

Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ
Инженерно-экономический факультет
Кафедра экономической информатики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе по курсу:
«Объектно-ориентированное программирование»
на тему:

«РАЗРАБОТКА АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ
ТЕСТИРОВАНИЯ ПО РАЗЛИЧНЫМ ТЕМАМ»

Студент гр. 924401

Мельников М.А

Руководитель ст. преподаватель

Салапура М.Н.

каф. эконом. информ. БГУИР

Минск 2021

СОДЕРЖАНИЕ

Введение.....	5
1 Обзор процесса тестирования, программных аналогов систем тестирования, методов и алгоритмов решения поставленной задачи	6
1.1 Обзор процесса тестирования.....	6
1.2 Обзор программных аналогов систем тестирования.....	8
1.3 Обзор методов и алгоритмов решения поставленной задачи	9
2 Функциональное моделирование на основе стандарта IDEF0	12
3 Структура используемых данных.....	18
4 Разработка и описание диаграммы классов приложения	22
5 Разработка и описание диаграммы вариантов использования приложения	31
6 Схема алгоритма работы всей программы и алгоритма работы двух и более основных методов	33
7 Описание алгоритма запуска приложения, его использования, результаты работы программы, тестирования ошибок	34
Заключение	45
Список использованных источников	47
Приложение А (обязательное) листинг кода.....	48
Приложение Б (обязательное) диаграмма классов	63
Приложение В (обязательное) диаграмма вариантов использования	64
Приложение Г (обязательное) схема алгоритмов	65

ВВЕДЕНИЕ

История развития тестов своими корнями уходит в глубину веков, она связана с измерением различных способностей, знаний, умений и навыков. Эти измерения можно считать предысторией тестов. Уже в середине III тысячелетия до н. э. в Древнем Вавилоне проводились испытания выпускников в школах писцов на знание арифметических действий, распределять рационы, делить имущество и так далее [1].

Первым этапом применения тестов в мировой практике можно считать период с 80-х годов XIX века до 20-х годов XX века. Этот период характеризуется зарождением и становлением тестирования [2].

К концу XIX века практическая проблема исследования индивидуальных различий стала толчком к появлению первых тестов. С 80-х годов XX века по настоящее время начинается четвертый этап в развитии тестирования. Он характеризуется широким использованием компьютеров, математических моделей, средств программирования для обучения и контроля.

Основным недостатком тестирования – является разработка качественного тестового инструментария. Это длительный, трудоемкий и дорогостоящий процесс. Стандартные наборы тестов для большинства дисциплин ещё не разработаны, а разработанные обычно имеют очень низкое качество [3].

Целью данного курсового проекта является автоматизация и повышения качества тестирования, уменьшения времени на разработку вопросов, проверки ответов, вывода результатов для тестируемых. Эти задачи реализуются с помощью разделения ролей, предоставления каждой роли своего базового функционала, который позволяет для первой роли создавать, удалять, редактировать и проверять различные тесты, а для второй – максимально быстро проходить тестирование и выводить результат в удобной для пользователя форме.

1 ОБЗОР ПРОЦЕССА ТЕСТИРОВАНИЯ, ПРОГРАММНЫХ АНАЛОГОВ СИСТЕМ ТЕСТИРОВАНИЯ, МЕТОДОВ И АЛГОРИТМОВ РЕШЕНИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ

1.1 Обзор процесса тестирования

Тестирование – это форма измерения знаний учащихся, основанная на применении педагогических тестов. Включает в себя подготовку качественных тестов, собственно проведение тестирования и последующую обработку результатов, которая даёт оценку обученности тестируемых [4].

Тест – это инструмент оценивания обученности учащихся, состоящий из системы тестовых заданий, стандартизированной процедуры проведения, обработки и анализа результатов.

Тесты можно классифицировать по различным признакам:

- по целям – информационные, диагностические, обучающие, мотивационные, аттестационные;
- по процедуре создания – стандартизованные, не стандартизованные;
- по способу формирования заданий – детерминированные, стохастические, динамические;
- по технологии проведения – бумажные, в том числе бумажные с использованием оптического распознавания, натурные, с использованием специальной аппаратуры, компьютерные;
- по форме заданий – закрытого типа, открытого типа, установление соответствия, упорядочивание последовательности;
- по наличию обратной связи – традиционные и адаптивные [5].

Традиционный тест содержит список вопросов и различные варианты ответов. Каждый вопрос оценивается в определенное количество баллов. Результат традиционного теста зависит от количества вопросов, на которые был дан правильный ответ. Иными словами, традиционный тест – система заданий, предъявляемая в порядке увеличения сложности в одно и то же время, с одинаковой системой оценивания для всех тестируемых [6].

Существуют различные **формы тестовых заданий**, а именно:

1 Задания с выбором ответов. Разделяются на задания с выбором одного правильного или неправильного ответа. Один вопрос и несколько вариантов ответа, где обязательно только один из них является (не)верным.

2 Задания на установления соответствия. Состоит из таблицы, где имеется две колонки: в первой находится либо вопрос, либо часть утверждения, во второй ответ на вопрос или вторая часть утверждения. Вопрос и соответствующий ответ практически всегда находятся в разных строках. Также два вопроса могут иметь один и тот же ответ.

3 Задания с выбором нескольких правильных ответов. Суть задания схожа заданием с выбором ответов, разница лишь в том, что правильных ответов может быть более одного.

4 Упорядочивание последовательности. Примером такого задания может служить расположение в хронологическом порядке исторических событий.

5 Задания с открытым ответом. Ответ на поставленный вопрос тестируемый дает в виде введенного предложения.

Тестирование выполняет две основные взаимосвязанные **функции**: диагностическую и обучающую:

1 Диагностическая функция заключается в выявлении уровня знаний, умений, навыков тестируемого. Это основная и самая очевидная функция тестирования. По объективности, широте и скорости диагностирования, тестирование превосходит все остальные формы педагогического контроля.

2 Обучающая функция тестирования состоит в мотивировании тестируемого к активизации работы по усвоению предоставленного материала. Для усиления обучающей функции тестирования могут быть использованы дополнительные меры стимулирования студентов, такие как: раздача преподавателем примерного перечня вопросов для самостоятельной подготовки, наличие в самом тесте наводящих вопросов и подсказок, совместный разбор результатов теста [7].

Достоинства:

1 Тестирование является более качественным и объективным способом оценивания, его объективность достигается путём стандартизации процедуры проведения, проверки показателей качества заданий и тестов целиком.

2 Тесты – это более объёмный инструмент, поскольку тестирование может включать в себя задания по всем темам курса, в то время как на устный экзамен обычно выносятся 2-4 темы, а на письменный – 3-5. Это позволяет выявить знания учащегося по всему курсу, исключив элемент случайности при вытаскивании билета. При помощи тестирования можно установить уровень знаний учащегося по предмету в целом и по отдельным его разделам.

3 Тестирование более эффективно с экономической точки зрения. Основные затраты при тестировании приходятся на разработку качественного инструментария, то есть имеют разовый характер. Затраты же на проведение теста значительно ниже, чем при письменном или устном контроле. Проведение тестирования и контроль результатов в группе из 30 человек занимает полтора два часа, устный или письменный экзамен – не менее четырёх часов [8].

Недостатки:

1 Данные, получаемые преподавателем в результате тестирования, хотя и включают в себя информацию о пробелах в знаниях по конкретным разделам, но не позволяют судить о причинах этих пробелов.

2 Тест не позволяет проверять и оценивать высокие, продуктивные уровни знаний, связанные с творчеством, то есть вероятностные, абстрактные и методологические знания.

3 Обеспечение объективности и справедливости теста требует принятия специальных мер по обеспечению конфиденциальности тестовых заданий. При повторном применении теста желательно внесение в задания изменений [9].

1.2 Обзор программных аналогов систем тестирования

В качестве программного аналога рассматривается программа для создания тестов и онлайн тестирования INDIGO.

Система тестирования INDIGO – это профессиональный инструмент автоматизации процесса тестирования и обработки результатов, который предназначен для решения широкого спектра задач:

1 Тестирование, контроль знаний учеников и студентов.

2 Определение профессионального уровня сотрудников, оценка персонала (HR).

3 Подготовка к сдаче экзаменов и аттестаций.

4 Психологическое тестирование.

5 Организация и проведение опросов, олимпиад, конкурсов.

Функциональные возможности. Данная система тестирования предоставляет широкий спектр возможностей, однако далее будут перечислены лишь важнейшие:

1 Система тестирования устанавливается на главный компьютер (сервер тестирования) с помощью инсталляционного пакета.

2 Система может работать как на изолированном компьютере, так и в локальной сети или через Интернет.

3 Центр тестирования можно развернуть на Вашем компьютере или в облаке на наших Интернет-серверах.

4 Все данные хранятся централизованно в базе данных системы.

5 Администраторы работают через программу клиент.

6 Одновременно могут работать сколько угодно администраторов с разных компьютеров.

7 Система имеет многоязычный пользовательский web-интерфейс и полностью поддерживает символы всех языков (Unicode).

Функции администратора: создание и редактирование тестов; управление базой пользователей; назначение тестов пользователям; проведение тестирования; просмотр результатов; построение отчетов и анализ статистики.

Функции пользователя: регистрация и авторизация; выбор теста; прохождение тестирования; просмотр результатов и ошибок.

Доступно 578 бесплатных тестов (на 66400 вопросов с пояснениями) на различные темы, начиная от психологических тестов и заканчивая ЕГЭ.

Функции управления тестами: добавление, редактирование, удаление; перемещение между группами; поиск; копирование; блокировка, разблокировка; защита на редактирование паролем; разграничение прав доступа между администраторами.

Функции управления пользователями: добавление, редактирование, удаление, перемещение между группами; поиск; блокировка, разблокировка; просмотр журнала событий; разграничение прав доступа между администраторами.

Функции, связанные с результатами: сортировка записей; многоуровневая группировка записей (по выбранным столбцам); фильтрация (выборку) записей по сложным условиям; экспорт всей таблицы или выделенных записей в файл (форматы Excel, HTML, XML, TXT); поиск записей в таблице; настройку выводимых столбцов.

Общие функции: просмотр протокола тестирования, который включает:

- общую информацию (о пользователе, тестировании и результатах);
- подробную информацию в виде образа теста, который проходил пользователь с учетом всех перемешиваний и случайных выборок, а также ответы пользователя на каждый вопрос;
- создание отчетов (с возможностью печати или экспорта в Word):
 - отчет по результату;
 - отчет по пользователю;
 - общий отчет по выборке результатов [10].

1.3 Обзор методов и алгоритмов решения поставленной задачи

Целью данного курсового проекта является автоматизация и повышения качества тестирования, уменьшения времени на разработку вопросов, проверки ответов, вывода результатов для тестируемых.

Задачи, которые необходимо выполнить в ходе выполнения курсового проекта:

- исследовать механизм работы системы тестирования по различным темам;

- ознакомиться с уже существующими программами для реализации системы тестирования по различным темам;

- проанализировать слабые стороны данного механизма;
- разработать функциональную модель основного процесса;
- разработать программный продукт;
- провести тестирование разработанного программного продукта;
- выполнить отладку ошибок, обнаруженных в процессе.

Система тестирования спроектирована таким образом, что поддерживает разделение ролей, а именно: студент и преподаватель, каждый из которых представляет класс (Student, Teacher) с собственным набором методов. Основным используемым объектом в системе является тест, также представленный отдельным классом (Test), который, в свою очередь, содержит список объектов класса Question. Для хранения как пользователей (студентов и преподавателей) и тестов используется отдельный пользовательский класс (Database), который перед предоставлением интерфейса считывает необходимую информацию из файлов, а после завершения пользовательского сеанса записывает измененную информацию обратно в файлы.

Функционал студента представлен такими действиями как:

- решение тестов;
- просмотр доступных тестов;
- просмотра личной информации;
- просмотра решенных тестов кратко и отдельно каждый подробно;
- вывод доступных и уже решенных тестов в отсортированном виде.

Перед предоставлением списка доступных тестов класс Database производит фильтрацию тестов по личным характеристикам студента, а именно по его группе и перечню изучаемых дисциплин.

Функционал преподавателя представлен следующими возможностями:

- создание, удаление, редактирование тестов;
- просмотр личной информации;
- просмотр доступных тестов;
- просмотр всех студентов, в которых он преподает;
- просмотр решенных тестов конкретного студента;
- вывод на экран студентов и доступных тестов в отсортированном виде.

Сортировка производится с помощью базовых методов сортировки контейнеров STL, для которых необходимо написать собственный компаратор. Поиск по контейнерам, хранящим информацию о тестах,

студентах и преподавателях ведется в основном линейно, так как большая часть представлен контейнером `list`. Фильтрация производится на основе сравнения характеристик, которые должны совпадать как у фильтрованного объекта, так и у объекта, которому вышеназванный необходим для взаимодействия.

В процессе работы программы под хранение данных в классе `Database` динамически выделяется и очищается память. Для оптимизации работы большинства функций в качестве параметров передаются не сами объекты, а указатели, хранящие адрес на динамически выделенный участок памяти под объект.

2 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ НА ОСНОВЕ СТАНДАРТА IDEF0

На рисунке 2.1 представлена контекстная диаграмма верхнего уровня с функциональной моделью «Автоматизированная система тестирования по различным темам», а также определены потоки входных и выходных данных, механизмы ограничения и управления данными.

