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 ВТ-231

Масленников Д. А.

Проверили:  
Буханов Д. Г.

Белгород 2025

**Цель работы:** знакомство со стандартной библиотекой шаблонов в С++; получение навыков использования классов контейнеров, итераторов, алгоритмов.  
**Задание**: 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.  
#include <iostream>

#include <fstream>

#include <list>

#include <string>

#include <sstream>

#include <stdexcept>

**class** **Point3D** {

**private:**

float x, y, z;

**public:**

Point3D(float x = 0, float y = 0, float z = 0) : x(x), y(y), z(z) {}

float getX() **const** { **return** x; }

float getY() **const** { **return** y; }

float getZ() **const** { **return** z; }

void setX(float val) { x = val; }

void setY(float val) { y = val; }

void setZ(float val) { z = val; }

};

**class** **Triangle** {

**private:**

int v1, v2, v3;

**public:**

Triangle(int v1 = 0, int v2 = 0, int v3 = 0) : v1(v1), v2(v2), v3(v3) {}

int getV1() **const** { **return** v1; }

int getV2() **const** { **return** v2; }

int getV3() **const** { **return** v3; }

void setV1(int val) { v1 = val; }

void setV2(int val) { v2 = val; }

void setV3(int val) { v3 = val; }

};

**class** **OBJModel** {

**private:**

std::list<Point3D> vertices;

std::list<Triangle> faces;

**public:**

bool loadFromFile(**const** std::string& filename) {

std::ifstream file(filename);

**if** (!file.is\_open()) {

**throw** std::runtime\_error("Failed to open file: " + filename);

}

std::string line;

**while** (std::getline(file, line)) {

std::istringstream iss(line);

std::string prefix;

iss >> prefix;

**if** (prefix == "v") {

float x, y, z;

**if** (!(iss >> x >> y >> z)) {

**throw** std::runtime\_error("Invalid vertex data in file: " + filename);

}

vertices.emplace\_back(x, y, z);

}

**else** **if** (prefix == "f") {

int v1, v2, v3;

**if** (!(iss >> v1 >> v2 >> v3)) {

**throw** std::runtime\_error("Invalid face data in file: " + filename);

}

**if** (v1 < 1 || v2 < 1 || v3 < 1 ||

v1 > vertices.size() ||

v2 > vertices.size() ||

v3 > vertices.size()) {

**throw** std::out\_of\_range("Vertex index out of range in face data");

}

faces.emplace\_back(v1-1, v2-1, v3-1);

}

}

file.close();

**return** true;

}

**friend** OBJModel **operator**+(**const** OBJModel& a1, **const** OBJModel& a2) {

OBJModel result;

result.vertices = a1.vertices;

result.faces = a1.faces;

size\_t baseIdx = a1.vertices.size();

result.vertices.insert(result.vertices.end(),

a2.vertices.begin(),

a2.vertices.end());

**for** (**const** **auto**& f : a2.faces) {

Triangle newFace = f;

newFace.setV1(f.getV1() + baseIdx);

newFace.setV2(f.getV2() + baseIdx);

newFace.setV3(f.getV3() + baseIdx);

result.faces.push\_back(newFace);

}

**return** result;

}

void saveToFile(**const** std::string& filename) **const** {

std::ofstream file(filename);

**if** (!file.is\_open()) {

**throw** std::runtime\_error("Failed to create file: " + filename);

}

**for** (**const** **auto**& vertex : vertices) {

file << "v " << vertex.getX() << " " << vertex.getY() << " " << vertex.getZ() << "**\n**";

}

**for** (**const** **auto**& face : faces) {

file << "f " << face.getV1()+1 << " " << face.getV2()+1 << " " << face.getV3()+1 << "**\n**";

}

file.close();

}

size\_t getVertexCount() **const** { **return** vertices.size(); }

size\_t getTriangleCount() **const** { **return** faces.size(); }

};

int main() {

try {

OBJModel model1, model2, result;

model1.loadFromFile("model1.obj");

model2.loadFromFile("model2.obj");

result = model1 + model2;

result.saveToFile("result.obj");

std::cout << "Модель корректно обьединена : "

<< result.getVertexCount()

<< ", triangles: " << result.getTriangleCount() << std::endl;

