Student Name: 马越
Student ID: 2022141460097

C++ 面向对象程序设计

Assignment 3

2 Structured Binding

Listing 1: Structured_Binding.cpp

```cpp
#include <iostream>
#include <map>
#include <string>
#include <functional>

template<typename Key, typename Value, typename F>
void update(std::map<Key, Value> &m, F foo) {
  for (auto &&[key, value]: m) value = foo(key);
}

int main() {
  std::map<std::string, long long int> m{
    {"a", 1},
    {"b", 2},
    {"c", 3}
  };
  update(m, [](std::string key) {
    return std::hash<std::string>{}(key);
  });
  for (auto &&[key, value]: m)
  std::cout << key << ":" << value << std::endl;
}
```

3 References

Listing 2: References.cpp

```cpp
#include <iostream>
#include <gtest/gtest.h>
using namespace std;

void swap(int &A, int &B) {
```

```
  int temp = A;
  A = B;
  B = temp;
}

TEST(SwapTest, SWAP) {
  int a{0}, b{1};
  swap(a, b);
  EXPECT_EQ(a, 1);
  EXPECT_EQ(b, 0);
}

int main(int argc, char **argv) {
  ::testing::InitGoogleTest(&argc, argv);
  std::cout << "RUNNING TESTS ..." << std::endl;
  int ret{RUN_ALL_TESTS()};
  if (!ret)
  std::cout << "<<<SUCCESS>>>" << std::endl;
  else
  std::cout << "FAILED" << std::endl;
  return 0;
}
```

4 Streams

Listing 3: Streams.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include <sstream>
#include <gtest/gtest.h>

namespace stu {
  static constexpr char FILENAME[] = "stu.dat";
  void input() {
    std::ofstream fout(FILENAME, std::ios::binary);
    if (!fout.is_open()) {
      throw std::runtime_error("文件打开失败");
    }
    std::cerr << "请输入学生姓名和成绩:" << std::endl;
    std::cerr << "(自动测试,无需输入)" << std::endl;
    std::string name;
    double score;
    while (std::cin >> name >> score) {
```

```
      size_t size = name.size();
      fout.write(reinterpret_cast<const char *>(&size), sizeof(size_t));  /// 首先保存字符串长度
      fout.write(name.c_str(), size);
      fout.write(reinterpret_cast<const char *>(&score), sizeof(double));
    }
    fout.close();
  }
  void display() {
    std::ifstream fin(FILENAME, std::ios::binary);
    if (!fin.is_open()) {
      throw std::runtime_error("文件打开失败");
    }
    std::string name;
    double score;
    size_t size;
    while (fin.read(reinterpret_cast<char *>(&size), sizeof(size_t))) {
      name.resize(size);
      fin.read(&name[0], size);
      fin.read(reinterpret_cast<char *>(&score), sizeof(double));
      std::cout << name << " " << score << std::endl;
    }
    fin.close();
  }
}

TEST(StuTest, input) {
  using stu::input;
  using stu::display;
  std::stringstream ss("stu1 10\nAlice 20.5\nBob 0\n");
  std::stringstream out;
  std::streambuf* cout_buff = std::cout.rdbuf();
  std::cin.rdbuf(ss.rdbuf());
  std::cout.rdbuf(out.rdbuf());
  input();
  display();
  std::cout.rdbuf(cout_buff);
  EXPECT_EQ(out.str(), "stu1 10\nAlice 20.5\nBob 0\n");
}

int main(int argc, char **argv) {
  ::testing::InitGoogleTest(&argc, argv);
  std::cout << "RUNNING TESTS ..." << std::endl;
  int ret{RUN_ALL_TESTS()};
  if (!ret)
  std::cout << "<<<SUCCESS>>>" << std::endl;
  else
```