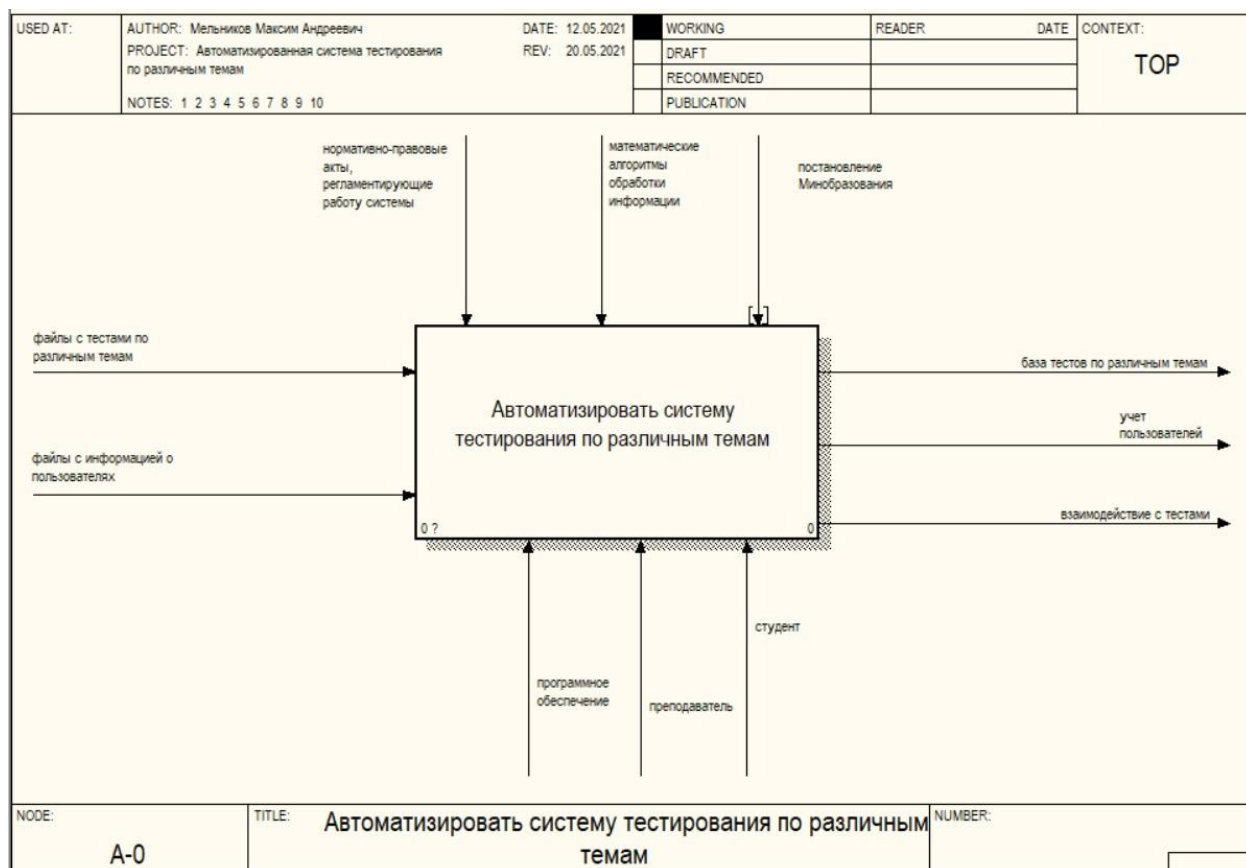


Рисунок 2.1 – контекстная диаграмма верхнего уровня

Далее идет декомпозиция, представленная на рисунке 2.2.

Входной поток включает в себя файлы с данными о пользователях и тестах. Это позволяет автоматизированной системе создать базу тестов, а также вести учет пользователей, и предоставлять последним различные тесты для необходимых целей. Для постановки цели системы требуются постановления Минобрнауки и нормативно-правовые акты, направляющие систему в нужном направлении. Для корректной работы системы требуется пользователь (студент или преподаватель), а также программное обеспечение, позволяющее автоматизировать систему.

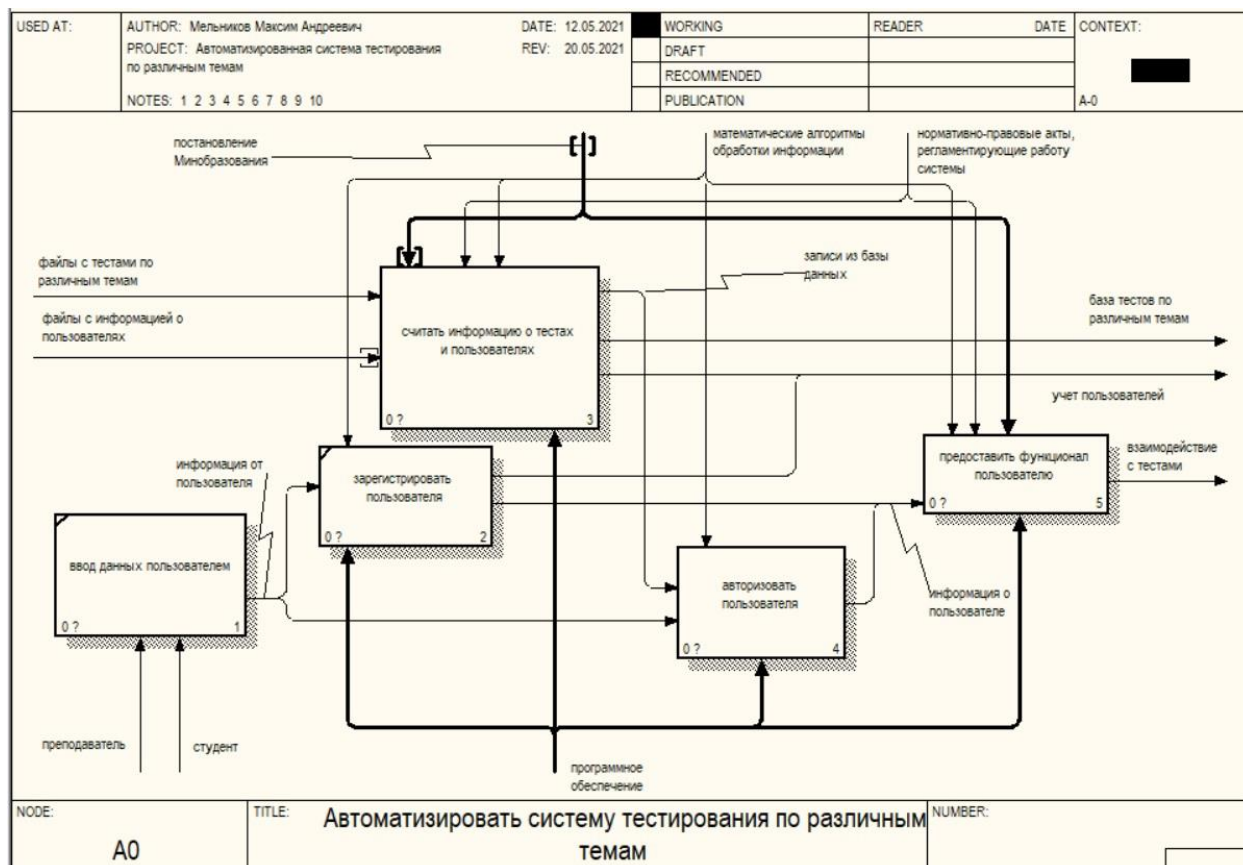


Рисунок 2.2 – декомпозиция контекстной диаграммы верхнего уровня

Данная диаграмма содержит блоки, представляющие основные действия, которые выполняет автоматизированная система. Для создания базы тестов и учета пользователей система должна прочесть входную информацию из файлов. Взаимодействие с системой у пользователя начинается либо с регистрации, либо с авторизации, в обоих случаях требуется личная (конфиденциальная) информация, которую обработает система и впоследствии предоставит доступ к функционалу.

На рисунке 2.3 продемонстрирована декомпозиция блока «считать информацию о тестах и пользователях».

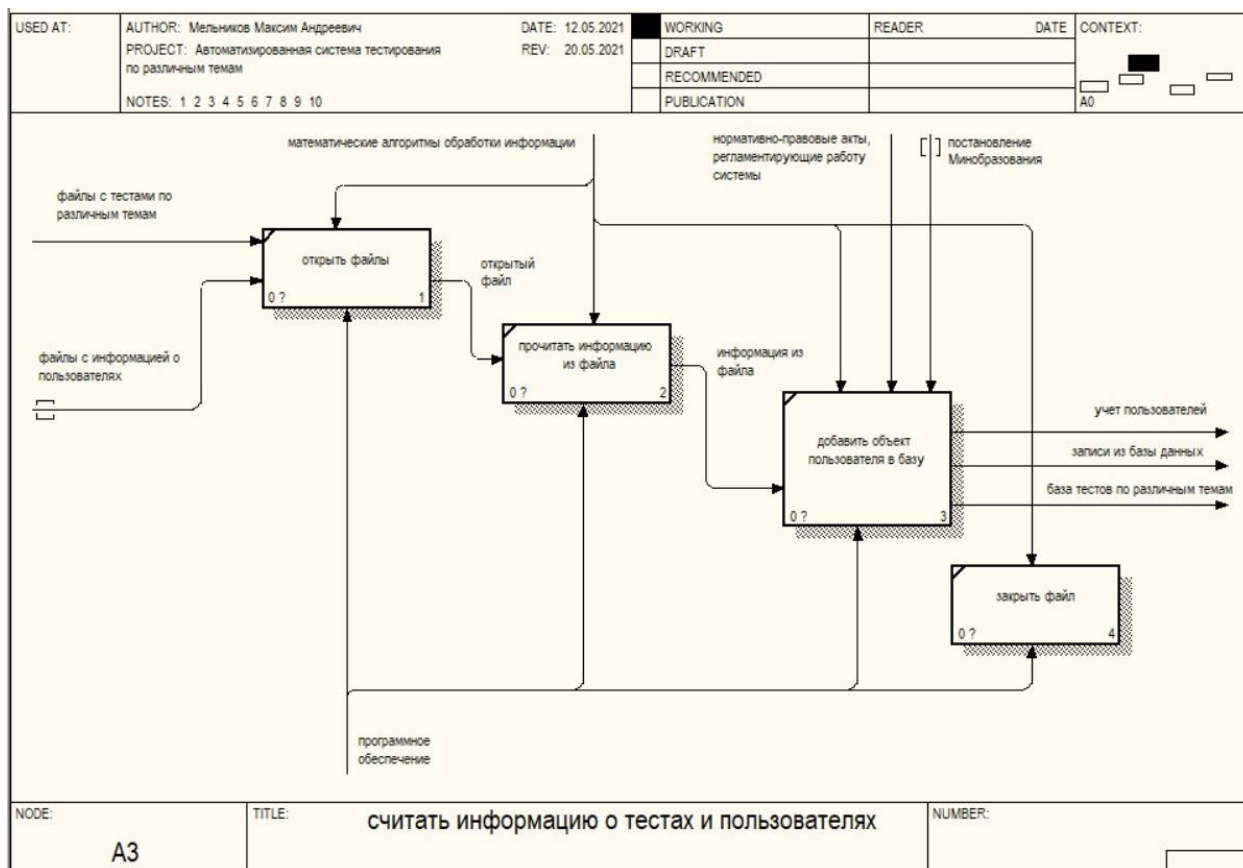


Рисунок 2.3 – декомпозиция блока «считать информацию о тестах и пользователях».

Для того, чтобы информация из файлов попала в систему тестирования и позволила создать базу тестов, а также учет пользователей, необходимо открыть файл, прочитать из него информацию, и на основе полученных данных создать сущность, которая будет ассоциироваться с пользователем и определять его возможности. В данном случае этой сущностью является объект класса. После прочтения файл обязательно нужно закрыть. Все вышеперечисленные действия выполняет сама система без участия пользователя.

На рисунке 2.4 отображена декомпозиция блока «авторизовать пользователя».

