Student Name: 戈天齐
Student ID: 2022141460202

c++
Assignment 3

1 Structured Binding

```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) {
// TODO
    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;
}
```

该函数将遍历 m 里面的每一个键值对，并把 key 改为希哈值
添加的一行代码，同时实现了结构绑定和范围循环，并更改 key 值

2 References

```cpp
#include <iostream>
void change(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}
int main() {
    int a;
    int b;
    std::cin >> a >> b;
    change(a, b);
    std::cout << a << " " << b << std::endl;
    return 0;
}
```

```
2 3
3 2

进程已结束,退出代码0
```

3 Stream

库函数的实现

```
#include <iostream>
#include <fstream>
#include <string>

    using namespace std;

    struct Student {
        string name;
        int score{};
    };
    int main() {
        // 读入学生信息并写入文件
        ofstream fout("stud.dat", ios::binary);
        if (!fout) {
            cerr << "Failed to open file" << endl;
            return 1;
        }
        int n;
        cout << "Enter number of students: ";
        cin >> n;
        Student stu;
        for (int i = 0; i < n; ++i) {
            cout << "Enter name and score for student " << i + 1 << ": ";
            cin >> stu.name >> stu.score;
            fout.write((char*)&stu, sizeof(stu));
        }
        fout.close();
        // 从文件中读取学生信息并显示
        ifstream fin("stud.dat", ios::binary);
        if (!fin) {
            cerr << "Failed to open file" << endl;
            return 1;
        }
        while (fin.read((char*)&stu, sizeof(stu))) {
            cout << "Name: " << stu.name << ", Score: " << stu.score << endl;
        }

        fin.close();

        return 0;
    }
```

```
/tmp/Streams/cmake-build-debug/Streams
Enter number of students: 3
Enter name and score for student 1: xiaoming 10
Enter name and score for student 2: xiaohong 9
Enter name and score for student 3: xiaohuang 8
Name: xiaoming, Score: 10
Name: xiaohong, Score: 9
Name: xiaohuang, Score: 8

进程已结束,退出代码0
```

4 STL(Containers)

```
#include <iostream>
#include <vector>
int main() {
    std::vector<int> x;
    std::cout << "输入五个整型\n";
    for(int i = 0; i < 5; i++){
        int temp;
        std::cin >> temp;
        x.push_back(temp);
    }
    std::cout << "正向迭代器遍历\n";
    for(auto it = x.begin(); it != x.end(); it++ )
    {
        std::cout << *it;
    }
    std::cout << std::endl;
    std::cout << "反向迭代器遍历\n";
    for(auto it = x.rbegin(); it != x.rend(); it++ ) {
        std::cout << *it;
    }
    return 0;
}
```

```
/tmp/STL(Containers)/cmake-build-debug/STL_Containers_
输入五个整型
0 1 2 3 4
正向迭代器遍历
01234
反向迭代器遍历
43210
进程已结束,退出代码0
```

