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 №8**

по дисциплине: ООП

тема: «Создание шаблонов в C++»

Выполнил: студент группы ВТ-231

Масленников Д. А.

Проверили:  
Буханов Д. Г.

Белгород 2025

**Цель работы:** Получение теоретических знаний о шаблонах классов в С++. Получение практических навыков по созданию классов-шаблонов С++.

Задания к лабораторной работе

1. Изучить теоретические сведения о шаблонах классов в С++.
2. Разработать программу в соответствии с заданным вариантом задания.

**Задание 1**  
 **Вариант 4 -** Множество

Реализовать шаблон класса в соответствии с указанным вариантом. Предусмотреть необходимые методы для работы со структурой данных, указанной в варианте. Предусмотреть исключительные ситуации, которые могут возникнуть в процессе работы.  
  
#include <iostream>

#include <stdexcept>

#include <algorithm>

using namespace std;

template<typename T, size\_t N = 100>

class Set {

private:

T data[N];

size\_t count;

public:

Set() : count(0) {}

void insert(const T& value) {

if (count >= N) {

throw invalid\_argument("Множество заполнено");

}

if (contains(value)) {

throw invalid\_argument("Элемент уже существует");

}

data[count++] = value;

}

void remove(const T& value) {

for (size\_t i = 0; i < count; ++i) {

if (data[i] == value) {

swap(data[i], data[count - 1]);

--count;

return;

}

}

throw invalid\_argument("Элемент отсутствует в множестве");

}

void removeInd(const T& ind) {

for (size\_t i = 0; i < count; ++i){

if (i == ind){

swap(data[i], data[count - 1]);

--count;

return;

}

}

throw invalid\_argument("Индекс отсутствует в множестве");

}

bool contains(const T& value) const {

for (size\_t i = 0; i < count; ++i) {

if (data[i] == value) {

return true;

}

}

return false;

}

void clear() {

count = 0;

}

size\_t size() const {

return count;

}

bool isEmpty() const{

return count == 0;

}

void print() const{

cout << "{";

for (size\_t i = 0; i < count; i++){

cout << data[i];

if (i != count - 1) cout << ", ";

}

cout << "}" << endl;

}

Set<T> copy() const{

Set<T> newSet;

size\_t i = 0;

while(i < count){

newSet.insert(data[i]);

i++;

}

return newSet;

}

};

int main(){

Set<int> mySet;

mySet.insert(1);

mySet.insert(2);

mySet.insert(3);

mySet.insert(4);

mySet.removeInd(0);

mySet.print();

}

Задание 2.

На основе разработанного шаблона решить прикладную задачу в соответствии с выбранным вариантом.

4. Дан файл, в котором храниться структуры следующего типа: кафедра->группа->студент->сессия->предметы->оценки, предметы могут быть разных типов, от этого зависит тип оценки (диф. зачет, зачет, экзамен, прослушал). Каждая структура описывает одного студента. Построить дерево оценок всех студентов одной кафедры, группы. Реализовать сортировку внутри группы, по оценкам студентов, и сортировку предметов по типу внутри студента.  
  
#include <iostream>

#include <string>

#include <algorithm>

#include <iomanip>

#include <fstream>

#include <stdexcept>

#include <sstream>

#include <vector>

#include "set.hpp"

class SetException : public std::runtime\_error {

public:

SetException(const std::string& msg) : std::runtime\_error(msg) {}

};

class Subject {

private:

std::string grade;

std::string type;

std::string name;

public:

Subject() : grade(""), type(""), name("") {}

Subject(std::string g, std::string t, std::string n)

: grade(g), type(t), name(n) {}

int getGrade() const {

if (grade == "отл" || grade == "зачет" || grade == "5") {

return 5;

} else if (grade == "хор" || grade == "4") {

return 4;

} else if (grade == "уд" || grade == "3") {

return 3;

} else {

return 2;

}

}

const std::string& getName() const {

return name;

}

const std::string& getType() const {

return type;

}

const std::string& getGradeString() const {

return grade;

}

bool operator==(const Subject& other) const {

return (name == other.name) && (type == other.type) && (grade == other.grade);

}

bool operator!=(const Subject& other) const {

return !(\*this == other);

}

bool operator<(const Subject& other) const {

if (type < other.type) {

return true;

} else if (type == other.type) {

return name < other.name;

} else {

return false;

}

}

friend std::ostream& operator<<(std::ostream& os, const Subject& subject) {

os << subject.name << " (" << subject.type << "): " << subject.grade;

return os;

}

};

class Student {

private:

std::string name;