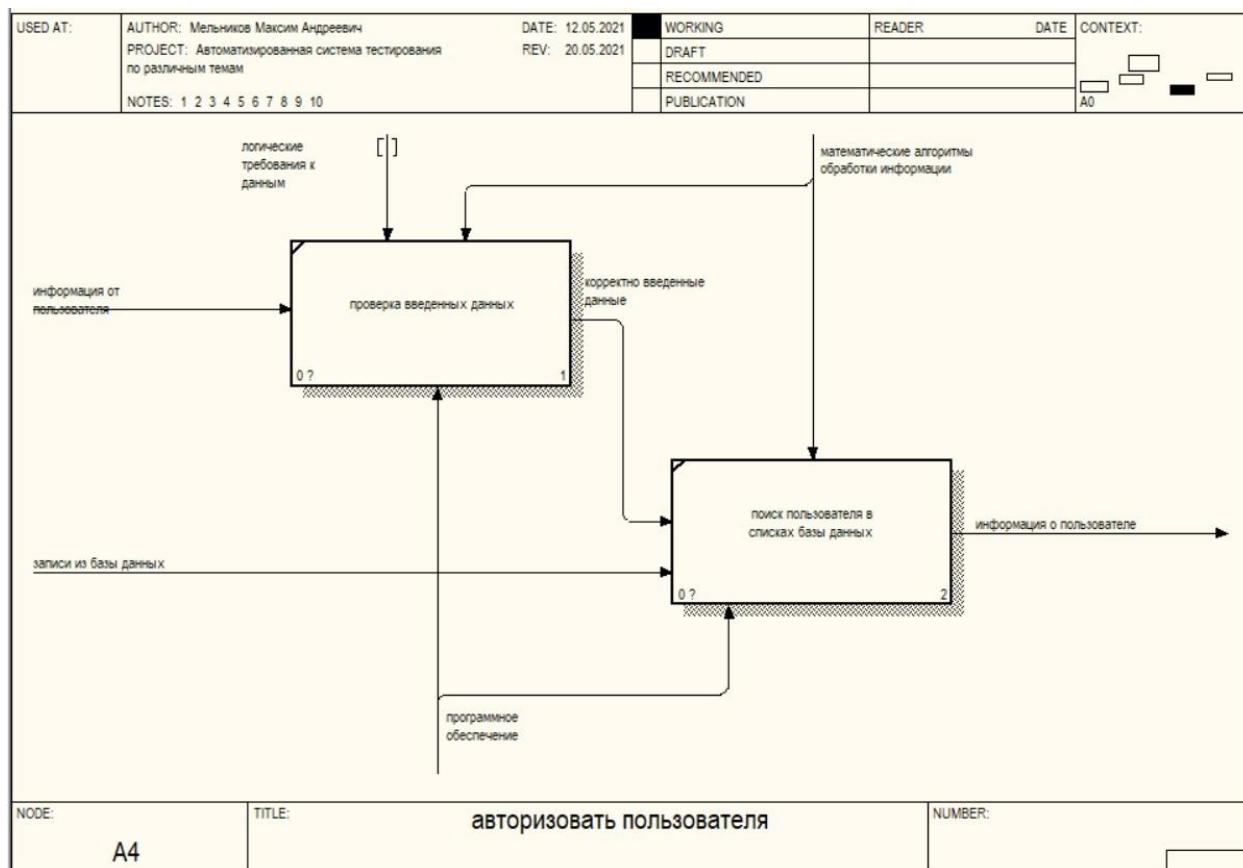


Рисунок 2.4 – декомпозиция блока «авторизовать пользователя»

Она состоит из двух блоков. Сперва системе необходимо проверить личные данные пользователя, которые он вводит при запросе. Затем система сравнивает эти данные с данными, находящимися в базе. При совпадении информации система предоставляет сущность, ассоциирующуюся в базе с пользователем, а именно объект класса. Данный объект имеет доступ к функционалу, ограниченному постановлениями Минобразования, а также нормативно правовыми актами.

На рисунке 2.5 представлена декомпозиция блока «предоставить функционал пользователю».

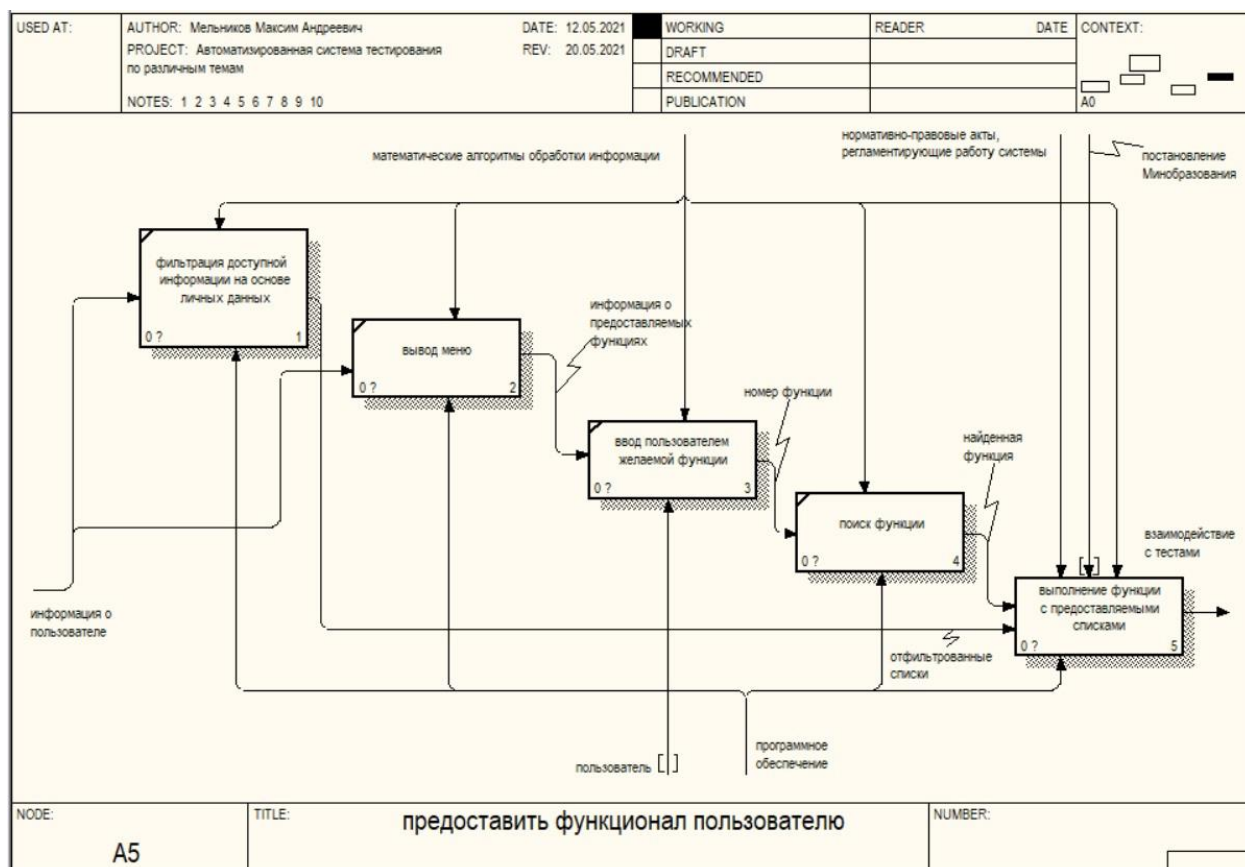


Рисунок 2.5 – декомпозиция блока «предоставить функционал пользователю»

Чтобы предоставить пользователю функционал, системе сперва нужно отфильтровать списки, с которыми данный пользователь будет в дальнейшем работать. Фильтрация происходит на основе личной информации пользователя. Только затем система выводит на экран меню, в котором описаны все возможности пользователя. Для выбора функции пользователь вводит номер, а затем осуществляется поиск данной функции в базе, и при обнаружении начинается ее выполнение. Функции работают с отфильтрованными списками и предоставляют взаимодействия с тестами для любого типа пользователя. Функции выполняются в соответствии с постановлением Минобрнауки и нормативно-правовых актов.

На рисунке 2.6 представлена декомпозиция блока «выполнение функции с предоставляемыми списка».

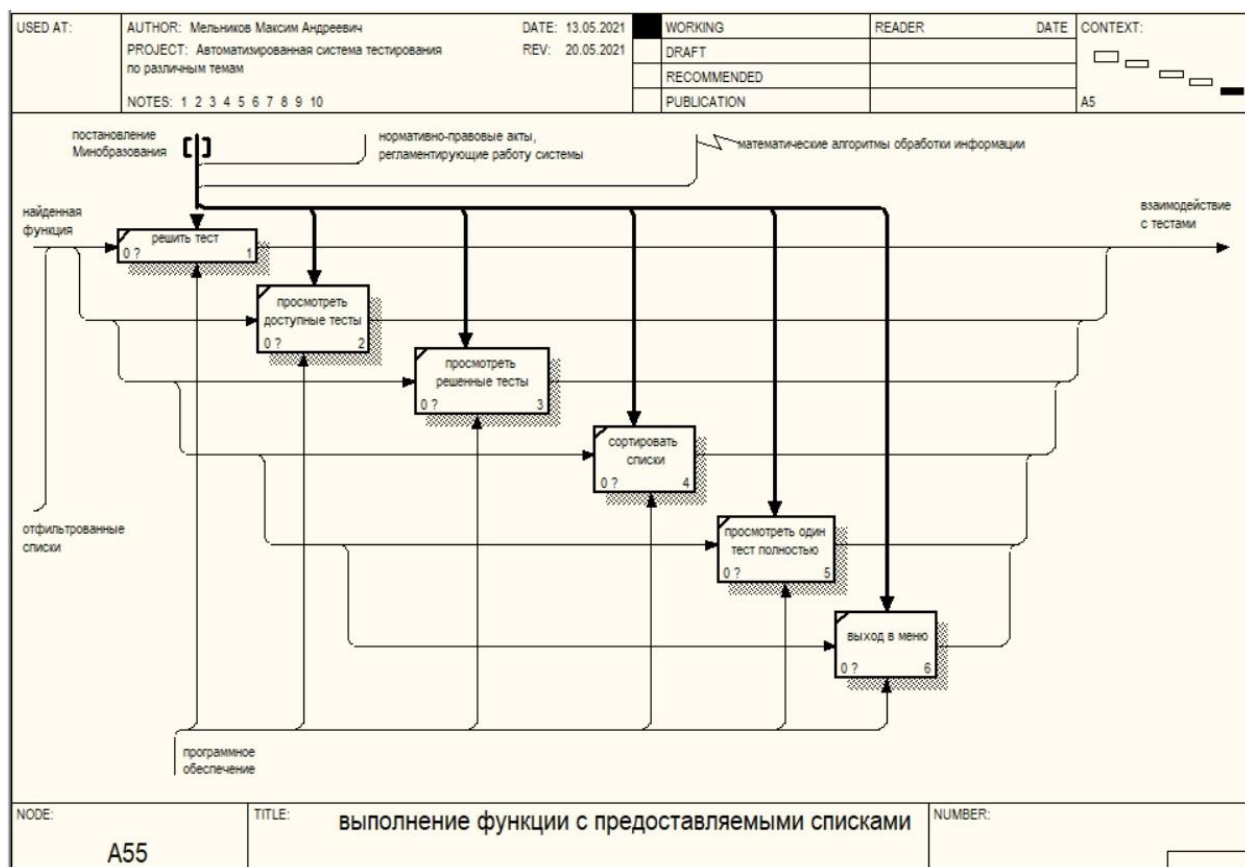


Рисунок 2.6 – декомпозиция блока «выполнение функции с предоставляемыми списками»

В данном случае рассматривается функционал студента, в котором он может решить тест, посмотреть доступные или решенные тесты, отсортировать списки или выйти обратно в меню. Каждая функция выполняется системой без участия студента, так как уже были введены необходимые и данные и произведен поиск нужной функции. Постановления Минобразования и нормативно-правовые акты служат для регламентирования функций автоматизированной системы. В итоге с помощью вышеописанных функций достигается важная цель проекта – взаимодействие с тестами.

3 СТРУКТУРА ИСПОЛЬЗУЕМЫХ ДАННЫХ

Все входные данные программы хранятся в файлах с расширением «.txt». Файл DataBaseFile хранит три строки, обозначающие пути к файлам, хранящим все пути файлов с тестами, студентами и преподавателями.

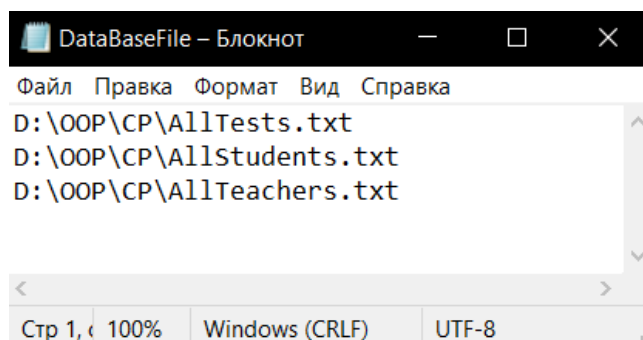


Рисунок 3.1 – Содержимое файла DataBaseFile

На рисунке 3.2 представлено содержимое файла AllTests, а именно пути ко всем существующим тестам.

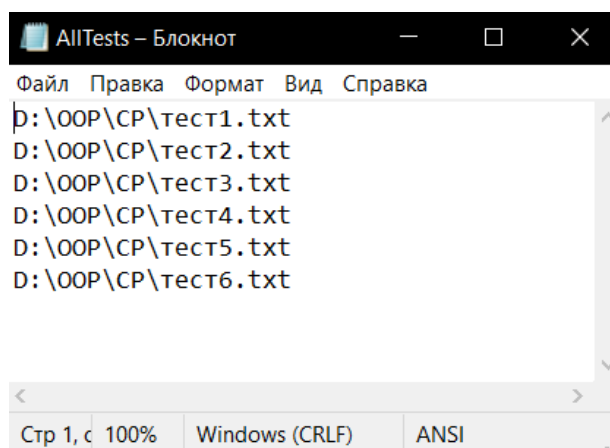


Рисунок 3.2 – Содержимое файла AllTests

В файлах AllStudents и AllTeachers также находятся пути ко всем существующим студентам и преподавателям, соответственно. На рисунке 3.3 приведен пример файла с тестом, где формат заполнения следующий:

- количество ответов в вопросах;
- курс, для которого тест предназначен;
- ID теста;
- предмет, для которого тест предназначен;
- краткое описание;
- вопрос;

- варианты ответа;
- количество баллов за вопрос;
- правильный вариант ответа.

Последние четыре пункта повторяются столько раз, сколько находится вопросов в самом тесте.

