计算机学院 高级程序语言设计 课程实验报告

实验题目：综合实验 2		学号：202300130150
日期：2024.5.27	班级： 4	姓名： 王成意

实验目的：

无

实验步骤与内容：

头文件内容（包括函数实现）：

```
#pragma once
#ifndef TWJ_H
#define TWJ_H
#include<iostream>
#include<fstream>
#include<vector>
#include<cassert>
#include<cmath>
using namespace std;

template<typename T>
class Point {
public:
    Point(T x, T y, T z) : x(x), y(y), z(z) {}
    Point(Point<T>& kkk)
    {
        this->x = kkk.x;
        this->y = kkk.y;
        this->z = kkk.z;
    }
    T x, y, z;
    void show()
    {
        cout << x << " " << y << " " << z << endl;
    }
};
template<typename T>
T distance(Point<T>* x, Point<T>* y)
{
    return sqrt((x->x - y->x) * (x->x - y->x) + (x->y - y->y) * (x->y - y->y) +
(x->z - y->z) * (x->z - y->z));
```

```
}

template<typename T>
class Facet {
public:
    int n = 0;
    vector<Point<T>*> vertices;
    Facet(int x) :n(x) {}
    T area();
};

template<typename T>
class Polyhedron {
public:
    vector<Point<T>*> vertices;
    vector<Facet<T>*> facets;
    void addVertex(Point<T>* x) {
        vertices.push_back(x);
    }
    void addfacet(Facet<T>* x) {
        facets.push_back(x);
    }
    Polyhedron(const char* path);
    ~Polyhedron();
    Point<T> get_low();
    Point<T> get_high();
    T area();
};

template<typename T>
inline Polyhedron<T>::Polyhedron(const char* path)
{
    ifstream fin;
    fin.open(path, ios::in);
    assert(fin.is_open() != 0);
    string pd;
    fin >> pd;
    assert(pd == "OFF");
    int points, facets, edges;
    fin >> points >> facets >> edges;
    T x = 0, y = 0, z = 0;
    for (int i = 1; i <= points; ++i)
    {
        fin >> x >> y >> z;
```

```
            addVertex(new Point<T>(x, y, z));
        }
        int nn = 0, ww = 0;
        for (int i = 1; i <= facets; ++i)
        {
            fin >> ww;
            Facet<T>* now = new Facet<T>(ww);
            for (int j = 1; j <= ww; ++j)
            {
                fin >> nn;
                now->vertices.push_back(this->vertices[nn]);
            }
            addfacet(now);
        }
}
template<typename T>
Polyhedron<T>::~Polyhedron()
{
    for (auto u : this->facets)
    {
        delete u;
    }
    for (auto u : this->vertices)
    {
        delete u;
    }
}
template<typename T>
inline Point<T> Polyhedron<T>::get_low()
{
    Point<T> kkk(*(this->vertices[0]));
    for (auto v : this->vertices)
    {
        if (v->x < kkk.x)
        {
            kkk.x = v->x;
        }
        if (v->y < kkk.y)
        {
            kkk.y = v->y;
        }
        if (v->z < kkk.z)
        {
            kkk.z = v->z;
```

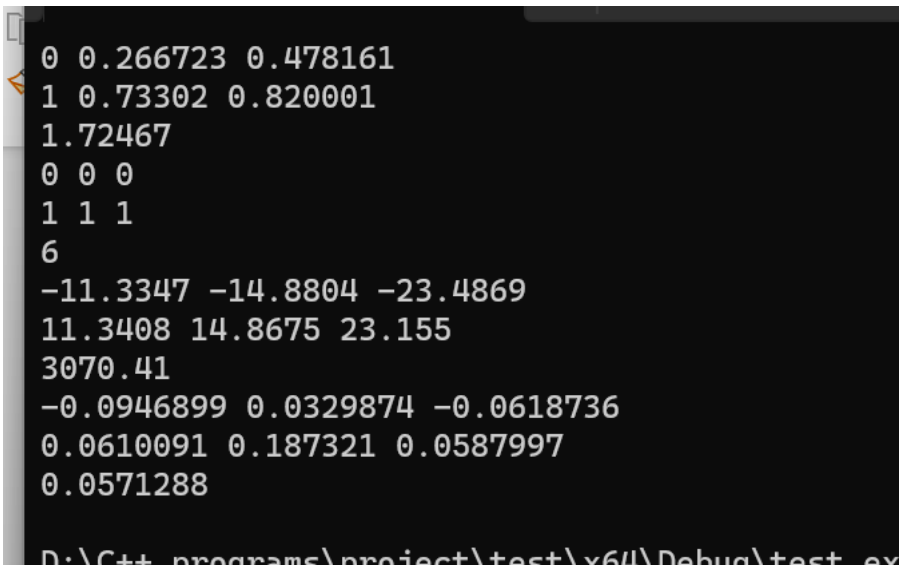
```
        }
    }
    return kkk;
}
template<typename T>
inline Point<T> Polyhedron<T>::get_high()
{
    Point<T> kkk(*(this->vertices[0]));
    for (auto v : this->vertices)
    {
        if (v->x > kkk.x)
        {
            kkk.x = v->x;
        }
        if (v->y > kkk.y)
        {
            kkk.y = v->y;
        }
        if (v->z > kkk.z)
        {
            kkk.z = v->z;
        }
    }
    return kkk;
}

template<typename T>
T Facet<T>::area()
{
    T ans = 0;
    for (int i = 1; i <= this->n - 2; ++i)
    {
        T x1 = distance(this->vertices[0], this->vertices[i]);
        T x2 = distance(this->vertices[0], this->vertices[i + 1]);
        T x3 = distance(this->vertices[i], this->vertices[i + 1]);
        T p = (x1 + x2 + x3) / 2;
        ans += sqrt(p * (p - x1) * (p - x2) * (p - x3));
    }
    return ans;
}

template<typename T>
T Polyhedron<T>::area()
{
```

```
    T ans = 0;
    for (auto u : this->facets)
    {
        ans += u->area();
    }
    return ans;
}
#endif
```