5 Linear Algebra library

```
#ifndef LINEARALGEBRA_H
#define LINEARALGEBRA_H
#include <iostream>
#include <vector>
#include <iomanip>
#include <algorithm>
#include <random>
using Matrix = std::vector<std::vector<double>>;
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 = 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 upper_triangular(const Matrix& matrix);
}
#endif //LINEARALGEBRA_H
```

```
#include "linearalgebra.h"
using Matrix = std::vector<std::vector<double>>;

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

    Matrix ones(size_t n, size_t m) {
        Matrix ones(n, std::vector<double>(m, 1));
        return ones;
    }

    Matrix Unit(size_t n) {
        Matrix unit(n, std::vector<double>(n));
        for (int i = 0; i < unit.size(); i++) {
            unit[i][i] = 1;
        }
        return unit;
    }

    Matrix random(size_t n, size_t m, double min, double max) {
        if (max > min) {
            std::random_device rd;//创建随机数引擎
            std::mt19937 gen(rd());
            std::uniform_real_distribution<> dis(min, max);
            Matrix random(n, std::vector<double>(m));
            for (int i = 0; i < n; i++)
                for (int j = 0; j < m; j++)
                    random[i][j] = dis(gen);
            return random;
        } else {
            throw std::logic_error("min >= max");

        }
    }

    void show(const Matrix &matrix) {
        for (int i = 0; i < matrix.size(); i++) {
            std::cout << "[";
            for (int j = 0; j < matrix[0].size(); j++) {
                std::cout <<std::fixed<< std::left << std::setw(10) << std::setprecision(3) << matrix[i][j];
            }
            std::cout << "]" << std::endl;
        }
    }

    Matrix multiply(const Matrix &matrix, double c) {
        Matrix matrix_answer = matrix;
        for (int i = 0; i < matrix.size(); i++)

            for (int j = 0; j < matrix[i].size(); j++)
                matrix_answer[i][j] *= c;
        return matrix_answer;
    }

    Matrix multiply(const Matrix &matrix1, const Matrix &matrix2) {
        //矩阵可成的前提是matrix1的列 == matrix2的行数

        if (matrix1.empty() || matrix2.empty()) {
            std::cout << "Matrix is empty" << std::endl;
            return zeros(0, 0);
        }
        int m1 = matrix1.size(), n1 = matrix1[0].size();
        int m2 = matrix2.size(), n2 = matrix2[0].size();
        if (n1 == m2) {
            int m = m1, n = n2;
            Matrix matrix_answer(m, std::vector<double>(n));
            for (int i = 0; i < m; i++) {
                for (int j = 0; j < n; j++) {
                    for (int k = 0; k < m2; k++) {
                        matrix_answer[i][j] += matrix1[i][k] * matrix2[k][j];
                    }
                }
            }
            return matrix_answer;
        } else {
            throw std::logic_error("Matrix type does not match, not multiplyable");

        }

    }

    Matrix sum(const Matrix &matrix, double c) {
        Matrix matrix_answer = matrix;
        for (int i = 0; i < matrix.size(); i++)

            for (int j = 0; j < matrix[i].size(); j++)
                matrix_answer[i][j] += c;
        return matrix_answer;
    }

    Matrix sum(const Matrix &matrix1, const Matrix &matrix2) {
        if (matrix1.empty() && matrix2.empty()) {
            return zeros(0, 0);
        } else if (matrix1.empty() && !matrix2.empty() || !matrix1.empty() && matrix2.empty()) {
            throw std::logic_error("Matrix type does not match");
        }
        int m1 = matrix1.size(), n1 = matrix1[0].size();
        int m2 = matrix2.size(), n2 = matrix2[0].size();
