Student Name: 王研博
Student ID: 2022141460122

**高级语言程序设计**
**Assignment 3**

## Section 1: 作业内容概述

## Section 2: Structed 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)
{
    for(auto &[x, y] :m) y = foo(x);

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

## Section 3: References

```cpp
#include <iostream>
using namespace std;
void swap(int &a, int &b)
{
    int temp = a;
    a = b;
    b = temp;
}
int main()
{
    int a = 10, b = 100;
    swap(a, b);
    cout<<a<<" "<<b<<endl;
    return 0;
}
```

## Section 4: Streams

```
#include <iostream>
#include <string>
#include <fstream>
using namespace std;
class Student
{
public:
    Student(string &name, int &score)
    {
        this->name = name;
        this->score = score;
    }
    void setName(string &name)
    {
        this->name = name;
    }
    void setScore(int score)
    {
        this->score = score;
    }
    int getScore()
    {
        return score;
    }
    string getName()
    {
        return name;
    }
private:
    string name;
    int score;
};
```

```
int main()
{
    string name;
    int score;
    Student stu(name, score);
    ofstream out("stud.dat");
    if(out.is_open())
    {
        while(cin >> name >> score)
        {
            stu.setName(name);
            stu.setScore(score);
            out << stu.getName() << " " << stu.getScore() << endl;
        }
    }
    else
    {
        cout << "open file failed" << endl;
    }

    ifstream in("stud.dat");
    if(in.is_open())
    {
        string line;
        while(getline(in, line))
        {
            cout << line << endl;
        }
        in.close();
    }
    else
    {
        cout << "open file failed" << endl;
    }
    return 0;
}
```

## Section 5: STL(Containers)

```
#include <iostream>
#include <vector>
using namespace std;
void Traversal(vector<int> v)
{
    for(vector<int>::iterator it = v.begin(); it != v.end(); it++)
    {
        cout<<*it<<' ';
    }
    cout<<endl;
}
void rTraversal(vector<int> v)
{
    for(vector<int>::reverse_iterator it = v.rbegin(); it != v.rend(); it++)
    {
        cout<<*it<<' ';
    }
    cout<<endl;
}
void test()
{
    vector<int> v;
    int num;
    for(int i = 0; i < 5; i++)
    {
        cin>>num;
        v.push_back(num);
    }
    Traversal(v);
    rTraversal(v);
}
int main()
{
    test();
    return 0;
}
```

# Section 6: Linear Algebra Library

## Code

linearalgebra.h

```
#include <iostream>
#include <vector>
#include <cmath>
#include <algorithm>
#include <random>
#include <iomanip>

```

```cpp
using Matrix = std::vector<std::vector<double>>;

Matrix construct_matrix(size_t m, size_t n);

Matrix zeros(size_t m, size_t n);

Matrix ones(size_t m, size_t n);

Matrix random(size_t m, size_t n, int min, int max);

Matrix show(const Matrix &A);

Matrix multiply(const Matrix &A, double c);

Matrix multiply(const Matrix &A, const Matrix &B);

Matrix sum(const Matrix &A, double c);

Matrix sum(const Matrix &A, const Matrix &B);

Matrix transpose(const Matrix &A);

Matrix minor(const Matrix &A, size_t x, size_t y);

double determinant(const Matrix &A);

Matrix inverse(const Matrix &A);

Matrix unit_matrix(size_t n);

Matrix ero_swap(const Matrix &A, size_t x, size_t y);

Matrix ero_multiply(Matrix A, size_t x, double c);

Matrix ero_sum(Matrix A, size_t x, double c, size_t y);

Matrix concatenate(const Matrix &A, const Matrix &B, int axis);

Matrix upper_triangular(const Matrix &matrix);
```