}

**catch** (**const** std::exception& e) {

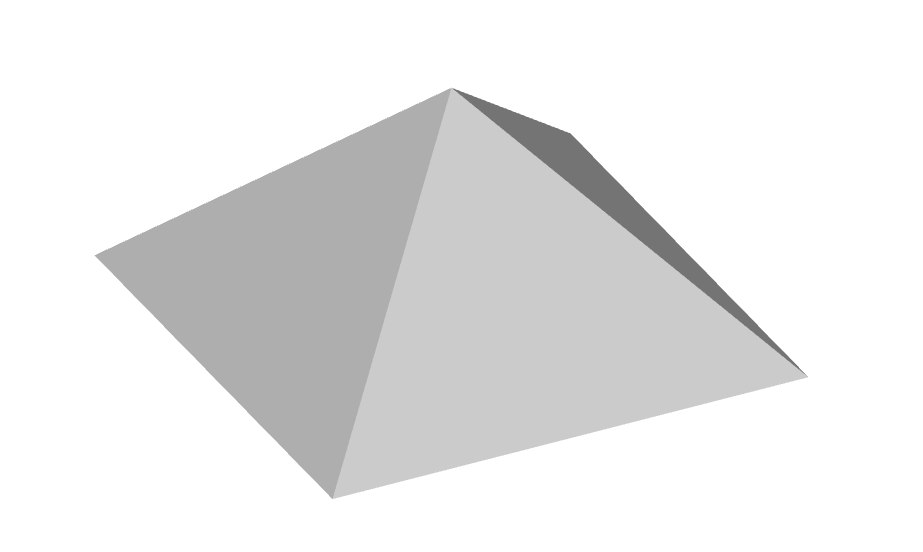
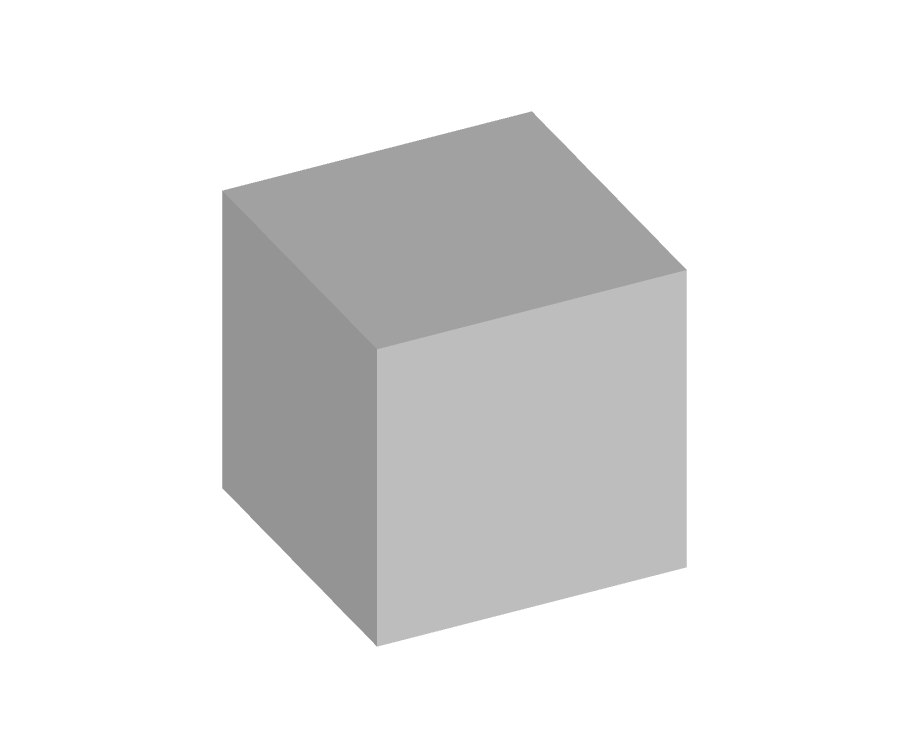
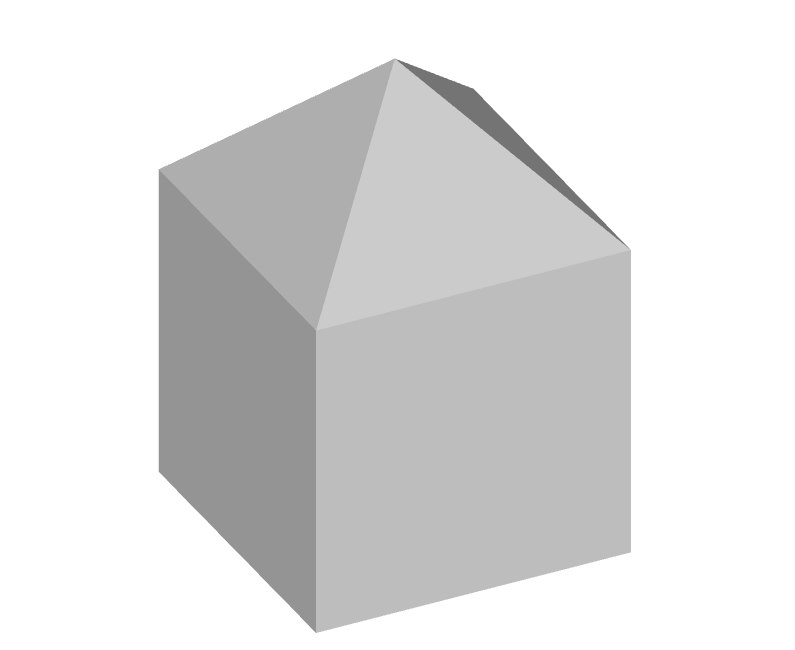
std::cerr << "Error: " << e.what() << std::endl;

**return** 1;

}

**return** 0;

}

  
  
  
  
  
  
  
  
  
  
model1 model2  
  
  
  
  
  
  
  
  
  
  
  
result