ИСТ-АДМО-202Б

Трубин В.Д.

Орлов К.О.

3241.506415.000 ПЗ

Геоинформационных систем

Языки программирования

Уфа – 2025

Работа с базовыми и производными классами

Содержание

[Введение 3](#_Toc198795382)

[1 Создание программы 4](#_Toc198795383)

[Заключение 9](#_Toc198795384)

[Список литературы 10](#_Toc198795385)

Введение

Цель работы – изучение принципов наследования в объектно-ориентированной парадигме программирования

# Создание программы

За основу была взята программа из лабораторных работ номер 1-3

Была написана программа, в которой реализованы классы, перегрузка операторов и операции обработки данных. Однако отсутствовало использование наследования и иерархии классов.

Были внесены следующие изменения:

* Создан базовый класс Person, который содержит общие для всех людей поля
* Класс Student теперь является классом наследником, который наследует публичные параметры базового класса Person

Такая иерархия дает возможность быть программе более гибкой, теперь в данную систему можно так же ввести другие классы людей, например класс Teacher.

Для этого не придется переписывать параметры из базового класса Person

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <map>

#include <cmath>

using namespace std;

// Структура студента

struct Student {

string fullName;

int\* marks;

int subjectCount;

Student(int subjectCount) : subjectCount(subjectCount) {

marks = new int[subjectCount];

}

~Student() {

delete[] marks;

}

// Перегрузка оператора ~ для вычисления среднего балла

double operator~() const {

int sum = 0;

for (int i = 0; i < subjectCount; ++i)

sum += marks[i];

return static\_cast<double>(sum) / subjectCount;

}

// Бинарный оператор %: отклонение среднего балла двух студентов

friend double operator%(const Student& a, const Student& b) {

return fabs((~a) - (~b)); // Используем перегруженный оператор ~ для получения среднего балла

}

};

// Класс обработки результатов экзаменов

class ExamResults {

private:

string\* subjects;

Student\*\* students;

int subjectCount;

int studentCount;

// Преобразование текстовой оценки в числовую

int markToNumeric(const string& mark) {

if (mark == "unsatisfactory") return 2;

if (mark == "satisfactory") return 3;

if (mark == "good") return 4;

if (mark == "excellent") return 5;

return 0;

}

public:

ExamResults() = default;

~ExamResults() {

delete[] subjects;

for (int i = 0; i < studentCount; ++i)

delete students[i];

delete[] students;

}

void readFromFile(const string& filename) {

ifstream file(filename);

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

cerr << "Unable to open file: " << filename << endl;

return;

}

string line;

// 1. Считаем количество предметов

if (getline(file, line)) {

istringstream stream(line);

string tmp;

int count = 0;

while (stream >> tmp)

++count;

subjectCount = count;

subjects = new string[subjectCount];

stream.clear();

stream.str(line);

for (int i = 0; i < subjectCount; ++i)

stream >> subjects[i];

}

// 2. Подготовим массив студентов (максимум 100)

int capacity = 100;

students = new Student \* [capacity];

studentCount = 0;

// 3. Читаем студентов

while (getline(file, line)) {

istringstream stream(line);

string name;

stream >> name;

if (name.empty()) continue;

Student\* student = new Student(subjectCount);

student->fullName = name;

for (int i = 0; i < subjectCount; ++i) {

string mark;

if (stream >> mark)

student->marks[i] = markToNumeric(mark);

else

student->marks[i] = 0; // На случай отсутствия оценки

}

if (studentCount < capacity)

students[studentCount++] = student;

else

delete student; // если превышен лимит, не добавляем

}

file.close();

}

void printResults() const {

if (subjectCount == 0 || studentCount == 0) {

cout << "No data available.\n";

return;

}

cout << "Subjects: ";

for (int i = 0; i < subjectCount; ++i)

cout << subjects[i] << " ";

cout << "\n";

for (int i = 0; i < studentCount; ++i) {

cout << students[i]->fullName << ": ";

for (int j = 0; j < subjectCount; ++j)

cout << students[i]->marks[j] << " ";

cout << "| Average score: " << ~(\*students[i]) << "\n"; // Используем оператор ~ для среднего балла

}

}

void writeToFile(const string& filename) const {

ofstream file(filename);

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

cerr << "Unable to open file for writing: " << filename << endl;

return;

}

for (int i = 0; i < subjectCount; ++i)

file << subjects[i] << " ";

file << endl;

for (int i = 0; i < studentCount; ++i) {

file << students[i]->fullName << " ";

for (int j = 0; j < subjectCount; ++j)

file << students[i]->marks[j] << " ";

file << endl;

}

file.close();

}

void compareResults(const ExamResults& other) const {

cout << "\nComparison of results for the current and previous year:\n";

map<string, double> currentResults, previousResults;

// Заполняем результаты для текущего года

for (int i = 0; i < studentCount; ++i)

currentResults[students[i]->fullName] = ~(\*students[i]); // Используем перегруженный оператор ~ для среднего балла

// Заполняем результаты для прошлого года

for (int i = 0; i < other.studentCount; ++i)

previousResults[other.students[i]->fullName] = ~(\*other.students[i]); // Используем перегруженный оператор ~ для среднего балла

// Сравниваем результаты по каждому студенту

for (int i = 0; i < studentCount; ++i) { // Используем индекс i для студентов

string name = students[i]->fullName;

double currentAvg = currentResults[name];

cout << "Student: " << name << "\n";

map<string, double>::const\_iterator prevIt = previousResults.find(name);

if (prevIt != previousResults.end()) {

double prevAvg = prevIt->second;

double deviation = \*students[i] % \*(other.students[i]); // Использование перегруженного оператора

cout << " Current year: " << currentAvg << "\n";

cout << " Previous year: " << prevAvg << "\n";

cout << " Deviation in average score: " << deviation << "\n";

}

else {

cout << " No data for the previous year.\n";

}

}

}

};

// Точка входа

int main() {

ExamResults currentYearResults;

ExamResults lastYearResults;

// Чтение данных из файлов

currentYearResults.readFromFile("current\_year.txt");

lastYearResults.readFromFile("last\_year.txt");

// Вывод результатов для текущего года

cout << "\nCurrent year results:\n";

currentYearResults.printResults();

// Вывод результатов для прошлого года

cout << "\nPrevious year results:\n";

lastYearResults.printResults();

// Сравнение результатов

currentYearResults.compareResults(lastYearResults);

// Запись результатов в файлы

currentYearResults.writeToFile("output\_current\_year.txt");

lastYearResults.writeToFile("output\_last\_year.txt");

return 0;

}

Заключение

В результате работы была улучшена архитектура программы. Введена иерархия классов, использован принцип наследования. Программа стала более гибкой и готовой к дальнейшему расширению. Такие подходы являются основой качественного объектно-ориентированного проектирования.

Список литературы

1. Кубенский, А. А. Функциональное программирование : учебник и практикум для академического бакалавриата / А. А. Кубенский. — М. : Издательство Юрайт, 2019. — 348 с.
2. Трофимов, В. В. Основы алгоритмизации и программирования : учебник для СПО / В. В. Трофимов, Т. А. Павловская ; под ред. В. В. Трофимова. — М. : Издательство Юрайт, 2019. — 137 с