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 <vector>
#include <unordered_map>
#include <string>
#include <fstream>
#include <sstream>
#include <unordered_set>
#include <cmath>
#include <algorithm>
#include <cstdint>

class Point {
private:
    float x, y, z;

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

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

```
    Point& operator=(const Point&) = default;

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

    struct Hash {
        size_t operator()(const Point& p) const {
            return std::hash<float>()(p.x) ^
                   (std::hash<float>()(p.y) << 1) ^
                   (std::hash<float>()(p.z) << 2);
        }
    };
};

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) {}

    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 v);
    inline void setV2(uint32_t v);
    inline void setV3(uint32_t v);
    inline void setColor(uint32_t c);

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

};

class OBJModel {
private:
```

```
    std::vector<Point> points;
    std::vector<Triangle> triangles;
    std::unordered_map<Point, size_t, Point::Hash> vertexMap;

    size_t findOrAddVertex(const Point& v);

public:
    bool loadFromFile(const std::string& filename);

    void removeUnusedVertices();

    void merge(const OBJModel& other);

    void setColorOBJ(uint32_t color);

    void sortPoints();

    void sortTriangles();

    const std::vector<Point>& getPoints() const;

    const std::vector<Triangle>& getTriangles() const;

    void saveToFile(const std::string& filename) const;
};
```

```
#include "obj_worker.hpp"


bool Point::operator==(const Point& other) const {
    return std::abs(x - other.x) < 1e-6 &&
           std::abs(y - other.y) < 1e-6 &&
           std::abs(z - other.z) < 1e-6;
}

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

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

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


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 v) {
   v1 = v;
}

inline void Triangle::setV2(uint32_t v) {
   v2 = v;
}

inline void Triangle::setV3(uint32_t v) {
   v3 = v;
}

inline void Triangle::setColor(uint32_t c) {
   color = c;
}

inline bool Triangle::operator<(const Triangle& other) const {
   return std::tie(v1, v2, v3) < std::tie(other.v1, other.v2,
other.v3);
}
```

```cpp
size_t OBJModel::findOrAddVertex(const Point& v) {
    auto it = vertexMap.find(v);
    if (it != vertexMap.end()) return it->second;

    points.push_back(v);
    vertexMap[v] = points.size() - 1;
    return points.size() - 1;
}


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

    points.clear();
    triangles.clear();
    vertexMap.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));
            std::vector<uint32_t> face;
            std::string token;

            while (iss >> token) {
                size_t pos = token.find('/');
                if (pos != std::string::npos) token = token.substr(0,
pos);

                try {
                    int idx = std::stoi(token);
                    if (idx < 0) idx = points.size() + idx + 1;
                    if (idx > 0)
face.push_back(static_cast<uint32_t>(idx - 1));
                }
```

```
                catch (...) {}
            }

            if (face.size() >= 3) {
                for (size_t i = 2; i < face.size(); ++i) {
                    triangles.emplace_back(face[0], face[i-1], face[i]);
                }
            }
        }
    }
    return true;
}

void OBJModel::removeUnusedVertices() {
    std::unordered_set<size_t> used;
    for(const auto& t : triangles) {
        if(t.getV1() < points.size()) {
            used.insert(t.getV1());
        }

        if(t.getV2() < points.size()) {
            used.insert(t.getV2());
        }

        if(t.getV3() < points.size()) {
            used.insert(t.getV3());
        }
    }

    std::vector<Point> newPoints;
    std::vector<size_t> indexMap(points.size());
    size_t newIndex = 0;

    for (size_t i = 0; i < points.size(); ++i) {
        if (used.count(i)) {
            indexMap[i] = newIndex++;
            newPoints.push_back(points[i]);
        }
    }

    for (auto& t : triangles) {
        t.setV1(indexMap[t.getV1()]);
        t.setV2(indexMap[t.getV2()]);
```

```
        t.setV3(indexMap[t.getV3()]);
    }

    points = std::move(newPoints);
    vertexMap.clear();
    for (size_t i = 0; i < points.size(); ++i) {
        vertexMap[points[i]] = i;
    }
}

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

void OBJModel::merge(const OBJModel& other) {
    std::vector<size_t> indexMap(other.points.size());

    for(size_t i = 0; i < other.points.size(); ++i) {
        indexMap[i] = findOrAddVertex(other.points[i]);
    }

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

void OBJModel::sortPoints() {
    std::sort(points.begin(), points.end(),
        [](const Point& a, const Point& b) {
            float ax = a.getX(), ay = a.getY(), az = a.getZ();
            float bx = b.getX(), by = b.getY(), bz = b.getZ();
            return std::tie(ax, ay, az) < std::tie(bx, by, bz);
        });
}

void OBJModel::sortTriangles() {
    std::sort(triangles.begin(), triangles.end());
}
```