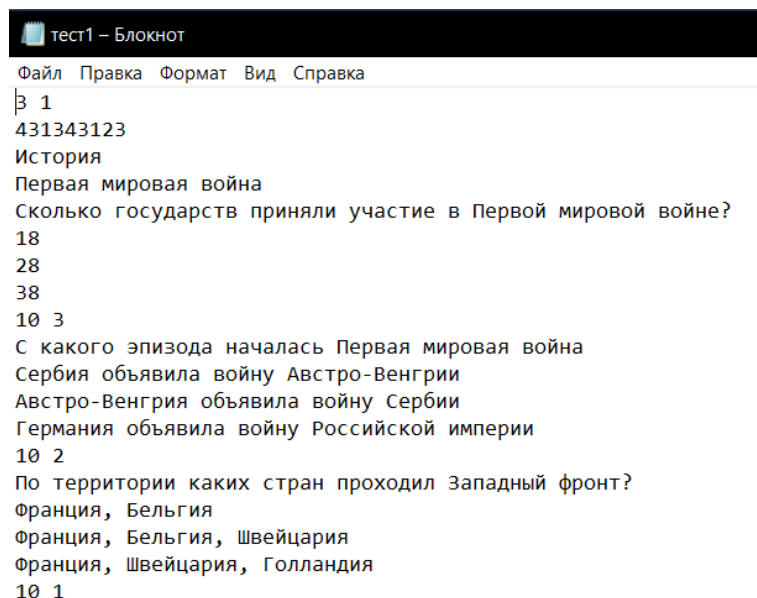


Рисунок 3.3 – Заполнение файла для теста

На рисунке 3.4 приведен пример файла студента, с следующим форматом заполнения:

- ФИО;
- пароль в хешированном виде;
- ID;
- факультет;
- группа;
- курс;
- количество учебных дисциплин и их названия, каждая с новой строки;
- количество решенных тестов;
- ID теста;
- предмет, для которого тест предназначен;
- краткое описание;
- количество набранных и максимально возможных баллов;
- количество ответов и сами ответы (в цифрах).

Последние пять пунктов повторяются столько же раз, сколько тестов решил студент.

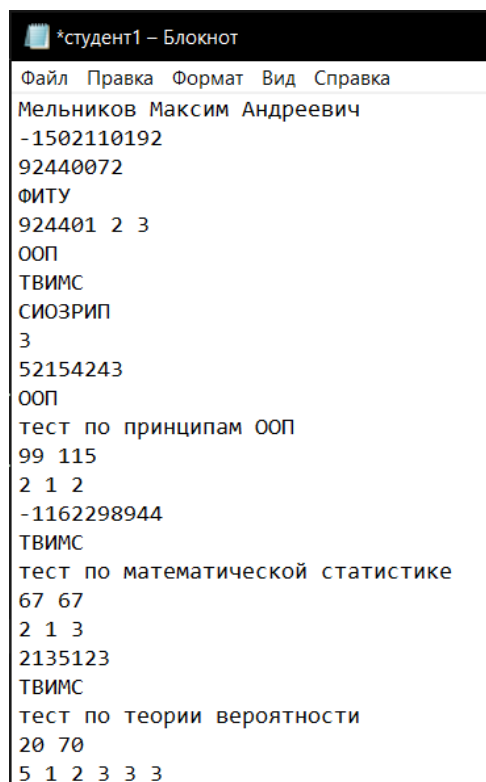


Рисунок 3.4 – Пример файла с информацией о студенте

Последним типом файлов является файл, в котором содержится информация о преподавателе. Пример приведен на рисунке 3.5, где формат заполнения следующий:

- ФИО;
- пароль в хешированном виде;
- ID;
- преподаваемый предмет;
- количество групп, в которых он преподает;
- номера групп.

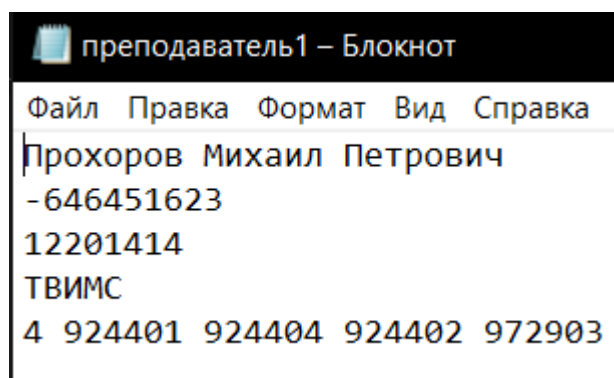


Рисунок 3.5 – Пример файла с информацией о преподавателе

Считывание данных происходит при создании объекта класса Database, точнее при вызове его конструктора, так как этот класс хранит все существующие тесты, студентов и преподавателей. Перед выходом из программы происходит запись измененных данных в те же файлы, откуда они были считаны.

4 РАЗРАБОТКА И ОПИСАНИЕ ДИАГРАММЫ КЛАССОВ ПРИЛОЖЕНИЯ

После анализа предметной области для реализации автоматизированной системы тестирования по различным темам было разработано семь классов, диаграмма которых представлена в приложении Б на рисунке Б1. Названия всех классов:

- User;
- Student;
- Teacher;
- DataBase;
- Test;
- Question;
- AdditionalFunctions;
- SolvedTest.

1 Класс User – абстрактный класс, предназначенный для наследования дочерними классами, Student и Teacher.

Данный класс содержит набор следующих полей:

- entityCount – статическое поле, предназначенное для отсчета количества текущих объектов данного класса, в том числе и дочерних, служит для создания уникального идентификатора;
- fullName – для хранения ФИО пользователя;
- hashedPassword – для хранения пароля пользователя уже в хешированном виде;
- id – уникальный идентификатор пользователя.

Список методов, реализованных в классе:

- User – конструктор по умолчанию и принимающий параметры для заполнения полей;
- ~User – виртуальный деструктор, чтобы вызывались деструкторы дочерних классов в случае динамического полиморфизма;
- GetID, GetHashedPassword, GetName, GetEntityCount – геттеры полей класса;
- PrintInformation – чисто виртуальный метод для вывода содержимого объекта, переопределен в дочерних классах;
- PrintUserInformation – метод для вывода информации о пользователе (его поля) на экран;
- Searching – метод поиска пользователя на основе совпадения пароля и идентификатора;
- Unload – чисто виртуальный метод для записи полей класса в файл.

Вид класса представлена на рисунке 4.1.

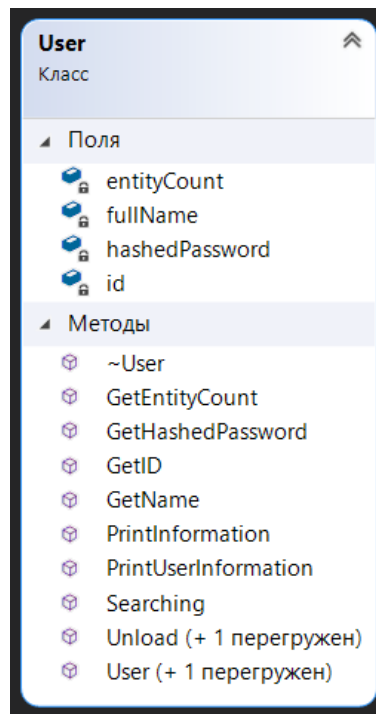


Рисунок 4.1 – Вид класса User.

2 Класс Student предназначен для входа в автоматизированную систему в качестве студента с соответствующим функционалом.

Данный класс содержит следующий набор полей:

- course – курс студента;
- faculty – факультет студента;
- group – группа студента;
- ptrSolvedTestList – указатель на список указателей тестов, решенных студентом;
- ptrSubjectList – смарт-указатель на список предметов студента;
- MenuChoice – перечисление, содержащее константы для выбора в меню.

Для реализации всего функционала были созданы методы, описанные ниже:

- Student – конструктор по умолчанию и принимающий аргументы для заполнения полей класса;
- ~Student – деструктор, в котором очищается динамически выделенная память;
- DeleteSolvedFromFilteredList – фильтрация доступных тестов, уже решенные перемещаются в отдельный список;

- DeleteEditedSolvedTest – метод удаленного теста, если он был изменен преподавателем;
- GetName, GetGroup, GetCourse, GetPtrSolvedTestList, GetPtrSubjectList – геттеры полей класса;
- Menu – метод, вызываемый после входа в систему и предоставляющий интерфейс для взаимодействия с системой;
- PrintAllSolvedTest – вывод на экран решенных тестов;
- PrintAvailableAndNoSolvedTest – вывод на экран фильтрованных тестов (доступных и еще не решенных);
- PrintInformation – вывод на экран личной информации, переопределенный метод;
- Unload – запись в файл информации о студенте, переопределенный метод.

Вид класса представлена на рисунке 4.2.

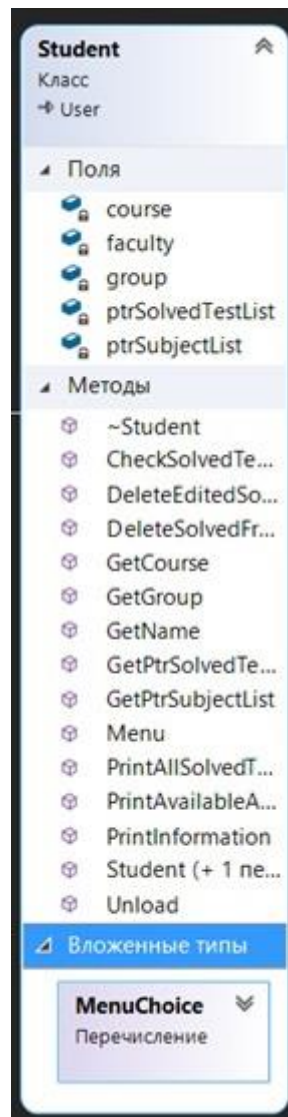


Рисунок 4.2 – Вид класса Student

3 Класс Teacher предназначен для входа в автоматизированную систему в качестве студента с соответствующим функционалом.

Данный класс содержит поля:

- ptrGroupList – смарт-указатель на список групп преподавателя;
- subject – предмет, который он преподает;
- MenuChoice – перечисление для удобного отображения интерфейса;
- QuestionMenu – перечисление для удобного отображения интерфейса

изменения вопроса.

Для реализации функционала были разработаны методы:

- Teacher – конструктор по умолчанию и принимающий аргументы для заполнения полей класса;
- ~Teacher – деструктор;
- CreateQuestion – метод для создания нового вопроса в тесте;
- CreateTest – метод для создание нового теста;
- GetHashedPassword, GetSubject, GetPtrGroupList – геттеры полей;
- Menu - метод, вызываемый после входа в систему и предоставляющий интерфейс для взаимодействия с системой;
- PrintAvailableTest – вывод доступных для взаимодействия тестов;
- PrintInformation – переопределяемый метод для вывода личной информации;
- PrintOwnStudents – вывод студентов, в группах которых ведет преподаватель;
- SearchAvailableTest – поиск доступного теста в отфильтрованном списке;
- Unload – переопределяемый метод для записи в файл.

Вид класса представлена на рисунке 4.3.

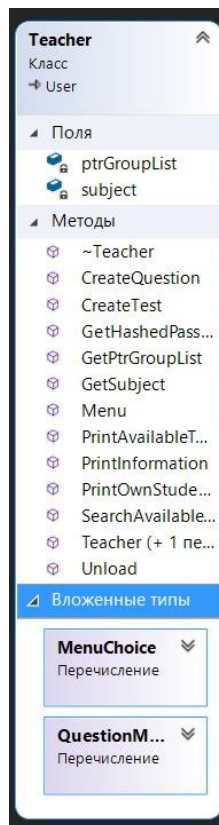


Рисунок 4.3 – Вид класса Teacher

4 Класс DataBase предназначен для хранения всех существующих тестов, студентов и преподавателей с помощью контейнеров STL. Также он предоставляет возможности регистрации и входа для пользователей, фильтрацию тестов на основе личной информации пользователя.

Поля класса:

- listOfStudents – список для хранения указателей на студентов;
- listOfTeachers – список для хранения указателей на преподавателей;
- listOfTests – список для хранения указателей на тесты;
- MenuChoice – перечисление для удобного отображения интерфейса;
- UserType – перечисление для отображения типа пользователя.

Методы класса, реализованные в данном курсовом проекте:

- DataBase – конструктор по умолчанию и принимающий в качестве параметра путь файла, из которого считывается вся необходимая информация;
- ~DataBase – деструктор, очищающий память, выделенную под объекты классов тестов, преподавателей и студентов;
- AuthorizationMenu – метод для авторизации, выбора типа пользователя;
- LoadStudentsFilter – предоставление отфильтрованного списка студентов для преподавателя;

- LoadTestWithFilter – предоставление отфильтрованного списка тестов для преподавателя и студента;
- Login – логин студента или преподавателя;
- PrintAllStudents, PrintAllTeachers, PrintAllTests – вывод на экран всех загруженных студентов, преподавателей и тестов (для отладки);
- Registration – регистрация студента или преподавателя;
- Unload – запись всей считанной ранее из файлов информации обратно в те же файлы.

Вид класса представлена на рисунке 4.4.

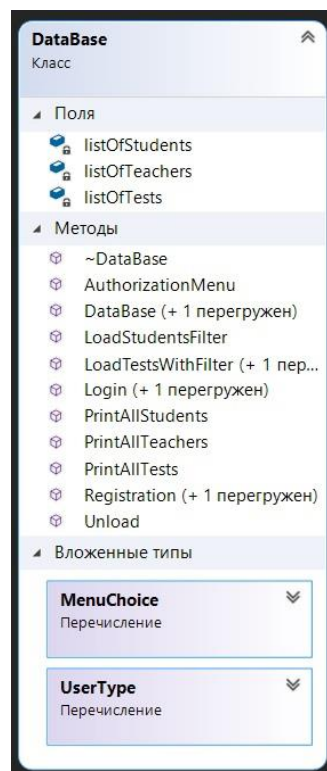


Рисунок 4.4 – Вид класса DataBase

5 Класс Test предназначен для взаимодействия с тестами, решения, просмотра, создания. Имеет набор полей:

- course – курс, для которого предназначен тест;
- entityCount – статическая переменная, обозначающая количество объектов класса, служит для создания уникальных идентификаторов;
- maxPointsPerTest – максимальное количество баллов за тест;
- numberOfQuestions – количество вопросов в тесте;
- ptrQuestionList – указатель на список вопросов;
- shortDescription – краткое описание теста;
- subject – предметная область теста;
- uniqueID – идентификатор теста.