        if (m1 == m2 && n1 == n2) {
            Matrix matrix_answer(m1, std::vector<double>(n1));
            for (int i = 0; i < m1; i++) {
                for (int j = 0; j < n1; j++) {
                    matrix_answer[i][j] = matrix1[i][j] + matrix2[i][j];
                }
            }
            return matrix_answer;
        } else {
            throw std::logic_error("Matrix type does not match");
        }
    }

    Matrix transpose(const Matrix &matrix) {
        if (matrix.empty())
            return zeros(0, 0);
        else {
            Matrix transpose(matrix[0].size(), std::vector<double>(matrix.size()));
            for (int i = 0; i < transpose.size(); i++)
                for (int j = 0; j < transpose[i].size(); j++) {
                    transpose[i][j] = matrix[j][i];
                }
            return transpose;
        }
    }

    Matrix minor(const Matrix &matrix, size_t n, size_t m) {
        if (n < matrix.size() && m < matrix[0].size()) {
            Matrix minor = matrix;
            minor.erase(minor.begin() + n);
            for (auto &i: minor)
```

```
            i.erase(i.begin() + m);
        return minor;
    } else {
        std::cout << "Out of range" << std::endl;
        return zeros(0, 0);
    }
}

double determinant(const Matrix &matrix) {
    //判断是否为空矩阵
    if (matrix.size() == 0 || matrix[0].size() == 0)
        return 1;
        //判断是否是方阵
    else if (matrix.size() != matrix[0].size()) {
        throw std::logic_error("Matrix is not square");
    } else {
        //一阶矩阵返回
        if (matrix.size() == 1) {
            return matrix[0][0];
        } else  {
            double answer = 0;
            for (int i = 0; i < matrix.size(); i++)
                answer += pow(-1, i + 1 + 1) * matrix[i][0] * determinant(minor(matrix, i, 0));
            return answer;
        }
    }
}

//交换两行
Matrix ero_swap(const Matrix &matrix, size_t r1, size_t r2) {
    if (r1 < matrix.size() && r2 < matrix.size() && r1 >= 0 && r2 >= 0) {
        Matrix answer = matrix;
        std::swap(answer[r1], answer[r2]);
        return answer;
    } else {

        throw std::logic_error("Out of range");
    }
}

//倍增
Matrix ero_multiply(const Matrix &matrix, size_t r, double c) {
    if (r < matrix.size()) {
        Matrix answer = matrix;
        for (int i = 0; i < matrix[i].size(); i++) {
            answer[r][i] *= c;
        }
        return answer;
    } else {
        std::cout << "Out of range" << std::endl;
        return zeros(0, 0);
    }
}

//倍加
Matrix ero_sum(const Matrix &matrix, size_t r1, double c, size_t r2) {
    Matrix answer = matrix;
    if (r1 < matrix.size() && r2 < matrix.size() && r1 >= 0 && r2 >= 0) {
        for (int i = 0; i < matrix[0].size(); i++) {
            answer[r2][i] += answer[r1][i] * c;
        }
    }
    return answer;
}

//上三角
Matrix upper_triangular(const Matrix &matrix) {
    if (0 == matrix.size()) {
        return zeros(0, 0);
    } else if (matrix.size() == matrix[0].size()) {
        Matrix answer = matrix;
        for (int i = 0; i < matrix[0].size() - 1; i++) {
            if(answer[i][i] == 0)
            {
                for (int j = i + 1; j < matrix.size(); j++) {
                    if (answer[j][i] != 0) {
                        answer = ero_swap(answer, i, j);
                        break;
                    }
                }
            }
            for (int j = i + 1; j < matrix.size(); j++) {
                answer = ero_sum(answer, i, -answer[j][i] / answer[i][i], j);
            }
        }
        return answer;
    } else {
        throw std::logic_error("Matrix is not square");
    }
}

//矩阵求逆
```

```