Main 函数：

```
#include"twj.h"
int main()
{
    Polyhedron<double> x("3holes.off");
    x.get_low().show();
    x.get_high().show();
    cout << x.area() << endl;
    Polyhedron<double> y("px6mt.off");
    y.get_low().show();
    y.get_high().show();
    cout << y.area() << endl;
    Polyhedron<double> z("elephant.off");
    z.get_low().show();
    z.get_high().show();
    cout << z.area() << endl;
    Polyhedron<double> kkk("bunny.off");
    kkk.get_low().show();
    kkk.get_high().show();
    cout << kkk.area() << endl;
    return 0;
}
```

运行结果：



分析实现：

构造函数：

```
inline Polyhedron<T>::Polyhedron(const char* path)
{
    ifstream fin;
    fin.open(path, ios::in);
    assert(fin.is_open() != 0);
    string pd;
    fin >> pd;
    assert(pd == "OFF");
    int points, facets, edges;
    fin >> points >> facets >> edges;
    T x = 0, y = 0, z = 0;
    for (int i = 1; i <= points; ++i)
    {
        fin >> x >> y >> z;
        addVertex(new Point<T>(x, y, z));
    }
    int nn = 0, ww = 0;
    for (int i = 1; i <= facets; ++i)
    {
        fin >> ww;
        Facet<T>* now = new Facet<T>(ww);
        for (int j = 1; j <= ww; ++j)
        {
            fin >> nn;
            now->vertices.push_back(this->vertices[nn]);
        }
        addfacet(now);
```

```
    }
  }
```

使用了 ifstream 库中的>>运算符，读入文件内容。第一个 assert 保证文件被破坏就无法运行，第二个 assert 验证第一行 OFF。随后按照实验要求正常读入。

析构函数：

枚举 vector 库中的每个元素进行 delete：

```
template<typename T>
Polyhedron<T>::~Polyhedron()
{
    for (auto u : this->facets)
    {
        delete u;
    }
    for (auto u : this->vertices)
    {
        delete u;
    }
    }
```

Get_low ()

```
template<typename T>
inline Point<T> Polyhedron<T>::get_low()
{
    Point<T> kkk(*(this->vertices[0]));
    for (auto v : this->vertices)
    {
        if (v->x < kkk.x)
        {
            kkk.x = v->x;
        }
        if (v->y < kkk.y)
        {
            kkk.y = v->y;
        }
        if (v->z < kkk.z)
        {
            kkk.z = v->z;
        }
    }
    return kkk;
  }
```

首先构造一个待返回的值 kkk（初始化成 vector 中的第一个元素），然后遍历枚举 vertex，找到所有的 min（缝合？），返回最后的 kkk。Get_high（）同理，略。

Distance 函数：

因为我没有在类里写 private，所以 distance 在类外写的。

```
template<typename T>
T distance(Point<T>* x, Point<T>* y)
{
    return sqrt((x->x - y->x) * (x->x - y->x) + (x->y - y->y) * (x->y - y->y) +
(x->z - y->z) * (x->z - y->z));
}
```

T Facet<T>::area()

```
T Facet<T>::area()
{
    T ans = 0;
    for (int i = 1; i <= this->n - 2; ++i)
    {
        T x1 = distance(this->vertices[0], this->vertices[i]);
        T x2 = distance(this->vertices[0], this->vertices[i + 1]);
        T x3 = distance(this->vertices[i], this->vertices[i + 1]);
        T p = (x1 + x2 + x3) / 2;
        ans += sqrt(p * (p - x1) * (p - x2) * (p - x3));
    }
    return ans;
}
```

按照实验指南上给出的方法，分成了好多三角形实现。

T Polyhedron<T>::area()

按照实验指南给出的方法，把所有的面加起来。

```
template<typename T>
T Polyhedron<T>::area()
{
    T ans = 0;
    for (auto u : this->facets)
    {
        ans += u->area();
    }
    return ans;
}
```

结论分析与体会：

非常好的实验，考点很综合。