Класс содержит следующие методы:

- Test – конструктор по умолчанию, конструктор, принимающие все необходимые параметры для создания нового объекта, конструктор, принимающий путь файла для считывания теста;
- ~Test – деструктор, который удаляет динамически выделенную память под поля теста;
- DownloadFromFile – чтение данных теста из файла;
- GetCourse, GetEntity, GetSubject, GetID, GetMaxPointsPerTest, GetNumberOfQuestions – геттеры полей класса;
- PrintTest – вывод теста на экран с вопросами;
- PrintTestBriefly – вывод на экран только необходимой информации о тесте;
- SetMaxPointsPerTest – сеттер для максимального значения баллов;
- Solving – решение теста, путем ввода ответов и создания объекта класса SolvedTest;
- UnloadTest – запись теста в файл.

Вид класса представлена на рисунке 4.5.

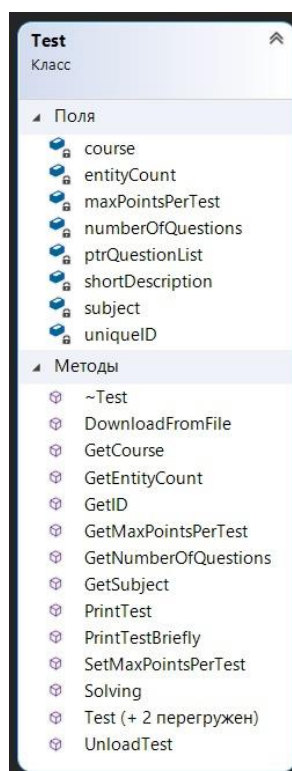


Рисунок 4.5 – Вид класса Test

6 Класс Question предназначен для работы с минимальной структурной единицей любого теста – вопросом. Включает в себя поля:

- answerOptions – указатель на массив строк вариантов ответа на вопрос;

- correctAnswerOption – правильный вариант ответа;
- numberOfAnswers – количество вариантов ответа;
- pointsPerQuestion – количество баллов за вопрос;
- question – сам вопрос.

Класс также содержит методы:

– Question – конструктор по умолчанию, а также конструктор, принимающий в качестве параметров все необходимые для создания вопроса поля;

– ~Question – деструктор, очищающий динамически выделенную память под поля класса;

– GetCorrectAnswerOption, GetNumberOfAnswers, GetPoints – геттеры полей класса;

– PrintCorrectAnswer – вывод на экран правильного ответа;

– PrintQuestion – вывод на экран вопроса с вариантами ответа;

– UnloadQuestion – запись информации о вопросе в файл.

Вид класса представлена на рисунке 4.6.

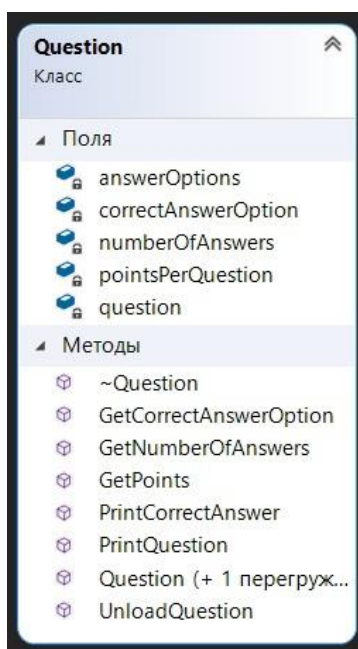


Рисунок 4.6 – Вид класса Question

7 Класс AdditionalFunctions является шаблонным и содержит один статический шаблонный метод проверки на ввод. Создан для того, чтобы хранить в себе шаблонные методы, которые могут понадобиться при создании приложения. Вид класса представлена на рисунке 4.7.

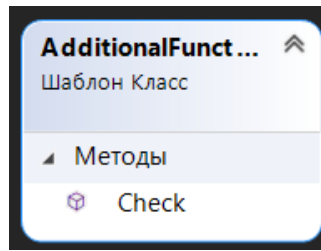


Рисунок 4.7 – Вид класса AdditionalFunctions

7 Класс SolvedTest предназначен для хранения ответов студента на вопросы, количество полученных баллов, информации о решенном тесте. Имеет следующие поля:

- answers – указатель на массив ответов студента;
- maxPoints – максимальное количество баллов за тест;
- receivedPoints – полученные баллы за тест;
- shorDescription – краткое описание решенного теста;
- subject – предметная область решенного теста;
- uniqueID – уникальный идентификатор теста.

Для реализации функционала были написаны методы:

- SolvedTest – конструктор, принимающий в качестве параметров все необходимые переменные для создания объекта класса;
- ~SolvedTest – деструктор, очищающий динамически выделенную память под поля класса;
- GetID, GetPercent, GetSubject – геттеры полей класса.

Вид класса представлена на рисунке 4.8.

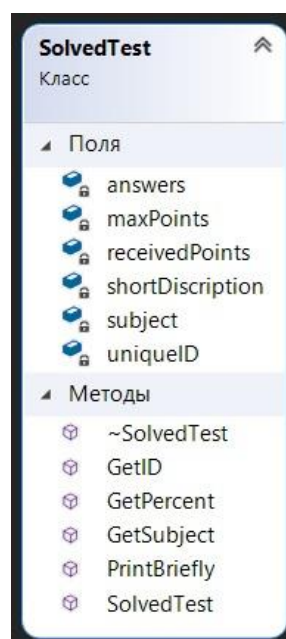


Рисунок 4.8 – Вид класса Question

5 РАЗРАБОТКА И ОПИСАНИЕ ДИАГРАММЫ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ ПРИЛОЖЕНИЯ

Диаграмма вариантов использования для автоматизированной системы тестирования по различным темам представлена в приложении В на рисунке В1.

В данной диаграмме присутствует два актора – студент и преподаватель. Они имеют как одинаковые, так и различные прецеденты. Как студент, так и преподаватель имеют возможности:

- зарегистрироваться, если до этого они ни разу не входили в систему;
- войти в систему, если уже прошли регистрацию и их данные имеются в системе, для этого нужно ввести пароль и логин;
- просмотреть свою личную информацию, логин, пароль, ФИО, а также информацию, соответствующую типу актора.

Для студента в основном все прецеденты представлены взаимодействием со списками доступных и уже решенных тестов. Эти списки выдает база данных сразу после входа в систему пользователем. Критерий фильтрации тестов строится на основе личной информации пользователя. Среди доступных тестов студент может выбрать тест для решения, либо просмотреть весь список кратко. Среди решенных тестов студент может просмотреть один тест полностью, то есть увидеть вопрос, свои варианты ответа и количество баллов, которое он набрал за вопрос. Также он может просмотреть весь список решенных тестов кратко. Оба списка доступны для сортировки по определенным критериям (доступные – по предмету, решенные – по предмету и проценту правильных ответов).

Для преподавателя все прецеденты представлены взаимодействием со списками доступных тестов и студентов, в группах которых он преподает. Фильтрация происходит также на основе личных данных пользователя. С доступными тестами преподаватель может делать следующие действия:

- создать новый тест, который добавляется в список доступных;
- удалить тест, удаление идет также и из списка;
- редактировать конкретный тест, его краткое описание, либо какой-то один вопрос. В вопросе он может изменить сам вопрос, варианты ответа, правильный вариант ответа, количество баллов за вопрос;
- просмотреть весь список тестов кратко, в табличном виде либо выбрать один тест для детального просмотра.

В списке студентов преподаватель может выбрать одного студента и просмотреть его решенные тесты, их название и количество баллов, которые

студент набрал. Также есть возможность просмотра всех студентов преподавателя в табличном виде, то есть кратко.

6 СХЕМА АЛГОРИТМА РАБОТЫ ВСЕЙ ПРОГРАММЫ И АЛГОРИТМА РАБОТЫ ДВУХ И БОЛЕЕ ОСНОВНЫХ МЕТОДОВ

В приложении Г на рисунке Г1 приведена схема алгоритма работы всей программы. Перед предоставлением доступа пользователю к интерфейсу программы производится загрузка всей необходимой информации путем чтения из файлов. Схема алгоритма чтения из файла представлена на рисунке Г2 приложения Г.

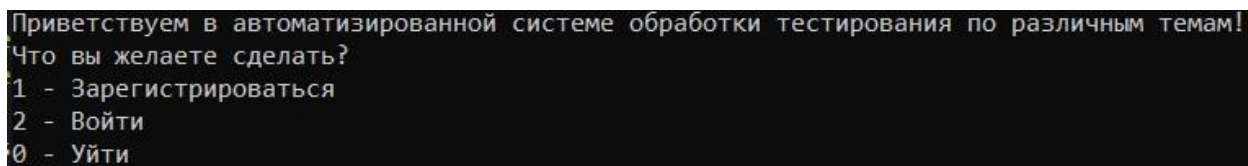
На этом этапе производится тщательная проверка сперва на открытие файлов с данным, а затем на считанные данные. Если было вызвано исключение, то оно будет обработано в конструкторе класса DataBase и приведет к тому, что будет выведено на экран сообщение об ошибке, некоторые объекты созданы не будут (если исключение возникло в конструкторе). Однако программа продолжит корректно работать и предложит пользователю сначала выбрать пройти регистрацию или ввести свои данные для входа.

После того, как пользователь смог зайти в систему, класс DataBase на основе личных данных пользователя производит фильтрацию списков и передает их в пользование последнего с помощью вызова метода Menu и передачи в качестве аргументов отфильтрованных списков.

После выполнения всех желаемых действий пользователь выходит из программы, но она работу на этом не прекращает. Перед выходом необходимо записать все данные обратно в файлы. Схема алгоритма записи данных обратно в файлы представлен на рисунке Г3 приложения Г. Сперва идет открытие файла. Затем создается итератор, указывающий на начало списка. Увеличивая в цикле значение итератора на единицу, алгоритм проходит весь список, пока не будет достигнут элемент, следующий за концом списка. В теле сперва создается путь файла, в который впоследствии будет записан объекта, а затем вызывается метод класса, отвечающий за запись объекта в файл. В качестве параметра он принимает строку, которая является ранее созданным путем файла.

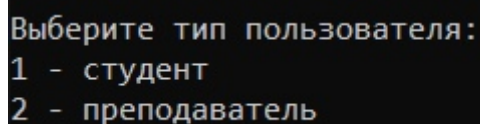
7 ОПИСАНИЕ АЛГОРИТМА ЗАПУСКА ПРИЛОЖЕНИЯ, ЕГО ИСПОЛЬЗОВАНИЯ, РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ, ТЕСТИРОВАНИЯ ОБРАБОТКИ ОШИБОК

При запуске программы на экране выводится меню, на котором пользователь сначала может выбрать регистрацию либо вход в систему, а затем и тип пользователя – студент или преподаватель, примеры снимков экрана представлены на рисунках 7.1 и 7.2.



```
Приветствуем в автоматизированной системе обработки тестирования по различным темам!  
Что вы желаете сделать?  
1 - Зарегистрироваться  
2 - Войти  
0 - Уйти
```

Рисунок 7.1 – Начальное меню программы



```
Выберите тип пользователя:  
1 - студент  
2 - преподаватель
```

Рисунок 7.2 – Выбор типа пользователя

Если пользователь выбрал регистрацию, то ему перед входом придется ввести все свои личные данные (кроме идентификатора, его система создает автоматически) для создания аккаунта в системе, пример экрана на рисунке 7.3.


```
Ваш уникальный ID = 856167011 пожалуйста, запомните его!  
Введите ФИО  
Хэтфилд Джеймс Алан  
  
Введите пароль  
1414  
  
Введите факультет  
КСИС  
  
Введите группу  
924404  
  
Введите курс  
2  
  
Введите свою учебную дисциплину  
Выйти - 0  
  
Предмет: ООП  
  
Предмет: Матем  
  
Предмет: СИОЗРИП  
  
Предмет: Физика  
  
Предмет: 0_
```

Рисунок 7.3. – Пример регистрации студента

Если пользователь выбрал вход в систему, то ему необходимо ввести всего лишь идентификатор и пароль. Затем класс Database найдет совпадения в списках и предоставит доступ пользователю к соответствующему объекту и его функционалу. Если системе не удалось найти аккаунт, выведется соответствующее сообщение, а ввод данных придется начать сначала. Пример ввода идентификатора и пароля представлен на рисунке 7.4.

```
CS D:\OOP\CP\Debug\CP.exe  
  
Введите свой ID 92440072  
  
Введите свой пароль (без пробелов) 14sa_
```

Рисунок 7.4 – Ввод пользователем идентификатора и пароля

После того, как пользователь вошел в систему, ему предоставляется доступ к функционалу, ограниченному ролью пользователя. Далее будет описываться функционал пользователя-студента. Снимок функционала представлен на рисунке 7.5.

```
Вы вошли как студент
Выберите желаемое действие:
1 - решить тест
2 - просмотреть доступные тесты
3 - просмотреть информацию о себе
4 - просмотреть решенные тесты
5 - посмотреть один решенный тест полностью
6 - Отсортировать список решенных тестов по предметам
7 - Отсортировать список решенных тестов по проценту правильных ответов
8 - Отсортировать список доступных по предметам (А-Я)
0 - Выйти
```

Рисунок 7.5 – Функционал студента

Если говорить упрощенно, то студент может взаимодействовать с двумя типа сущностей – доступными ему тестами (функции 1,2,8) и уже решенные им тестами (функции 4,5,6,7). Как доступные, так решенные тесты хранятся в списках и предоставляются системой в пользование студенту перед отображением меню. Также студент может просмотреть свою личную информацию, нажав на 3. Пример представлен на рисунке 7.6.

```
D:\OOP\CP\Debug\CP.exe
```

ФИО	ID	Ф-тет	Курс	Группа	Предметы
Мельников Максим Андреевич	92440072	ФИТУ	2	924401	ООП, ТВИМС, СИОЗРИП,

Для продолжения нажмите любую клавишу . . .

