МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА» (БГТУ им. В.Г. Шухова)

Кафедра программного обеспечения

вычислительной техники и автоматизированных

систем

Лабораторная работа №9

по дисциплине: ООП

тема: «**Использование стандартной библиотеки шаблонов STL**»

Выполнил: студент группы
ПВ-233
Мороз Роман Алексеевич

Проверили:
Морозов Данила Александрович

Белгород 2025 г.

Цель работы: знакомство со стандартной библиотекой шаблонов в C++; получение навыков использования классов контейнеров, итераторов, алгоритмов.

Содержание работы

Разработать программное обеспечения для решения соответствующего варианта. Оформить отчет. Для реализации поставленных задач требуется использовать следующие библиотеки классов: list, vector, queue, iostream, algorithm, set, iterator, map, stack.

Вариант 4.

Разработать программное обеспечение для решения следующей задачи: загрузка формата *.obj в программу, обработка объекта путем добавления цвета отображения различных элементов объекта. Вывести полученные списки на экран. Выполнить сложение *.obj объектов, путем удаления лишних точек, лежащих внутри новой поверхности. Организовать сортировку точек, треугольников. Класс для хранения *.obj представляет собой набор из двух list. Один точки, другой треугольники.

```
#pragma once

#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
#include <fstream>
#include <list>
#include <unordered_map>
#include <functional>
#include <string>
#include <cstdint>
#include <sstream>

#define EPSILON 1e-6

class Point {
private:
   _Float32 x;
   _Float32 y;
   _Float32 z;
```

```
public:
    Point(_Float32 x = 0, _Float32 y = 0, _Float32 z = 0) :
        x(x), y(y), z(z) {}

    ~Point() = default;

    bool operator==(const Point& other) const noexcept;

    bool operator<(const Point& other) const noexcept ;

    _Float32 distance(const Point& other) const;

    _Float32 dot(const Point& other) const;

    inline _Float32 getX() const;
    inline _Float32 getY() const;
    inline _Float32 getZ() const;

    inline void setX(_Float32 x);
    inline void setY(_Float32 y);
    inline void setZ(_Float32 z);
};

class Triangle {
private:
    uint32_t v1;
    uint32_t v2;
    uint32_t v3;
    uint32_t color;

public:
    Triangle(uint32_t v1 = 0, uint32_t v2 = 0, uint32_t v3 = 0, uint32_t
color = 0) :
        v1(v1), v2(v2), v3(v3), color(color) {}

    ~Triangle() = default;

    bool operator==(const Triangle& other) const;
    bool operator<(const Triangle& other) const;

    inline uint32_t getV1() const;
    inline uint32_t getV2() const;
```

```
    inline uint32_t getV3() const;
    inline uint32_t getColor() const;

    inline void setV1(uint32_t v1);
    inline void setV2(uint32_t v2);
    inline void setV3(uint32_t v3);
    inline void setColor(uint32_t color);
};

template<> struct std::hash<Point> {
    size_t operator()(const Point& p) const noexcept {
        return hash<float>()(p.getX()) ^
               (hash<float>()(p.getY()) << 1) ^
               (hash<float>()(p.getZ()) << 2);
    }
};

class OBJModel {
private:
    std::list<Point> points;
    std::list<Triangle> triangles;

    int findOrAddVertex(const Point& v);

public:
    OBJModel() = default;
    ~OBJModel() = default;

    bool loadFromFile(const std::string& filename);
    void setColorOBJ(const u_int32_t color);
    void merge(OBJModel& other);
    void sortPoints();
    void sortTriangles();
    void print() const;

    const std::list<Triangle>& getTriangles() const;
    const std::list<Point>& getPoints() const;
};
```

```
#include "obj_worker.hpp"

_Float32 Point::distance(const Point& other) const {
    _Float32 dx = x - other.x;
```

```
    _Float32 dy = y - other.y;
    _Float32 dz = z - other.z;

    return std::sqrt(dx * dx + dy * dy + dz * dz);
}

inline _Float32 Point::getX() const {
    return x;
}

inline _Float32 Point::getY() const {
    return y;
}

inline _Float32 Point::getZ() const {
    return z;
}

inline void Point::setX(_Float32 x) {
    this->x = x;
}

inline void Point::setY(_Float32 y) {
    this->y = y;
}

inline void Point::setZ(_Float32 z) {
    this->z = z;
}

bool Point::operator==(const Point& other) const noexcept {
    return x == other.x && y == other.y && z == other.z;
}

bool Point::operator<(const Point& other) const noexcept {
    if (x != other.x) {
        return x < other.x;
    }

    if (y != other.y) {
        return y < other.y;
    }
```

```
    return z < other.z;
}

_Float32 Point::dot(const Point& other) const {
    return x * other.x + y * other.y + z * other.z;
}

bool Triangle::operator==(const Triangle& other) const {
    return (v1 == other.v1 && v2 == other.v2 && v3 == other.v3);
}

inline uint32_t Triangle::getV1() const {
    return v1;
}

inline uint32_t Triangle::getV2() const {