```
  std::cout << "FAILED" << std::endl;
  return 0;
}
```

5 STL

Listing 4: STL.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include <sstream>
#include <vector>
#include <gtest/gtest.h>

std::vector<int> v;
const int N = 5;
void input() {
  v.resize(N);
  for (int i = 0; i < N; ++i) {
    std::cin >> v[i];
  }
}
void traverse() {
  for (auto it = v.begin(); it != v.end(); ++it) {
    std::cout << *it << " ";
  }
  std::cout << std::endl;
}
void rev_traverse() {
  for (auto it = v.rbegin(); it != v.rend(); ++it) {
    std::cout << *it << " ";
  }
  std::cout << std::endl;
}

TEST(IterTest, test) {
  std::stringstream ss("1 3 2 4 5\n");
  std::stringstream out;
  std::streambuf* cout_buff = std::cout.rdbuf();
  std::cin.rdbuf(ss.rdbuf());
  std::cout.rdbuf(out.rdbuf());
  input();
  traverse();
  rev_traverse();
```

```
  std::cout.rdbuf(cout_buff);
  EXPECT_EQ(out.str(), "1 3 2 4 5 \n5 4 2 3 1 \n");
}

int main(int argc, char **argv) {
  ::testing::InitGoogleTest(&argc, argv);
  std::cout << "RUNNING TESTS ..." << std::endl;
  int ret{RUN_ALL_TESTS()};
  if (!ret)
  std::cout << "<<<SUCCESS>>>" << std::endl;
  else
  std::cout << "FAILED" << std::endl;
  return 0;
}
```

6 Linear Algebra library

Listing 5: linearalgebra.h

```
#ifndef LINEARALGEBRA_H
#define LINEARALGEBRA_H

#include <vector>
#include <cstddef>
#include <random>
#include <iostream>
#include <iomanip>
#include <stdexcept>
#include <algorithm>
#include <cmath>
#include <functional>

namespace algebra {
  using Matrix = std::vector<std::vector<double>>;
  constexpr int PRECISION = 3;

  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 concatenate(const Matrix &matrix1, const Matrix &matrix2, int axis = 0);
  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 inverse(const Matrix &matrix);
  Matrix upper_triangular(const Matrix &matrix);
}  // algebra

#endif //LINEARALGEBRA_H
```

Listing 6: linearalgebra.cpp

```
#include "linearalgebra.h"

namespace algebra {
  Matrix zeros(size_t n, size_t m) {
    return std::vector<std::vector<double>>(n, std::vector<double>(m, 0.0));
  }
  Matrix ones(size_t n, size_t m) {
    return std::vector<std::vector<double>>(n, std::vector<double>(m, 1.0));
  }

  Matrix random(size_t n, size_t m, double min, double max) {
    if (min > max) {
      throw std::logic_error("");
    }
    std::random_device rd;
    std::mt19937 mt(rd());
    std::uniform_real_distribution<double> dist(min, max);
    Matrix ret(n, std::vector<double>(m));
    for (auto &row: ret)
    for (auto &elem: row)
    elem = dist(mt);
    return ret;
  }

  void show(const Matrix &matrix) {
    int maxlen = 0;
    bool neg_flag = false;
    std::function<int(double)> get_len = [](double elem) {
      return int(std::log10(std::max(1., std::abs(elem)))) + 1;
    };
    for (auto &row: matrix) {
      for (double elem: row) {
        if (get_len(elem) > maxlen) {
```

```
        maxlen = get_len(elem);
        neg_flag = elem < 0;
      }
    }
  }
  int maxwidth = maxlen + PRECISION;
  if (PRECISION != 0) {
    maxwidth++;
  }
  if (neg_flag) {
    maxwidth++;
  }
  std::streambuf *cout_buff = std::cout.rdbuf();
  for (int i = 0; i < matrix.size(); ++i) {
    (i == 0) ? std::cout << "[[" : std::cout << " [";
    if (!matrix[0].empty()) {
      for (int j = 0; j < matrix[0].size(); ++j) {
        std::cout << std::fixed << std::setprecision(PRECISION) << std::setw(maxwidth)
        << std::setiosflags(std::ios::right) << matrix[i][j];
        if (j != matrix[0].size() - 1) {
          std::cout << " ";
        }
      }
    }
    (i == matrix.size() - 1) ? std::cout << "]]" : std::cout << "]\n";
  }
  std::cout.rdbuf(cout_buff);
}

Matrix multiply(const Matrix &matrix, double c) {
  Matrix res = matrix;
  for (int i = 0; i < matrix.size(); ++i) {
    for (int j = 0; j < matrix[0].size(); ++j) {
      res[i][j] *= c;
    }
  }
  return res;
}

Matrix multiply(const Matrix &matrix1, const Matrix &matrix2) {
  if (matrix1.empty() || matrix1[0].empty()) {
    return {};
  }
  if (matrix2.empty() || matrix2[0].empty()) {
    return {};
  }
```

```
  if (matrix1[0].size() != matrix2.size()) {
    throw std::logic_error("");
  }
  Matrix res = zeros(matrix1.size(), matrix2[0].size());
  for (int i = 0; i < matrix1.size(); ++i) {
    for (int j = 0; j < matrix2[0].size(); ++j) {
      for (int k = 0; k < matrix1[0].size(); ++k) {
        res[i][j] += matrix1[i][k] * matrix2[k][j];
      }
    }
  }
  return res;
}

Matrix sum(const Matrix &matrix, double c) {
  Matrix res = matrix;
  for (auto &row: res) {
    for (auto &elem: row) {
      elem += c;
    }
  }
  return res;
}

Matrix sum(const Matrix &matrix1, const Matrix &matrix2) {
  if (matrix1.size() != matrix2.size()) {
    throw std::logic_error("");
  }
  if (matrix1.empty() || matrix2.empty()) {
    return {};
  }
  if (matrix1[0].size() != matrix2[0].size()) {
    throw std::logic_error("");
  }
  if (matrix1[0].empty()) {
    return {};
  }
  Matrix res = matrix1;
  for (int i = 0; i < matrix1.size(); ++i) {
    for (int j = 0; j < matrix1[0].size(); ++j) {
      res[i][j] += matrix2[i][j];
    }
  }
  return res;
}