    Matrix inverse(const Matrix &matrix) {
        //行列式存在且不等于0,并且不是空矩阵
        if (matrix.size() == 0 || matrix[0].size() == 0) {
            std::cout << "Empty matrix" << std::endl;
            return zeros(0, 0);
        } else  if(determinant(matrix) == 0) {
            throw std::logic_error("non_singular_matrix");
        } else {
            //求出上三角矩阵
            Matrix unit_to_inverse = Unit(matrix.size());
            Matrix matrix_to_unit = matrix;
            for (int i = 0; i < matrix[0].size() - 1; i++) {
                if(matrix_to_unit[i][i] == 0)
                {
                    for (int j = i + 1; j < matrix.size(); j++) {
                        if (matrix_to_unit[j][i] != 0) {
                            matrix_to_unit = ero_swap(matrix_to_unit, i, j);
                            unit_to_inverse = ero_swap(unit_to_inverse, i, j);
                            break;
                        }
                    }
                }
                for (int j = i + 1; j < matrix.size(); j++) {
                    unit_to_inverse = ero_sum(unit_to_inverse, i, -matrix_to_unit[j][i] / matrix_to_unit[i][i], j);
                    matrix_to_unit = ero_sum(matrix_to_unit, i, -matrix_to_unit[j][i] / matrix_to_unit[i][i], j);
                }
            }
            for (int i = 1; i < matrix_to_unit.size(); i++) {
                if(matrix_to_unit[i][i] == 0)
                {
                    for (int j = i + 1; j < matrix.size(); j++) {
                        if (matrix_to_unit[j][i] != 0) {
                            matrix_to_unit = ero_swap(matrix_to_unit, i, j);
                            unit_to_inverse = ero_swap(unit_to_inverse, i, j);
                            break;
                        }
                    }
                }
                for (int j = 0; j < i; j++) {
                    unit_to_inverse = ero_sum(unit_to_inverse, i, -matrix_to_unit[j][i] / matrix_to_unit[i][i], j);
                    matrix_to_unit = ero_sum(matrix_to_unit, i, -matrix_to_unit[j][i] / matrix_to_unit[i][i], j);
                }
```

```cpp
            }
            for (int i = 0; i < matrix_to_unit.size(); i++) {
                for(int j = 0; j < matrix_to_unit[0].size(); j++){
                    unit_to_inverse[i][j] /= matrix_to_unit[i][i];
                    matrix_to_unit[i][j] /= matrix_to_unit[i][i];
                }
            }
            return unit_to_inverse;
        }
    }
    //矩阵结合
    Matrix concatenate(const Matrix &matrix1, const Matrix &matrix2, int axis) {
        if (!axis && matrix1[0].size() == matrix2[0].size()) {
            Matrix concatenate(matrix1.size() + matrix2.size(), std::vector<double>(matrix2[0].size()));
            for (int i = 0; i < matrix1.size() + matrix2.size(); i++) {
                concatenate[i] = i < matrix1.size() ? matrix1[i] : matrix2[i - matrix1.size()];
            }
            return concatenate;
        } else if (!axis && matrix1[0].size() != matrix2[0].size()) {
            throw std::logic_error("Matrix size not match");
        } else if (axis && matrix1.size() == matrix2.size()) {
            Matrix concatenate(matrix1.size(), std::vector<double>(matrix1[0].size() + matrix2[0].size()));
            for (int j = 0; j < matrix1[0].size() + matrix2[0].size(); j++)
                for (int i = 0; i < matrix1.size(); i++) {
                    concatenate[i][j] = j < matrix1[0].size() ? matrix1[i][j] : matrix2[i][j - matrix1[0].size()];
                }
            return concatenate;
        } else {
            throw std::logic_error("Matrix size not match");
        }
    }
}
```