Рисунок 7.6 – Вывод личной информации студента

Нажав на 2, на экране будет выведен список доступных и еще не решенных тестов для студента. Доступными тестами являются те тесты, у которых совпадает предмет с предметом студента, а также имеется одинаковый курс. Пример на рисунке 7.7.

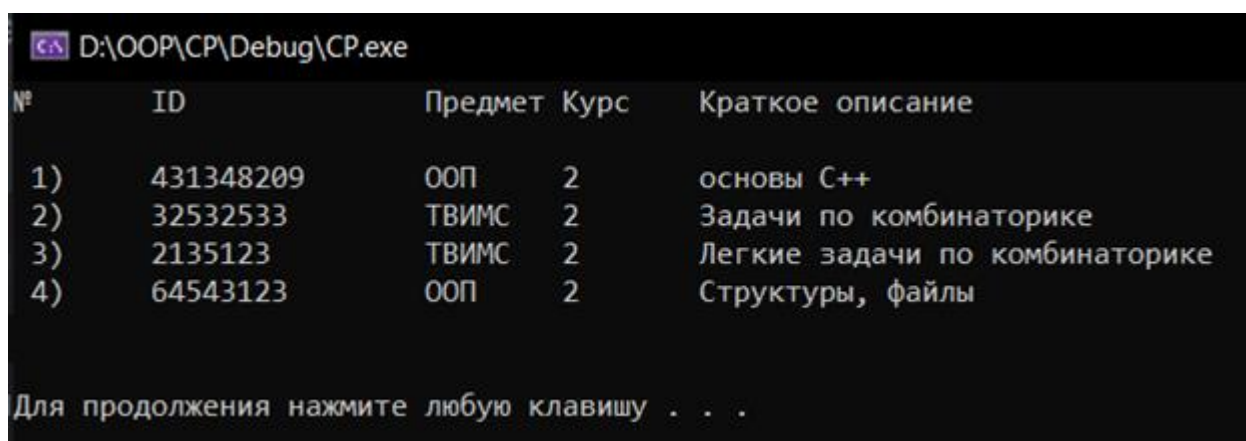


Рисунок 7.7 – Вывод списка доступных студенту тестов

Нажав на 1 для решения теста, на экран также выведется список доступных и нерешенных тестов для того, чтобы студент смог выбрать один из тестов, а затем запустится сам процесс решения. Процесс решения представляет собой вывод поочередно каждого вопроса с вариантами ответа и ожиданием ввода ответа со стороны студента. После того, как ответ будет введен на экране сразу же появится результат выбора, правильный ли ответ дал студент или нет и количество баллов, полученных за вопрос. Далее студенту нужно нажать любую клавишу для вывода следующего вопроса. Пример представлен на рисунке 7.8, где был выбран для решения тест номер три.

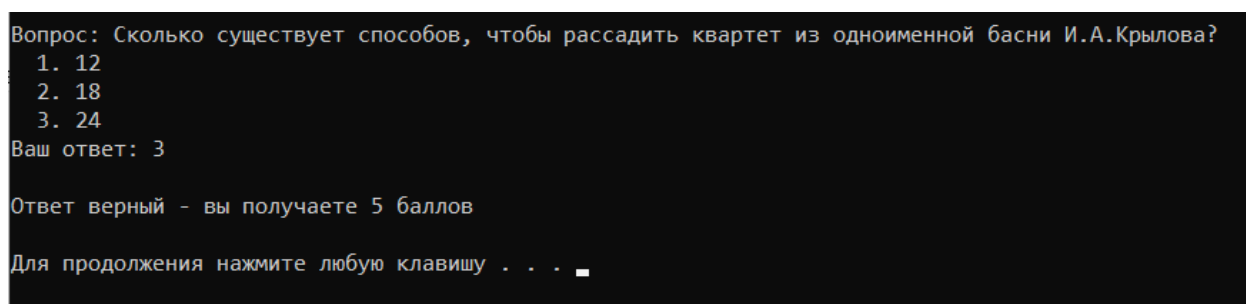


Рисунок 7.8 – Пример решения теста

После ответа на все вопросы тест удаляется из списка доступных и переходит в список решенных. Нажав на 4, можно просмотреть список решенных тестов, пример приведен на рисунке 7.9. Как видно ранее доступный тест переместился в список решенных тестов.

№	ID	Предмет	Баллы	Краткое описание
1)	213512334	ТВИМС	70/70	Тест по теории вероятности и математической статистике №1
2)	2135123	ТВИМС	55/75	Легкие задачи по комбинаторике

Для продолжения нажмите любую клавишу . . .

Рисунок 7.9 – Пример просмотра списка решенных тестов

Также при желании просмотреть свои ответы на вопросы решенного теста студенту необходимо нажать на цифру 5 в меню и выбрать из списка, ранее представленного на рисунке 7.9, желаемых тест. Просмотр ответов происходит в следующем порядке, сначала выводится информация о тесте, а затем печатается каждый вопрос, его варианты ответа, ответ студента и количество полученных баллов за вопрос, пример приведен на рисунке 7.10.

Вопрос: Сколько существует способов, чтобы рассадить квартет из одноименной басни И.А.Крылова?

1. 12
2. 18
3. 24

Правильный ответ: 3
Ваш ответ: 3
Вы получили: 5 баллов
Для продолжения нажмите любую клавишу . . .

Рисунок 7.10 – Пример просмотра решенного теста

Еще одной функцией, предоставленной студенту, является вывод списков в отсортированном виде. Для списка доступных тестов представлена одна сортировка – по предмету. Нажав на клавишу восемь, будет выведено сообщение отсортирован ли список. Если список пуст или в нем один элемент, то его сортировать не нужно, соответствующее сообщение также будет выведено на экран. Пример списка доступных тестов до и после сортировки представлен на рисунках 7.11 и 7.12.

CP D:\OOP\CP\Debug\CP.exe

№	ID	Предмет	Курс	Краткое описание
1)	431348209	ООП	2	основы C++
2)	32532533	ТВИМС	2	Задачи по комбинаторике
3)	64543123	ООП	2	Структуры, файлы

Для продолжения нажмите любую клавишу . . .

Рисунок 7.11 – Список доступных тестов до сортировки

CP D:\OOP\CP\Debug\CP.exe

№	ID	Предмет	Курс	Краткое описание
1)	431348209	ООП	2	основы C++
2)	64543123	ООП	2	Структуры, файлы
3)	32532533	ТВИМС	2	Задачи по комбинаторике

Для продолжения нажмите любую клавишу . . .

Рисунок 7.12 – Список доступных тестов после сортировки

Также сортировка доступна и для списка решенных тестов. Так как там сейчас мало тестов, то перед демонстрацией все доступные тесты будут решены. Список решенных тестов до сортировок представлен на рисунке 7.13.

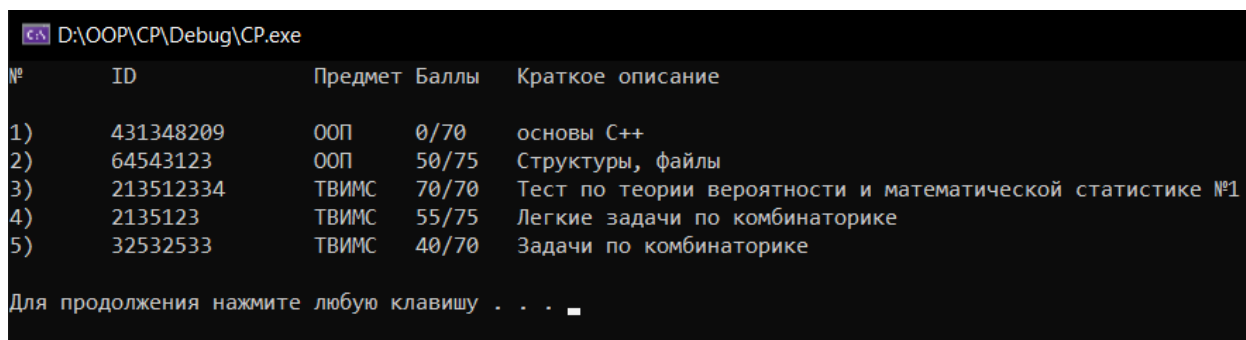
CP D:\OOP\CP\Debug\CP.exe

№	ID	Предмет	Баллы	Краткое описание
1)	213512334	ТВИМС	70/70	Тест по теории вероятности и математической статистике №1
2)	2135123	ТВИМС	55/75	Легкие задачи по комбинаторике
3)	431348209	ООП	0/70	основы C++
4)	64543123	ООП	50/75	Структуры, файлы
5)	32532533	ТВИМС	40/70	Задачи по комбинаторике

Для продолжения нажмите любую клавишу . . .

Рисунок 7.13 – Список решенных до сортировки

Нажав на клавиши шесть и семь производится сортировка по предметам и проценту набранных баллов, примеры представлены на рисунках 7.14 и 7.15, соответственно.

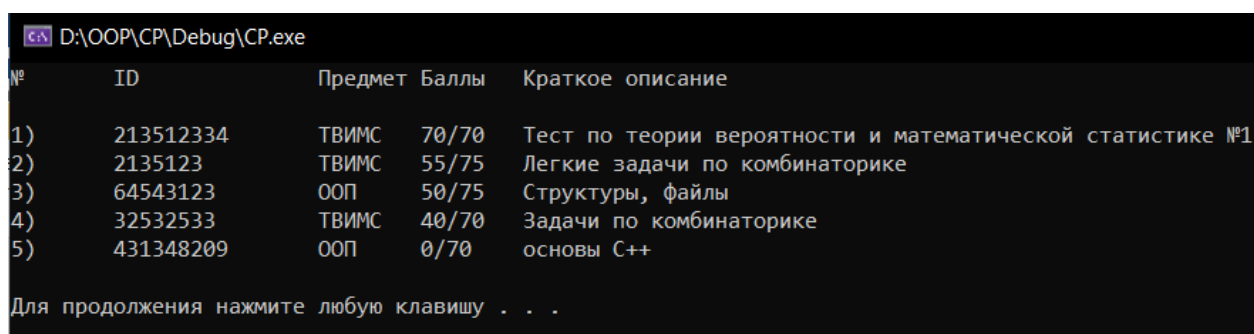


D:\OOP\CP\Debug\CP.exe

№	ID	Предмет	Баллы	Краткое описание
1)	431348209	ООП	0/70	основы C++
2)	64543123	ООП	50/75	Структуры, файлы
3)	213512334	ТВИМС	70/70	Тест по теории вероятности и математической статистике №1
4)	2135123	ТВИМС	55/75	Легкие задачи по комбинаторике
5)	32532533	ТВИМС	40/70	Задачи по комбинаторике

Для продолжения нажмите любую клавишу . . .

Рисунок 7.14 – Список после сортировки по предмету



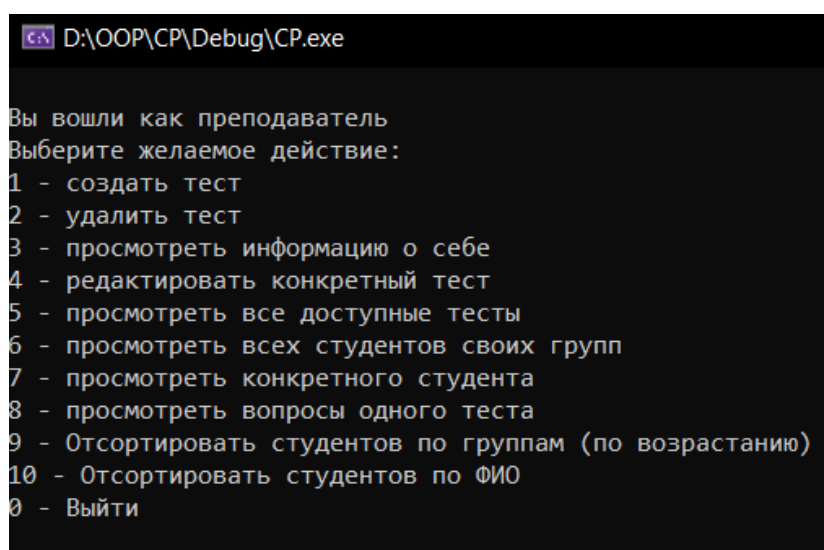
D:\OOP\CP\Debug\CP.exe

№	ID	Предмет	Баллы	Краткое описание
1)	213512334	ТВИМС	70/70	Тест по теории вероятности и математической статистике №1
2)	2135123	ТВИМС	55/75	Легкие задачи по комбинаторике
3)	64543123	ООП	50/75	Структуры, файлы
4)	32532533	ТВИМС	40/70	Задачи по комбинаторике
5)	431348209	ООП	0/70	основы C++

Для продолжения нажмите любую клавишу . . .

Рисунок 7.15 – Список после сортировки по проценту набранных баллов

Далее, на рисунке 7.16, представлен функционал преподавателя.