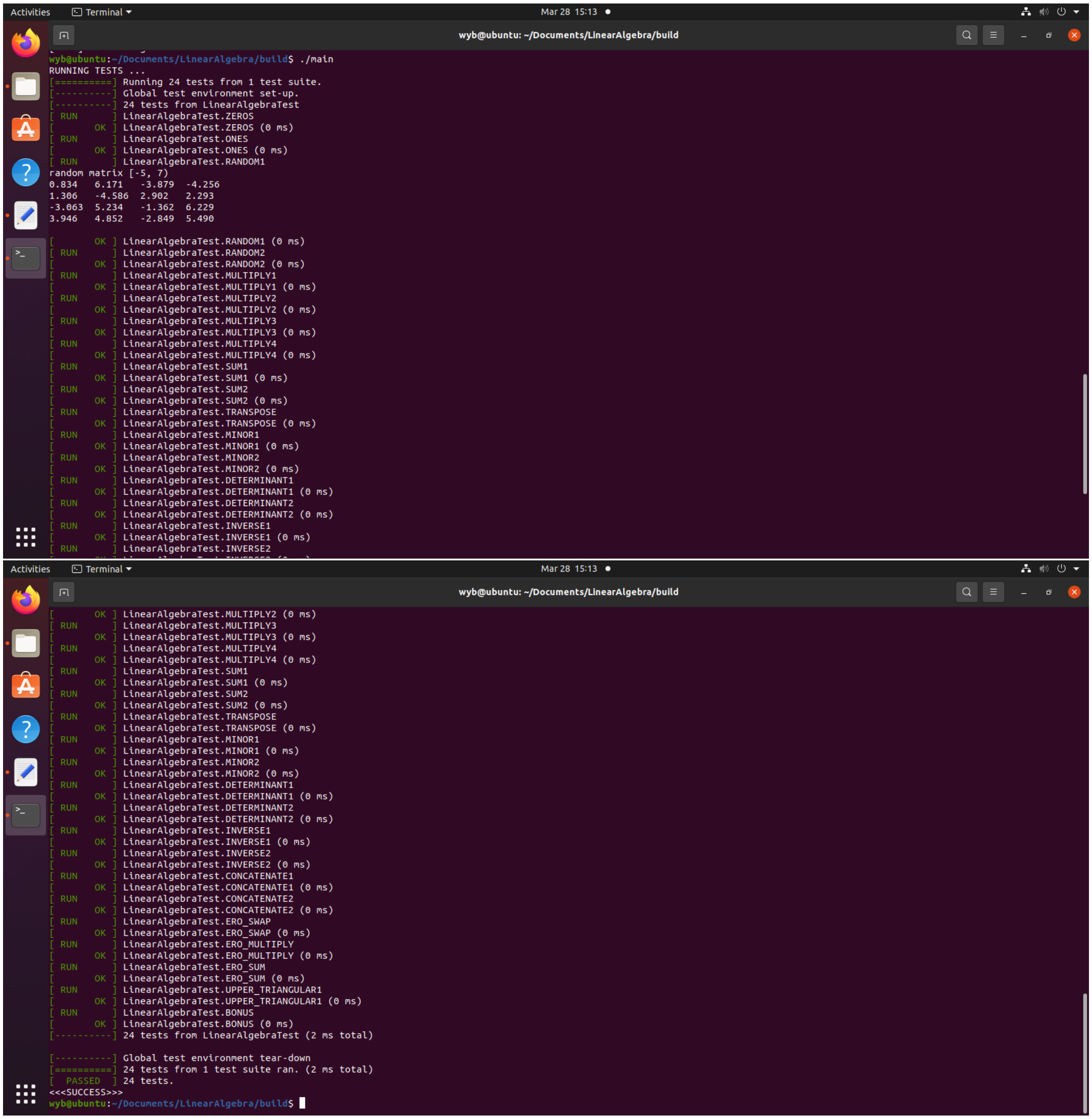
linearalgebra.cpp

```cpp
Matrix construct_matrix(size_t m, size_t n) {
    if (m == 0 || n == 0) {
        std::cout << "Matrix is empty" << std::endl;
        return Matrix();
    }
    Matrix A(m, std::vector<double>(n, 0.0));
    for (size_t i = 0; i < m; i++) {
        for (size_t j = 0; j < n; j++) {
            std::cin >> A[i][j];
        }
    }
    return A;
}

Matrix zeros(size_t m, size_t n) {
    Matrix A(m, std::vector<double>(n, 0.0));
    return A;
}

Matrix ones(size_t m, size_t n) {
    Matrix A(m, std::vector<double>(n, 1.0));
    return A;
}

Matrix random(size_t m, size_t n, int min, int max) {
    if (min > max) {
        throw (std::logic_error(""));
        return {};
    }
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_real_distribution<> dis(min, max);
    Matrix A(m, std::vector<double>(n, 0.0));
    for (size_t i = 0; i < m; i++) {
        for (size_t j = 0; j < n; j++) {
            A[i][j] = dis(gen);
        }
    }
    return A;
}

Matrix show(const Matrix &A) {
    if (A.empty()) {
        std::cout << "Matrix is empty" << std::endl;
        return A;
    }
    for (size_t i = 0; i < A.size(); i++) {
        for (size_t j = 0; j < A[i].size(); j++) {
            printf("%-8.3f", A[i][j]);
        }
        std::cout << std::endl;
    }
    return A;
}

Matrix multiply(const Matrix &A, double c) {
    Matrix B = zeros(A.size(), A[0].size());
    for (size_t i = 0; i < A.size(); i++) {
        for (size_t j = 0; j < A[i].size(); j++) {
            B[i][j] = A[i][j] * c;
        }
    }
    return B;
}

Matrix multiply(const Matrix &A, const Matrix &B) {
    if (A.empty() || B.empty()) {
        return {};
    }
    if (A[0].size() != B.size()) {
        throw std::logic_error("");
    }
    Matrix C = zeros(A.size(), B[0].size());
    for (size_t i = 0; i < A.size(); i++) {
        for (size_t j = 0; j < B[0].size(); j++) {
            for (size_t k = 0; k < A[0].size(); k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
    return C;
}

Matrix sum(const Matrix &A, double c) {
    if (A.empty()) {
        return {};
    }
    Matrix B = zeros(A.size(), A[0].size());
    for (size_t i = 0; i < A.size(); i++) {
        for (size_t j = 0; j < A[i].size(); j++) {