```

```cpp
Matrix transpose(const Matrix &matrix) {
  if (matrix.empty() || matrix[0].empty()) {
    return {};
  }
  Matrix res = zeros(matrix[0].size(), matrix.size());
  for (int i = 0; i < matrix.size(); ++i) {
    for (int j = 0; j < matrix[0].size(); ++j) {
      res[j][i] = matrix[i][j];
    }
  }
  return res;
}

Matrix minor(const Matrix &matrix, size_t n, size_t m) {
  if (matrix.empty()) {
    throw std::logic_error("");
  }
  if (matrix.size() <= n || matrix[0].size() <= m) {
    throw std::logic_error("");
  }
  Matrix res = zeros(matrix.size() - 1, matrix[0].size() - 1);
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m; ++j) {
      res[i][j] = matrix[i][j];
    }
    for (int j = m; j < res.size(); ++j) {
      res[i][j] = matrix[i][j + 1];
    }
  }
  for (int i = n; i < res.size(); ++i) {
    for (int j = 0; j < m; ++j) {
      res[i][j] = matrix[i + 1][j];
    }
    for (int j = m; j < res[0].size(); ++j) {
      res[i][j] = matrix[i + 1][j + 1];
    }
  }
  return res;
}

double determinant(const Matrix &matrix) {
  if (matrix.empty()) {
    return 1.;
  }
  if (matrix.size() != matrix[0].size()) {
    throw std::logic_error("");
```

```
  }
  if (matrix.size() == 1) {
    return matrix[0][0];
  }
  if (matrix.size() == 2) {
    return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0];
  }
  double ans = 0.;
  for (int i = 0; i < matrix[0].size(); ++i) {
    ans += matrix[0][i] * pow(-1, i) * determinant(minor(matrix, 0, i));
  }
  return ans;
}

Matrix concatenate(const Matrix &matrix1, const Matrix &matrix2, int axis) {
  if (axis == 0) {
    if (matrix1.empty()) return matrix2;
    if (matrix2.empty()) return matrix1;
    if (matrix1[0].size() != matrix2[0].size()) {
      throw std::logic_error("");
    }
    Matrix res = matrix1;
    for (auto &row: matrix2) {
      res.emplace_back(row);
    }
    return res;
  } else if (axis == 1) {
    if (matrix1.size() != matrix2.size()) {
      throw std::logic_error("");
    }
    if (matrix1.empty()) return {};
    if (matrix1[0].empty()) return matrix2;
    if (matrix2[0].empty()) return matrix1;
    Matrix res = matrix1;
    for (int i = 0; i < res.size(); ++i) {
      res[i].insert(res[i].end(), matrix2[i].begin(), matrix2[i].end());
    }
    return res;
  } else {
    throw std::logic_error("");
  }
}

Matrix ero_swap(const Matrix &matrix, size_t r1, size_t r2) {
  if (r1 >= matrix.size() || r2 >= matrix.size()) {
    throw std::logic_error("");
```

```
218     }
219     if (r1 == r2) return matrix;
220     Matrix res = matrix;
221     std::swap(res[r1], res[r2]);
222     return res;
223   }
224
225   Matrix ero_multiply(const Matrix &matrix, size_t r, double c) {
226     if (r >= matrix.size()) {
227       throw std::logic_error("");
228     }
229     Matrix res = matrix;
230     for (auto &elem: res[r]) {
231       elem *= c;
232     }
233     return res;
234   }
235