D:\OOP\CP\Debug\CP.exe

Вы вошли как преподаватель
Выберите желаемое действие:

- 1 - создать тест
- 2 - удалить тест
- 3 - просмотреть информацию о себе
- 4 - редактировать конкретный тест
- 5 - просмотреть все доступные тесты
- 6 - просмотреть всех студентов своих групп
- 7 - просмотреть конкретного студента
- 8 - просмотреть вопросы одного теста
- 9 - Отсортировать студентов по группам (по возрастанию)
- 10 - Отсортировать студентов по ФИО
- 0 - Выйти

Рисунок 7.16 – Функционал преподавателя

Также, как и студент, преподаватель может просмотреть личную информацию, введя цифру три. Пример приведен на рисунке 7.17.

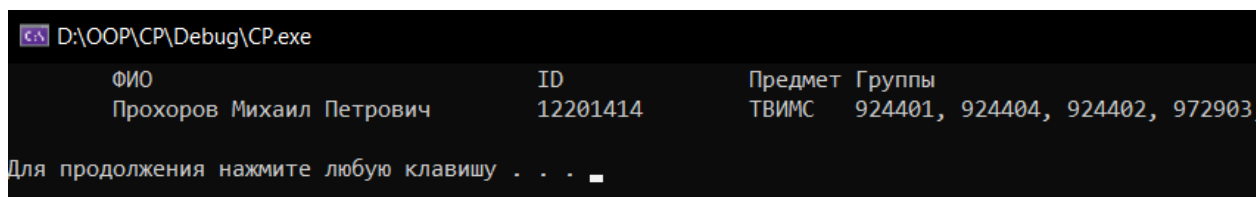


Рисунок 7.17 – Личная информация преподавателя

Говоря кратко, преподаватель может взаимодействовать с двумя сущностями – тестами (функции 1,2,4,5,8) и студентами (функции 6,7,10). Два этих типа хранятся в списках, которые класс Database предоставляет в пользование преподавателю перед выводом меню на экран. Фильтрация списков осуществляется на основе личной информации преподавателя – предмета и списка групп, в которых он этот предмет ведет.

Чтобы создать новый тест, необходимо ввести единицу, далее на экране появится просьба ввести данные о тесте, а затем и вопросы. После каждого вопроса система спрашивает, желает ли преподаватель ввести еще один вопрос. Создание теста и вопроса показано на рисунках 7.18 и 7.19.

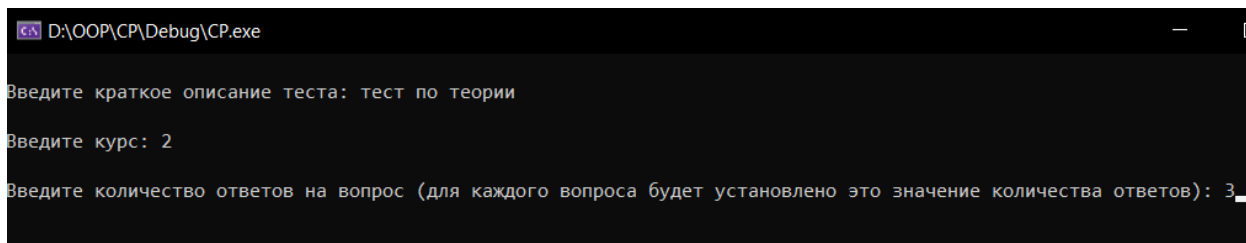


Рисунок 7.18 – Ввод данных о тесте

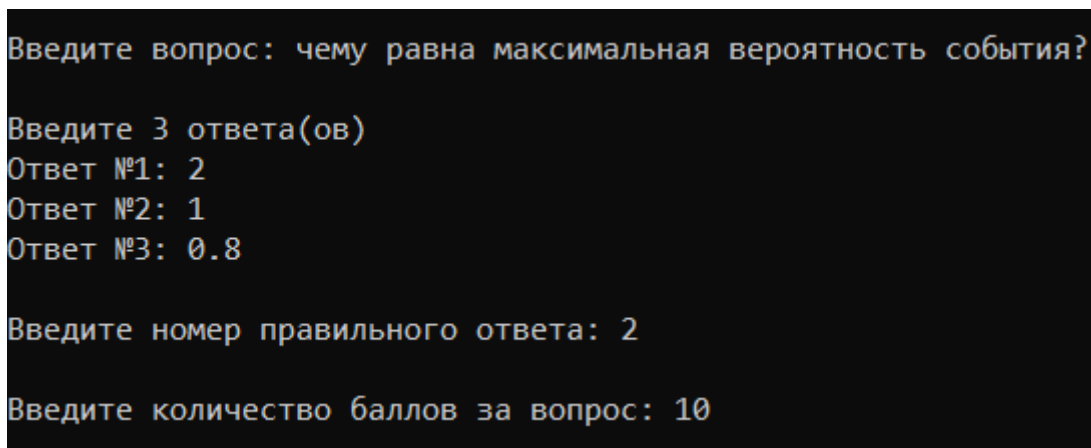
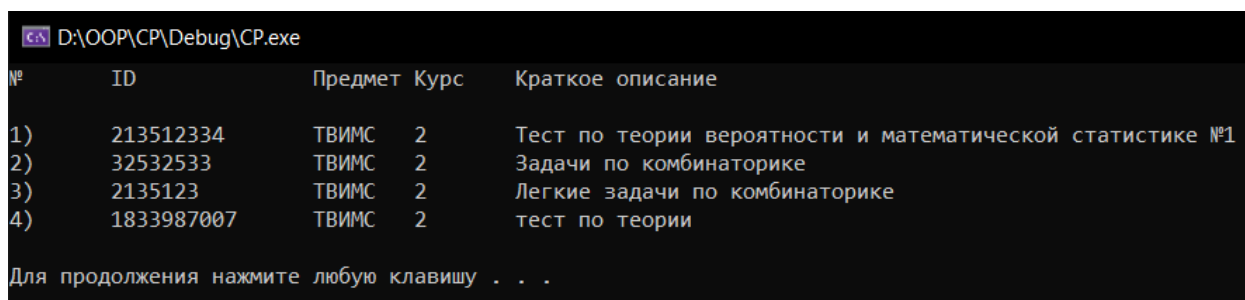


Рисунок 7.19 – Создание вопроса

Чтобы просмотреть все доступные тесты необходимо ввести цифру пять. Список доступных тестов приведен на рисунке 7.20.

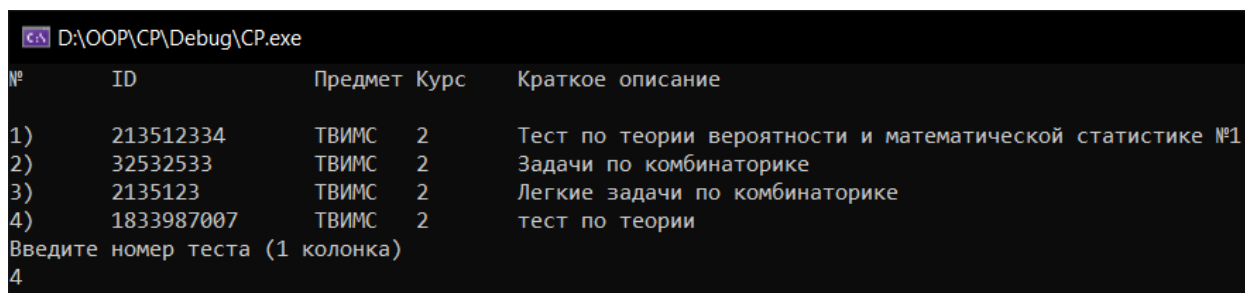


№	ID	Предмет	Курс	Краткое описание
1)	213512334	ТВИМС	2	Тест по теории вероятности и математической статистике №1
2)	32532533	ТВИМС	2	Задачи по комбинаторике
3)	2135123	ТВИМС	2	Легкие задачи по комбинаторике
4)	1833987007	ТВИМС	2	тест по теории

Для продолжения нажмите любую клавишу . . .

Рисунок 7.20 – Список доступных тестов

Как видно, недавно созданный тест был добавлен в этот список. Чтобы удалить тест нужно ввести цифру 2 и выбрать из списка тест на удаление. Пример представлен на рисунке 7.21.



№	ID	Предмет	Курс	Краткое описание
1)	213512334	ТВИМС	2	Тест по теории вероятности и математической статистике №1
2)	32532533	ТВИМС	2	Задачи по комбинаторике
3)	2135123	ТВИМС	2	Легкие задачи по комбинаторике
4)	1833987007	ТВИМС	2	тест по теории

Введите номер теста (1 колонка)
4

Рисунок 7.21 – Удаление теста

Чтобы редактировать тест достаточно ввести цифру четыре. Для проверки личности необходимо ввести свой пароль, это дополнительная мера предосторожности при редактировании тестов. Также, как и при удалении затем преподаватель выбирает из списка тот тест, который он желает изменить. Для изменения доступны краткое описание теста и его вопросы. При выборе изменения вопроса, на экране будут отображены все вопросы, и просьба ввести номер вопроса для редактирования, пример на рисунке 7.22. После выбора вопроса будет предоставлено множество функций, позволяющих изменить каждый аспект вопроса, пример на рисунке 7.23.

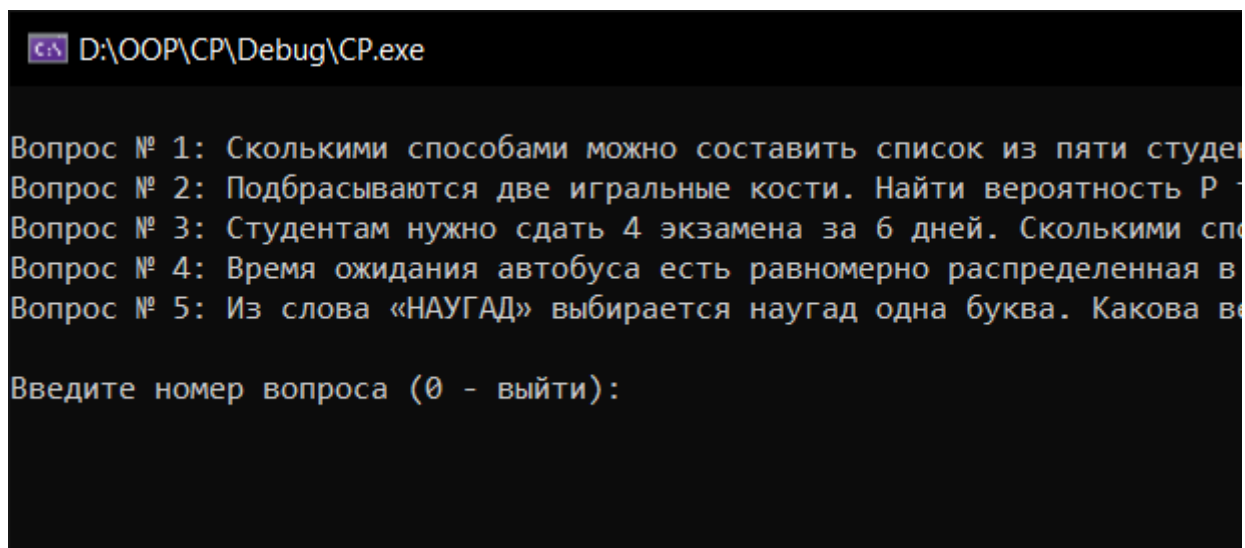


Рисунок 7.22 – Перечень вопросов для изменения

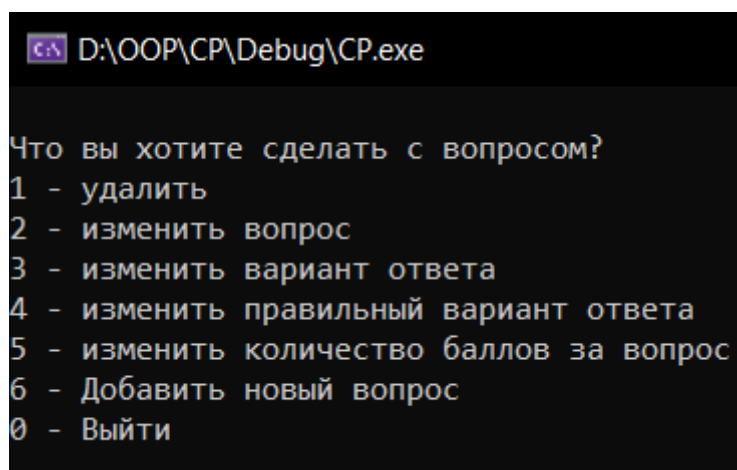


Рисунок 7.23 – Меню редактирования вопроса

Чтобы просмотреть всех студентов, преподавателю нужно ввести цифру шесть. На экране будет представлен список его студентов. Если необходимо просмотреть решенные тесты студента, достаточно ввести цифру семь и выбрать студента из списка, пример приведен на рисунке 7.24.

Также у преподавателя есть возможность отсортировать список студентов по группам и фамилии (нажатие на цифры девять и десять). Пример первоначального списка представлен на рисунке 7.24, а отсортированные по группе и фамилии – на рисунках 7.25 и 7.26 соответственно.