Set<Subject> grades;

public:

Student() : name("") {}

Student(std::string n) : name(n) {}

void addGrade(const Subject& s) {

grades.add(s);

}

std::string getName() const {

return name;

}

double getAvgGrade() const {

double sum = 0;

int size = grades.size();

if (size == 0) {

return 0.0;

}

for (size\_t i = 0; i < size; ++i) {

sum += grades[i].getGrade();

}

return sum / size;

}

void print() const {

std::cout << "- cтудент: " << name << "\n";

std::cout << "- cредний балл: " << std::fixed << std::setprecision(2) <<

getAvgGrade() << "\n";

for (size\_t i = 0; i < grades.size(); ++i) {

std::cout << " --- " << grades[i] << "\n";

}

}

bool operator==(const Student& other) const {

return name == other.name;

}

bool operator!=(const Student& other) const {

return !(\*this == other);

}

bool operator<(const Student& other) const {

return name < other.name;

}

friend std::ostream& operator<<(std::ostream& os, const Student& student) {

student.print();

return os;

}

};

class Group {

private:

std::string name;

Set<Student> students;

public:

Group(std::string n) : name(n) {}

void addStudent(const Student& s) {

students.add(s);

}

std::string getName() const {

return name;

}

Set<Student>& getStudents() {

return students;

}

void SortByGrades() {

bool swapped;

size\_t n = students.size();

do {

swapped = false;

for (size\_t i = 1; i < n; i++) {

if (students[i-1].getAvgGrade() < students[i].getAvgGrade()) {

std::swap(students[i-1], students[i]);

swapped = true;

}

}

n--;

} while (swapped);

}

void print() const {

std::cout << "группа: " << name << "\ncписок студентов:\n";

for (size\_t i = 0; i < students.size(); ++i) {

std::cout << students[i] << "\n";

}

}

friend std::ostream& operator<<(std::ostream& os, const Group& group) {

group.print();

return os;

}

};

class GroupReader {

public:

static std::vector<Group> readFromFile(const std::string& filename) {

std::ifstream file(filename);

if (!file) throw std::runtime\_error("Cannot open file: " + filename);

std::vector<Group> groups;

std::string line;

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

auto parts = split(line, '>');

if (parts.size() != 6) continue;

Group& group = findOrCreateGroup(groups, parts[1]);

Student& student = findOrCreateStudent(group, parts[2]);

student.addGrade(Subject(parts[5], parts[4], parts[3]));

}

return groups;

}

private:

static std::vector<std::string> split(std::string& s, char delim) {

std::vector<std::string> res;

std::stringstream ss(s);

std::string item;

while (std::getline(ss, item, delim)) {

res.push\_back(item);

}

return res;

}

static Group& findOrCreateGroup(std::vector<Group>& groups, const std::string& name) {

for (auto& group : groups) {

if (group.getName() == name) return group;

}

groups.emplace\_back(name);

return groups.back();

}

static Student& findOrCreateStudent(Group& group, const std::string& name) {

for (size\_t i = 0; i < group.getStudents().size(); i++) {

if (group.getStudents()[i].getName() == name) {

return group.getStudents()[i];

}

}

group.addStudent(Student(name));

for (size\_t i = 0; i < group.getStudents().size(); i++) {

if (group.getStudents()[i].getName() == name) {

return group.getStudents()[i];

}

}

throw std::runtime\_error("Failed to create student: " + name);

}

};

int main() {

try {

auto groups = GroupReader::readFromFile("students.txt");

for (auto& group : groups) {

group.SortByGrades();

std::cout << group << "\n";

}

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

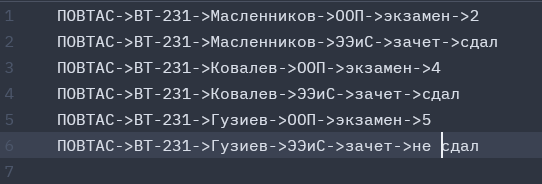
std::cerr << "ошибка: " << e.what() << "\n";

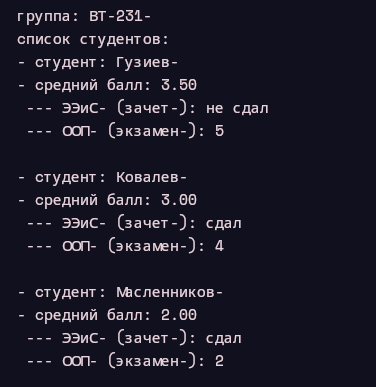
return 1;

}

return 0;

}



исходный файл  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
вывод программы