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

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

void OBJModel::saveToFile(const std::string& filename) const {
    std::ofstream out(filename);
    for (const auto& p : points) {
        out << "v " << p.getX() << " " << p.getY() << " " << p.getZ() <<
"\n";
    }
    for (const auto& t : triangles) {
        out << "f " << t.getV1()+1 << " " << t.getV2()+1 << " " <<
t.getV3()+1 << "\n";
    }
}
```

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

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

    void TearDown() override {
        remove("complex_input.obj");
        remove("processed.obj");
    }

    void createComplexTestFile() {
        std::ofstream f("complex_input.obj");
        f << "# Complex model with normals\n";
        for(int i = 0; i < 1000; ++i) {
            f << "v " << i << " " << i+1 << " " << i+2 << "\n";
            f << "vn " << 0 << " " << 0 << " " << 1 << "\n";
```

```
        }
        for(int i = 1; i <= 997; ++i) {
            f << "f " << i << "//" << i << " "
              << i+1 << "//" << i+1 << " "
              << i+2 << "//" << i+2 << "\n";
        }
    }
};

TEST_F(OBJModelTest, CleanProcessing) {
    OBJModel model;

    ASSERT_TRUE(model.loadFromFile("complex_input.obj"));

    ASSERT_EQ(model.getPoints().size(), 1000);
    ASSERT_EQ(model.getTriangles().size(), 997 * 1);

    model.removeUnusedVertices();
    model.sortPoints();
    model.sortTriangles();

    model.saveToFile("processed.obj");

    std::ifstream out("processed.obj");
    std::string line;
    int vertex_count = 0;
    int normal_count = 0;
    int face_count = 0;

    while(std::getline(out, line)) {
        if(line.substr(0, 2) == "v ") vertex_count++;
        if(line.substr(0, 3) == "vn ") normal_count++;
        if(line.substr(0, 2) == "f ") face_count++;
    }

    EXPECT_EQ(normal_count, 0) << "All normals should be removed";
    EXPECT_EQ(vertex_count, model.getPoints().size());
    EXPECT_EQ(face_count, model.getTriangles().size());

    auto& verts = model.getPoints();
    for(size_t i = 1; i < verts.size(); ++i) {
        ASSERT_FALSE(verts[i-1] == verts[i])
            << "Duplicate vertices at positions "
```

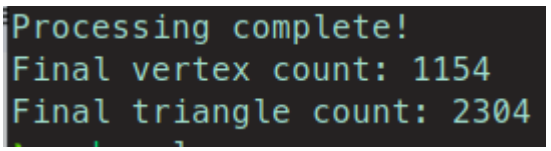
```
            << i-1 << " and " << i;
    }
}

TEST_F(OBJModelTest, ColorPersistence) {
    OBJModel model;
    model.loadFromFile("complex_input.obj");

    model.setColorOBJ(0xFFA500);

    for(const auto& t : model.getTriangles()) {
        ASSERT_EQ(t.getColor(), 0xFFA500);
    }

    model.merge(model);
    model.removeUnusedVertices();

    for(const auto& t : model.getTriangles()) {
        EXPECT_EQ(t.getColor(), 0xFFA500);
    }
}

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
```

```
#include "obj_worker.hpp"
#include <iostream>

int main() {
    OBJModel model;

    if (!model.loadFromFile("triangle.obj")) {
        std::cerr << "Failed to load model!" << std::endl;
        return 1;
    }

    model.setColorOBJ(0xFFA500);
    model.removeUnusedVertices();
    model.sortPoints();
    model.sortTriangles();

    OBJModel additional;
    if (additional.loadFromFile("additional.obj")) {
        additional.setColorOBJ(0x00FF00);
        model.merge(additional);
    }

    model.saveToFile("processed_model.obj");

    std::cout << "Processing complete!\n";
    std::cout << "Final vertex count: " << model.getPoints().size() <<
"\n";
    std::cout << "Final triangle count: " << model.getTriangles().size()
<< "\n";

    return 0;
}
```



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