            B[i][j] = A[i][j] + c;
        }
    }
    return B;
}

Matrix sum(const Matrix &A, const Matrix &B) {
```

```cpp
    if (A.empty() && B.empty()) {
        return {};
    }
    if ((A.empty() && !B.empty()) || (!A.empty() && B.empty())) {
        throw std::logic_error("");
    }
    Matrix C = zeros(A.size(), A[0].size());
    for (size_t i = 0; i < A.size(); i++) {
        for (size_t j = 0; j < A[i].size(); j++) {
            C[i][j] = A[i][j] + B[i][j];
        }
    }
    return C;
}

Matrix transpose(const Matrix &A) {
    if (A.empty()) {
        return {};
    }
    Matrix B = zeros(A[0].size(), A.size());
    for (auto i = 0; i < A[0].size(); i++) {
        for (auto j = 0; j < A.size(); j++) {
            B[i][j] = A[j][i];
        }
    }
    return B;
}

Matrix minor(const Matrix &A, size_t x, size_t y) {
    Matrix B = zeros(A.size(), A[0].size());
    for (auto i = 0; i < A.size(); i++) {
        for (auto j = 0; j < A[0].size(); j++) {
            B[i][j] = A[i][j];
        }
    }
    B.erase(B.begin() + x);
    for (auto i = 0; i < B.size(); i++) {
        B[i].erase(B[i].begin() + y);
    }
    return B;
}

double determinant(const Matrix &A) {
    if (A.empty()) {
        return 1;
    }
    if (A.size() != A[0].size()) {
        throw std::logic_error("");
    }
    if (A.size() == 1) {
        return A[0][0];
    }
    if (A.size() == 2) {
        return A[0][0] * A[1][1] - A[0][1] * A[1][0];
    }
    if (A.size() > 2) {
        double det = 0;
        for (auto i = 0; i < A.size(); i++) {
            det += A[0][i] * pow(-1, i) * determinant(minor(A, 0, i));
        }
        return det;
    }
}

Matrix ero_swap(const Matrix &A, size_t x, size_t y) {
    Matrix ret = A;
    if (x >= A.size() || y >= A.size()) {
        throw std::logic_error("");
    }
    for (auto i = 0; i < A[0].size(); i++) {
        std::swap(ret[x][i], ret[y][i]);
    }
    return ret;
}

Matrix ero_multiply(Matrix A, size_t x, double c) {
    if (x >= A.size()) {
        throw std::logic_error("");
    }
    for (auto i = 0; i < A[0].size(); i++) {
        A[x][i] *= c;
    }
    return A;
}

Matrix ero_sum(Matrix A, size_t x, double c, size_t y) {
    for (auto i = 0; i < A[0].size(); i++) {
        A[y][i] += c * A[x][i];
    }
    return A;
}

