0 ms)
[ RUN      ] LinearAlgebraTest.MULTIPLY2 (0 ms)
[ RUN      ] LinearAlgebraTest.MULTIPLY2 (0 ms)
[ RUN      ] LinearAlgebraTest.MULTIPLY3 (0 ms)
[ RUN      ] LinearAlgebraTest.MULTIPLY3 (0 ms)
[ RUN      ] LinearAlgebraTest.MULTIPLY4 (0 ms)
[ RUN      ] LinearAlgebraTest.SUM1 (0 ms)
[ RUN      ] LinearAlgebraTest.SUM2 (0 ms)
[ RUN      ] LinearAlgebraTest.TRANSPOSE (0 ms)
[ RUN      ] LinearAlgebraTest.MINOR1 (0 ms)
[ RUN      ] LinearAlgebraTest.MINOR2 (0 ms)
[ RUN      ] LinearAlgebraTest.DETERMINANT1 (0 ms)
[ RUN      ] LinearAlgebraTest.DETERMINANT2 (0 ms)
[ RUN      ] LinearAlgebraTest.INVERSE1 (0 ms)
[ RUN      ] LinearAlgebraTest.INVERSE2 (0 ms)
[ RUN      ] LinearAlgebraTest.CONCATENATE1 (0 ms)
[ RUN      ] LinearAlgebraTest.CONCATENATE2 (0 ms)
[ RUN      ] LinearAlgebraTest.ERO_SWAP (0 ms)
[ RUN      ] LinearAlgebraTest.ERO_MULTIPLY (0 ms)
[ RUN      ] LinearAlgebraTest.ERO_SUM (0 ms)
[ RUN      ] LinearAlgebraTest.UPPER_TRIANGULAR1 (0 ms)
[ RUN      ] LinearAlgebraTest.BONUS (0 ms)
[ RUN      ] LinearAlgebraTest.BONUS (0 ms)
----- 24 tests from LinearAlgebraTest (2 ms total)

----- Global test environment tear-down
----- 24 tests from 1 test suite ran. (2 ms total)
PASSED 24 tests.
<<<SUCCESS>>>
wyb@ubuntu:~/Documents/LinearAlgebra/build$
```

Passed all googletest samples