编译运行

```
root@fa571cbced78:/ws# cd /ws/LinearAlgebra
root@fa571cbced78:/ws/LinearAlgebra# mkdir build
root@fa571cbced78:/ws/LinearAlgebra# cd build
root@fa571cbced78:/ws/LinearAlgebra/build# cmake..
bash: cmake..: command not found
root@fa571cbced78:/ws/LinearAlgebra/build# cmake ..
-- The C compiler identification is GNU 11.2.0
-- The CXX compiler identification is GNU 11.2.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/local/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found GTest: /usr/local/lib/libgtest.a
-- Configuring done
-- Generating done
-- Build files have been written to: /ws/LinearAlgebra/build
root@fa571cbced78:/ws/LinearAlgebra/build# make
Scanning dependencies of target main
[ 25%] Building CXX object CMakeFiles/main.dir/src/main.cpp.o
[ 50%] Building CXX object CMakeFiles/main.dir/src/linearalgebra.cpp.o
[ 75%] Building CXX object CMakeFiles/main.dir/src/unit_test.cpp.o
[100%] Linking CXX executable main
[100%] Built target main
root@fa571cbced78:/ws/LinearAlgebra/build# ls
CMakeCache.txt  CMakeFiles  Makefile  cmake_install.cmake  main
root@fa571cbced78:/ws/LinearAlgebra/build# ./main
RUNNING TESTS ...
[==========] Running 24 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 24 tests from LinearAlgebraTest
[ RUN      ] LinearAlgebraTest.ZEROS
[       OK ] LinearAlgebraTest.ZEROS (0 ms)
[ RUN      ] LinearAlgebraTest.ONES
[       OK ] LinearAlgebraTest.ONES (0 ms)
[ RUN      ] LinearAlgebraTest.RANDOM1
random matrix [-5, 7)
[1.351     5.473     2.712     -3.406    ]
[4.551     0.999     -2.760    3.132     ]
[-2.704    5.518     1.233     2.779     ]
[5.497     -3.345    2.917     -3.365    ]

[       OK ] LinearAlgebraTest.RANDOM1 (0 ms)
[ RUN      ] LinearAlgebraTest.RANDOM2
[       OK ] LinearAlgebraTest.RANDOM2 (0 ms)
[ RUN      ] LinearAlgebraTest.MULTIPLY1
[       OK ] LinearAlgebraTest.MULTIPLY1 (0 ms)
[ RUN      ] LinearAlgebraTest.MULTIPLY2
Matrix is empty
[       OK ] LinearAlgebraTest.MULTIPLY2 (0 ms)
[ RUN      ] LinearAlgebraTest.MULTIPLY3
[       OK ] LinearAlgebraTest.MULTIPLY3 (0 ms)
[ RUN      ] LinearAlgebraTest.MULTIPLY4
[       OK ] LinearAlgebraTest.MULTIPLY4 (0 ms)
[ RUN      ] LinearAlgebraTest.SUM1
[       OK ] LinearAlgebraTest.SUM1 (0 ms)
[ RUN      ] LinearAlgebraTest.SUM2
[       OK ] LinearAlgebraTest.SUM2 (0 ms)
[ RUN      ] LinearAlgebraTest.TRANSPOSE
[       OK ] LinearAlgebraTest.TRANSPOSE (0 ms)
[ RUN      ] LinearAlgebraTest.MINOR1
[       OK ] LinearAlgebraTest.MINOR1 (0 ms)
[ RUN      ] LinearAlgebraTest.MINOR2
[       OK ] LinearAlgebraTest.MINOR2 (0 ms)
[ RUN      ] LinearAlgebraTest.DETERMINANT1
[       OK ] LinearAlgebraTest.DETERMINANT1 (1 ms)
[ RUN      ] LinearAlgebraTest.DETERMINANT2
[       OK ] LinearAlgebraTest.DETERMINANT2 (0 ms)
[ RUN      ] LinearAlgebraTest.INVERSE1
Empty matrix
[       OK ] LinearAlgebraTest.INVERSE1 (0 ms)
[ RUN      ] LinearAlgebraTest.INVERSE2
[       OK ] LinearAlgebraTest.INVERSE2 (0 ms)
[ RUN      ] LinearAlgebraTest.CONCATENATE1
[       OK ] LinearAlgebraTest.CONCATENATE1 (0 ms)
[ RUN      ] LinearAlgebraTest.CONCATENATE2
[       OK ] LinearAlgebraTest.CONCATENATE2 (0 ms)
[ RUN      ] LinearAlgebraTest.ERO_SWAP
[       OK ] LinearAlgebraTest.ERO_SWAP (0 ms)
[ RUN      ] LinearAlgebraTest.ERO_MULTIPLY
[       OK ] LinearAlgebraTest.ERO_MULTIPLY (0 ms)
[ RUN      ] LinearAlgebraTest.ERO_SUM
[       OK ] LinearAlgebraTest.ERO_SUM (0 ms)
[ RUN      ] LinearAlgebraTest.UPPER_TRIANGULAR1
[       OK ] LinearAlgebraTest.UPPER_TRIANGULAR1 (0 ms)
[ RUN      ] LinearAlgebraTest.BONUS
[       OK ] 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>>>
```