```

```
Matrix unit_matrix(size_t n) {
    Matrix A = zeros(n, n);
    for (auto i = 0; i < n; i++) {
        A[i][i] = 1;
    }
    return A;
}

Matrix inverse(const Matrix &A) {
    if (A.empty()) {
        return {};
    }
    if (determinant(A) == 0 || A.size() != A[0].size()) {
        throw std::logic_error("");
    }
    Matrix B = zeros(A.size(), A[0].size());
    for (auto i = 0; i < A.size(); i++) {
        for (auto j = 0; j < A[0].size(); j++) {
            B[i][j] = A[i][j];
        }
    }
    Matrix C = unit_matrix(A.size());
    for (auto i = 0; i < A.size(); i++) {
        if (B[i][i] == 0) {
            for (auto j = i + 1; j < A.size(); j++) {
                if (B[j][i] != 0) {
                    C = ero_swap(C, i, j);
                    B = ero_swap(B, i, j);

                    break;
                }
            }
        }
        C = ero_multiply(C, i, 1 / B[i][i]);
        B = ero_multiply(B, i, 1 / B[i][i]);

        for (auto j = 0; j < A.size(); j++) {
            if (j != i && B[j][i] != 0) {
                C = ero_sum(C, i, -B[j][i], j);
                B = ero_sum(B, i, -B[j][i], j);

            }
        }
    }
    return C;
}
```

```

Matrix concatenate(const Matrix &A, const Matrix &B, int axis) {
    if (axis == 0) {
        if (A[0].size() != B[0].size()) {
            throw std::logic_error("");
        }
        Matrix C = zeros(A.size() + B.size(), A[0].size());
        for (auto i = 0; i < A.size(); i++) {
            for (auto j = 0; j < A[0].size(); j++) {
                C[i][j] = A[i][j];
            }
        }
        for (auto i = 0; i < B.size(); i++) {
            for (auto j = 0; j < B[0].size(); j++) {
                C[i + A.size()][j] = B[i][j];
            }
        }
        return C;
    }
    if (axis == 1) {
        if (A.size() != B.size()) {
            throw std::logic_error("");
        }
        Matrix C = zeros(A.size(), A[0].size() + B[0].size());
        for (auto i = 0; i < A.size(); i++) {
            for (auto j = 0; j < A[0].size(); j++) {
                C[i][j] = A[i][j];
            }
        }
        for (auto i = 0; i < B.size(); i++) {
            for (auto j = 0; j < B[0].size(); j++) {
                C[i][j + A[0].size()] = B[i][j];
            }
        }
        return C;
    }
}

Matrix upper_triangular(const Matrix &matrix)
{
    if (matrix.empty()) {
        return {};
    }
    if (matrix.size() != matrix[0].size()) {
        throw std::logic_error("");
    }
    Matrix A = zeros(matrix.size(), matrix[0].size());
    for (auto i = 0; i < matrix.size(); i++) {
        for (auto j = 0; j < matrix[0].size(); j++) {
            A[i][j] = matrix[i][j];
        }
    }
    for(auto i = 0; i < A.size(); i++)
    {
        for(auto j = 0; j < A.size(); j++)
        {
            if(A[i][i] == 0)
            {
                for(auto k = i + 1; k < A.size(); k++)
                {
                    if(A[k][i] != 0)
                    {
                        A = ero_swap(A, i, k);
                        break;
                    }
                }
            }
            if(j > i)
            {
                A = ero_sum(A, i, -A[j][i] / A[i][i], j);
            }
        }
    }
    return A;
}
```



```
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
[       OK ] LinearAlgebraTest.ZEROS (0 ms)
[ RUN      ] LinearAlgebraTest.ONES
[       OK ] 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

[       OK ] LinearAlgebraTest.RANDOM1 (0 ms)
[ RUN      ] LinearAlgebraTest.RANDOM2
[       OK ] LinearAlgebraTest.RANDOM2 (0 ms)
[ RUN      ] LinearAlgebraTest.MULTIPLY1
[       OK ] LinearAlgebraTest.MULTIPLY1 (0 ms)
[ RUN      ] LinearAlgebraTest.MULTIPLY2
[       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 (0 ms)
[ RUN      ] LinearAlgebraTest.DETERMINANT2
[       OK ] LinearAlgebraTest.DETERMINANT2 (0 ms)
[ RUN      ] LinearAlgebraTest.INVERSE1
[       OK ] LinearAlgebraTest.INVERSE1 (0 ms)
[ RUN      ] LinearAlgebraTest.INVERSE2
```

```
[       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 (0 ms)
[ RUN      ] LinearAlgebraTest.DETERMINANT2
[       OK ] LinearAlgebraTest.DETERMINANT2 (0 ms)
[ RUN      ] LinearAlgebraTest.INVERSE1
[       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>>>
wyb@ubuntu:~/Documents/LinearAlgebra/build$
```

Passed all googletest samples