    return v2;
}

inline uint32_t Triangle::getV3() const {
    return v3;
}

inline uint32_t Triangle::getColor() const {
    return color;
}

inline void Triangle::setV1(uint32_t v1) {
    this->v1 = v1;
}

inline void Triangle::setV2(uint32_t v2) {
    this->v2 = v2;
}

inline void Triangle::setV3(uint32_t v3) {
    this->color = v3;
}

inline void Triangle::setColor(uint32_t color) {
    this->color = color;
}
```

```cpp
bool Triangle::operator<(const Triangle& other) const {
    return (v1 < other.v1) ||
           (v1 == other.v1 && (v2 < other.v2 ||
           (v2 == other.v2 && v3 < other.v3)));
}

bool OBJModel::loadFromFile(const std::string& filename) {
    std::ifstream file(filename);
    if (!file.is_open()) return false;

    points.clear();
    triangles.clear();

    std::string line;
    while (std::getline(file, line)) {
        if (line.substr(0, 2) == "v ") {
            std::istringstream iss(line.substr(2));
            float x, y, z;
            if (iss >> x >> y >> z) {
                points.emplace_back(x, y, z);
            }
        } else if (line.substr(0, 2) == "f ") {
            std::istringstream iss(line.substr(2));
            int v1, v2, v3;

            if (iss >> v1 >> v2 >> v3) {
                triangles.emplace_back(v1-1, v2-1, v3-1, 0);
            }
        }
    }
    return true;
}

int OBJModel::findOrAddVertex(const Point& v) {
    static std::unordered_map<Point, int> vertexMap;
    auto it = vertexMap.find(v);

    if (it != vertexMap.end()) {
        return it->second;
    }

    points.push_back(v);
```

```
    int index = points.size() - 1;
    vertexMap[v] = index;

    return index;
}

void OBJModel::setColorOBJ(uint32_t color) {
    for (auto& t : triangles) {
        t.setColor(color);
    }
}


void OBJModel::merge(OBJModel& other) {
    std::vector<int> indexMap;
    for (const auto& v : other.points) {
        indexMap.push_back(findOrAddVertex(v));
    }

    for (auto& t : other.triangles) {
        triangles.emplace_back(
            indexMap[t.getV1()],
            indexMap[t.getV2()],
            indexMap[t.getV3()],
            t.getColor()
        );
    }
}

void OBJModel::print() const {
    std::cout << "Vertices (" << points.size() << "):\n";

    for (const auto& p : points) {
        std::cout << "v " << p.getX() << " " << p.getY() << " " <<
p.getZ() << "\n";
    }

    std::cout << "\nTriangles (" << triangles.size() << "):\n";

    for (const auto& t : triangles) {
        std::cout << "f " << t.getV1() + 1 << " " << t.getV2() + 1 <<
" " << t.getV3() + 1
```

```
                << " Color: 0x" << std::hex << t.getColor() <<
std::dec << "\n";
    }
}

const std::list<Point>& OBJModel::getPoints() const {
    return points;
}

const std::list<Triangle>& OBJModel::getTriangles() const {
    return triangles;
}

void OBJModel::sortPoints() {
    points.sort();
}

void OBJModel::sortTriangles() {
    triangles.sort();
}
```

```
#include <gtest/gtest.h>
#include <fstream>
#include "obj_worker.hpp"

class OBJModelTest : public ::testing::Test {
protected:
    void SetUp() override {
        createTestFiles();
    }

    void TearDown() override {
        removeTestFiles();
    }

    void createTestFiles() {
        std::ofstream("valid.obj") << "v 0 0 0\nv 1 0 0\nv 0 1 0\nf 1 2
3\nf 2 3 1\n";
        std::ofstream("empty.obj");
    }

    void removeTestFiles() {
        remove("valid.obj");
```

```
        remove("empty.obj");
        remove("output.obj");
    }
};

TEST(PointTest, Equality) {
    Point p1(1.0f, 2.0f, 3.0f);
    Point p2(1.0f, 2.0f, 3.0f);
    Point p3(1.1f, 2.0f, 3.0f);
    ASSERT_TRUE(p1 == p2);
    ASSERT_FALSE(p1 == p3);
}

TEST(PointTest, DistanceCalculation) {
    Point p1(0, 0, 0);
    Point p2(3, 4, 0);
    EXPECT_FLOAT_EQ(p1.distance(p2), 5.0f);
}

TEST(TriangleTest, ColorOperations) {
    Triangle t(1, 2, 3);
    t.setColor(0xFF00FF);
    EXPECT_EQ(t.getColor(), 0xFF00FF);
}

TEST_F(OBJModelTest, LoadInvalidFile) {
    OBJModel model;
    EXPECT_FALSE(model.loadFromFile("nonexistent.obj"));
}

TEST_F(OBJModelTest, ColorAssignment) {
    OBJModel model;
    model.loadFromFile("valid.obj");
    model.setColorOBJ(0x00FF00);

    for (const auto& t : model.getTriangles()) {
        EXPECT_EQ(t.getColor(), 0x00FF00);
    }
}

TEST_F(OBJModelTest, Sorting) {
    OBJModel model;
    model.loadFromFile("valid.obj");
```

```
    model.merge(model);

    model.sortPoints();
    const auto& verts = model.getPoints();
    EXPECT_TRUE(std::is_sorted(verts.begin(), verts.end()));

    model.sortTriangles();
    const auto& tris = model.getTriangles();
    EXPECT_TRUE(std::is_sorted(tris.begin(), tris.end()));
}

int main(int argc, char** argv) {
    ::testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```