236   Matrix ero_sum(const Matrix &matrix, size_t r1, double c, size_t r2) {
237     if (r1 >= matrix.size() || r2 >= matrix.size()) {
238       throw std::logic_error("");
239     }
240     Matrix res = matrix;
241     for (int i = 0; i < res[0].size(); ++i) {
242       res[r2][i] += res[r1][i] * c;
243     }
244     return res;
245   }
246
247   Matrix inverse(const Matrix &matrix) {
248     if (matrix.empty() || matrix[0].empty()) {
249       return {};
250     }
251     if (matrix.size() != matrix[0].size()) {
252       throw std::logic_error("");
253     }
254     if (determinant(matrix) == 0.) {
255       throw std::logic_error("");
256     }
257     Matrix res = zeros(matrix.size(), matrix[0].size());
258     Matrix matrix_copy = matrix;
259     for (int i = 0; i < res.size(); ++i)
260     res[i][i] = 1.;
261     for (int j = 0; j < matrix.size(); ++j) {  // col
262       int i = j;
263       while (matrix_copy[i][j] == 0.) {
```

```
264         i++;
265       }
266       res = ero_swap(res, j, i);
267       matrix_copy = ero_swap(matrix_copy, j, i);
268       res = ero_multiply(res, j, 1. / matrix_copy[j][j]);
269       matrix_copy = ero_multiply(matrix_copy, j, 1. / matrix_copy[j][j]);
270       for (int k = 0; k < matrix.size(); ++k) {
271         if (k != j) {
272           res = ero_sum(res, j, -matrix_copy[k][j], k);
273           matrix_copy = ero_sum(matrix_copy, j, -matrix_copy[k][j], k);
274         }
275       }
276     }
277     return res;
278   }
279
280   Matrix upper_triangular(const Matrix &matrix) {
281     if (matrix.empty() || matrix[0].empty()) {
282       return {};
283     }
284     if (matrix.size() != matrix[0].size()) {
285       throw std::logic_error("");
286     }
287     Matrix res = matrix;
288     int row = 0;
289     for (int j = 0; j < res[0].size(); ++j) {  // col
290       if (row == res.size() - 1) {
291         break;
292       }
293       int i = row;
294       while (res[i][j] == 0. && i < res.size()) i++;
295       if (i == res.size()) {
296         row++;
297         continue;
298       }
299       res = ero_swap(res, row, i);
300       for (int k = row + 1; k < res.size(); ++k) {
301         res = ero_sum(res, row, -res[k][j] / res[row][j], k);
302       }
303       row++;
304     }
305     return res;
306   }
307 }  // algebra
```

参考GoogleTest，修改CMakeLists.txt如下。

Listing 7: CMakeLists.txt

```
cmake_minimum_required(VERSION 3.13)
project(LinearAlgebra)

set(CMAKE_CXX_STANDARD 14)
#cmake_policy(SET CMP0135 NEW)

include(FetchContent)
FetchContent_Declare(
    googletest
    URL https://github.com/google/googletest/archive/refs/tags/v1.13.0.zip
)

FetchContent_MakeAvailable(googletest)

find_package(GTest REQUIRED)
enable_testing()

include_directories(include/)

add_executable(main
    src/main.cpp
    src/linearalgebra.cpp
    src/unit_test.cpp
)
target_link_libraries(main
    GTest::GTest
    GTest::Main
)

include(GoogleTest)
gtest_discover_tests(main)
```

测试结果如下图。