```
❯ ./obj_test
[==========] Running 6 tests from 3 test suites.
[----------] Global test environment set-up.
[----------] 2 tests from PointTest
[ RUN      ] PointTest.Equality
[       OK ] PointTest.Equality (0 ms)
[ RUN      ] PointTest.DistanceCalculation
[       OK ] PointTest.DistanceCalculation (0 ms)
[----------] 2 tests from PointTest (0 ms total)

[----------] 1 test from TriangleTest
[ RUN      ] TriangleTest.ColorOperations
[       OK ] TriangleTest.ColorOperations (0 ms)
[----------] 1 test from TriangleTest (0 ms total)

[----------] 3 tests from OBJModelTest
[ RUN      ] OBJModelTest.LoadInvalidFile
[       OK ] OBJModelTest.LoadInvalidFile (7 ms)
[ RUN      ] OBJModelTest.ColorAssignment
[       OK ] OBJModelTest.ColorAssignment (0 ms)
[ RUN      ] OBJModelTest.Sorting
```

```
CXX = g++
CXXFLAGS = -std=c++17 -Wall -Wextra -pthread
GTEST = -lgtest -lgtest_main
INCLUDE = -I.
```

```
SRC = obj_worker.cpp
TEST_SRC = tests.cpp
OBJ = $(SRC:.cpp=.o)
TEST_OBJ = $(TEST_SRC:.cpp=.o)
TARGET = obj_test

all: $(TARGET)

$(TARGET): $(OBJ) $(TEST_OBJ)
	$(CXX) $(CXXFLAGS) $(INCLUDE) $^ -o $@ $(GTEST)

%.o: %.cpp
	$(CXX) $(CXXFLAGS) $(INCLUDE) -c $< -o $@

test: $(TARGET)
	./$(TARGET)

clean:
	rm -f $(OBJ) $(TEST_OBJ) $(TARGET)

.PHONY: all test clean
```

Вывод: познакомились со стандартной библиотекой шаблонов в C++; получили навыки использования классов контейнеров, итераторов, алгоритмов.