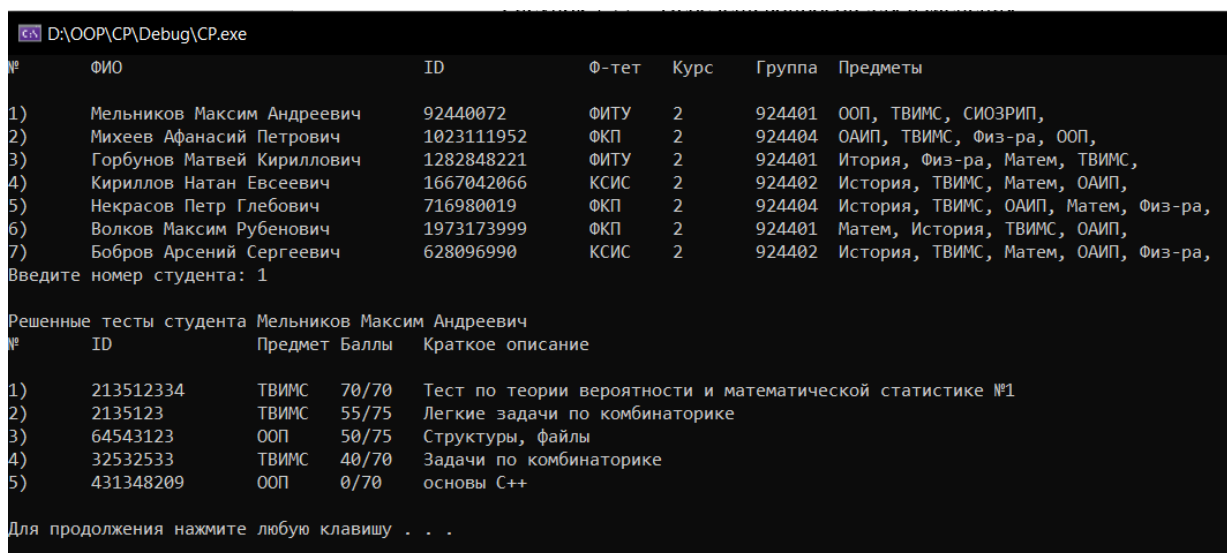


Рисунок 7.24 – Выбор одного студента из списка для просмотра его решенных тестов

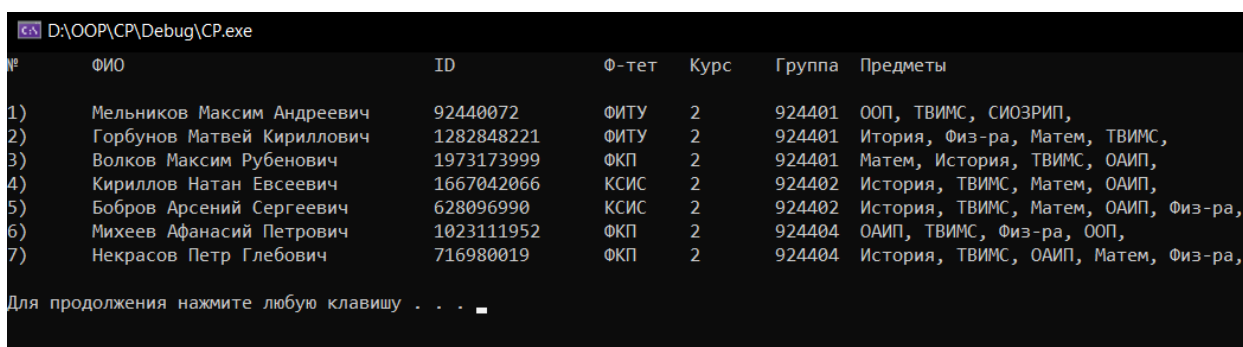


Рисунок 7.25 – Список студентов после сортировки по группам

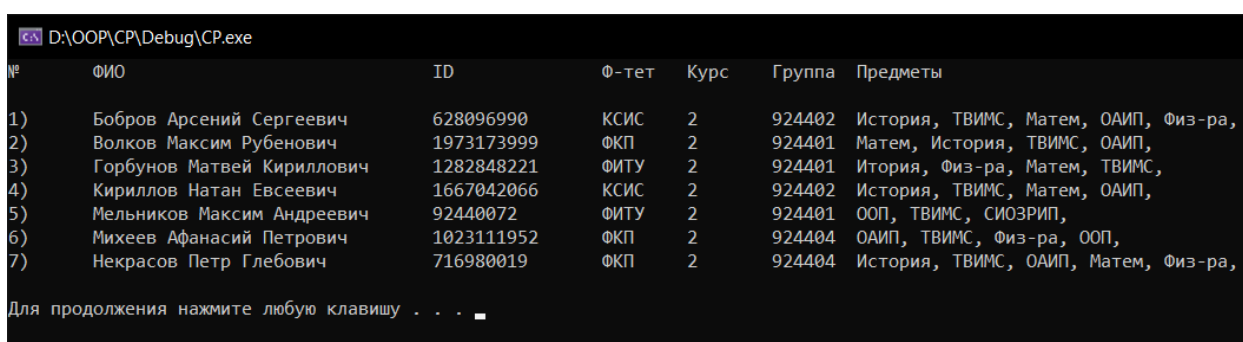


Рисунок 7.26 – Список студентов после сортировки по фамилии

Обработка ошибок в программе происходит при некорректном вводе, либо при считывании из файла. Каждый после ввода пользователем числового значения вызывается метод проверки на корректный ввод, если возникла

ошибка, то будет выведено сообщение, продемонстрированное на рисунке 7.27.

```
Приветствуем в автоматизированной системе обработки тестирования по различным темам!  
Что вы желаете сделать?  
1 - Зарегистрироваться  
2 - Войти  
0 - Уйти  
342  
  
Неверный ввод!Вводить можно только цифры и в разумных пределах!  
Введите новое значение: 3412  
Неверный ввод!Вводить можно только цифры и в разумных пределах!  
Введите новое значение: 342  
Неверный ввод!Вводить можно только цифры и в разумных пределах!  
Введите новое значение: 534352  
Неверный ввод!Вводить можно только цифры и в разумных пределах!  
Введите новое значение:
```

Рисунок 7.27 – снимок экрана при некорректном вводе

Если же при создании объектов для базы данных во входных файлах окажется некорректная информация, то будет вызвано исключение, и объект не будет создан, а на консоль выведется сообщение о некорректности данных. Пример на рисунке 7.28.

```
Конструктор STUDENT 013E2C78  
Конструктор USER 013E2948  
Конструктор STUDENT 013E2948  
Конструктор USER 013E27B0  
Конструктор STUDENT 013E27B0  
Конструктор USER 013E2B68  
Конструктор STUDENT 013E2B68  
Некорректные данные при считывании  
Для продолжения нажмите любую клавишу . . .
```

Рисунок 7.28 – Ошибка при считывании из файла

ЗАКЛЮЧЕНИЕ

В результате курсового проекта было реализовано консольное приложение, позволяющее автоматизировать систему тестирования по различным темам.

В рамках работы над курсовым проектом были реализованы:

- разделение ролей, авторизация пользователя, хранение пароля в зашифрованном виде;
- просмотр необходимой информации;
- добавление, редактирование, и удаление записей;
- поиск, сортировка и фильтрация записей;
- базовые принципы ООП;
- обработка ошибок программы.

Также были использованы:

- стандартные, пользовательские, дружественные, виртуальные функции;
- пространства имен (стандартные и собственные);
- механизмы абстракции;
- классы и наследование;
- перегрузка методов;
- шаблоны классов и методов;
- динамическое выделение памяти и умные указатели;
- потоки C++, перегрузка операторов ввода/вывода, контролирование работы с потоком;
- инкапсуляция, переопределение методов, абстрактные классы, передача параметров по ссылке и по значению, статические методы и поля.

Создание кода производилось в IDE Visual Studio 2019 на языке C++. При проектировании системы были выполнено моделирование с использованием стандартов IDEEF0 и UML 2.0.

Данная автоматизированная система позволяет легко и быстро проходить тестирование студентам, а также отслеживать свои результаты по нажатию нескольких клавиш. Для преподавателя данная система является хорошим помощником в обучении студентов, так как предоставляет удобный интерфейс для создания, удаления, редактирования тестов, а также отслеживания результатов своих студентов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Cyberleninka [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://cyberleninka.ru/article/n/istoriya-vozniknoveniya-i-razvitiya-testirovaniya> – Дата доступа: 10.04.2021
- [2] Wikipedia [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/Педагогическое_тестирование – Дата доступа: 10.04.2021
- [3] www.Grandars.ru [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.grandars.ru/college/psihologiya/pedagogicheskoe-testirovanie.html> – Дата доступа: 10.04.2021
- [4] Studme.org [Электронный ресурс]. – Электронные данные. – Режим доступа: https://studme.org/170812/pedagogika/pedagogicheskoe_testirovanie – Дата доступа: 15.04.2021
- [5] StudFiles [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://studfile.net/preview/6268117/page:33/> – Дата доступа: 15.04.2021
- [6] Инфоурок [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://infourok.ru/klassifikaciya-vidov-pedagogicheskikh-testov-2342846.html> – Дата доступа: 18.04.2021
- [7] Core [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://core.ac.uk/download/pdf/49212824.pdf> – Дата доступа: 18.04.2021
- [8] Учебные материал [Электронный ресурс]. – Электронные данные. – Режим доступа: https://works.doklad.ru/view/C6oeJg_RTDk.html – Дата доступа: 18.04.2021
- [9] Электронная библиотека полоцкого государственного университета [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://elib.psu.by:8080/handle/123456789/2976> – Дата доступа: 20.04.2021
- [10] INDIGO [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://indigotech.ru/> – Дата доступа: 20.04.2021

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг кода

Файл «User.h»

```
//абстрактный родительский класс
//статические поле и метод
//виртуальные методы

#pragma once
#include "Test.h"
using namespace std;

class User
{
    string fullName;
    int id, hashedPassword;
    static int entityCount;
public:
    User();
    User
    (
        string, //name
        int, //password
        int //id
    );
    virtual ~User();
    virtual void PrintInformation() = 0;
    virtual void Unload(string) = 0;
    static int GetEntityCount();
    ...
};
```

Файл «Teacher.h»

```
//разделение ролей – роль преподавателя
//наследование от класса User
//использование умных указателей
//переопределение методов
//дружественные методы

#pragma once
#include "Student.h"

class Teacher : public User
{
    ...
    string subject;
```

Продолжение приложения А

```
    shared_ptr<list<int>> ptrGroupList;
    ...
public:
    Teacher();
    Teacher
    (
        string, //name
        int,     //hashed password
        int,     //id
        string,  //subject
        shared_ptr<list<int>> //list of groups
    );
    ~Teacher();
    void Unload(string) override;
    void PrintInformation() override;
    void PrintOwnStudents(list<Student*>*);
    ...
    friend void EditTest(Teacher*, Test*,bool*);

};
```

Файл «Student.h»

```
//разделение ролей – роль студента
//наследование от класса User
//передача аргументов через указатель

#pragma once
#include "User.h"

class Student : public User
{
    string faculty;
    int group, course;
    shared_ptr<list<string>> ptrSubjectList;
    list<SolvedTest*>* ptrSolvedTestList;
    ...
public:
    ...
    shared_ptr<list<string>>GetPtrSubjectList();
    list<SolvedTest*>* GetPtrSolvedTestList();
    void DeleteEditedSolvedTest(int);
    friend bool sort::SortStudentByGroups(const Student*, const
Student*);
    friend bool sort::SortStudentAlphabetic(Student*, Student*);
};
```

Продолжение приложения А

```
Test* SearchTestWithID(list<Test*>*,int);
```

Файл «SolvedTest.h»

```
//использование собственного пространства имен sort
//перегрузка операторов ввода-вывода
//инкапсуляция

#pragma once
#include <iostream>
#include <list>
#include <string>
#include <fstream>
#define DEBUG
using namespace std;

class Test;
class Student;
class SolvedTest;

namespace sort
{
    bool SortSolvedBySubject(const SolvedTest*, const SolvedTest*);
    bool SortByAnswerPercentage(SolvedTest*, SolvedTest*);
    bool SortStudentByGroups(const Student*, const Student*);
    bool SortStudentAlphabetic(Student*, Student*);
    bool SortTestBySubject(const Test*, const Test*);
}

class SolvedTest
{
    string shortDiscription, subject;
    shared_ptr<list<int>> answers;
    int uniqueID, maxPoints, receivedPoints;
public:
    SolvedTest
    (
        shared_ptr<list<int>>, //answers
        string, //short decription
        int, //unique ID
        string, //subject
        int, //received points
        int //max points
    );
    ...
    friend ostream& operator<<(ostream&, const SolvedTest&);
```


Продолжение приложения А

```
friend ostream& operator<<(ostream&, const SolvedTest&);  
...  
};
```

Файл «DataBase.cpp»

```
//передача аргументов по значению и ссылке  
//обработка ошибок  
//поток C++  
//регистрация и авторизация пользователей  
//хранение пароля в зашифрованном виде  
  
DataBase::DataBase(string dataBaseFilePath)  
{  
    string allTestFilePath, allStudentFilePath, allTeacherFilePath;  
    string someTestFilePath, someStudentFilePath,  
someTeacherFilePath;  
    ifstream dataBaseFile, allTestFile, studentFile, teacherFile;  
    dataBaseFile.exceptions(ifstream::badbit | ifstream::failbit);  
    allTestFile.exceptions(ifstream::badbit | ifstream::failbit);  
    studentFile.exceptions(ifstream::badbit | ifstream::failbit);  
    teacherFile.exceptions(ifstream::badbit | ifstream::failbit);  
    int size;  
  
    try  
    {  
        dataBaseFile.open(dataBaseFilePath);  
        getline(dataBaseFile, allTestFilePath);  
        getline(dataBaseFile, allStudentFilePath);  
        getline(dataBaseFile, allTeacherFilePath);  
        dataBaseFile.close();  
    }  
    catch (const ifstream::failure&)  
    {  
        cout << "\nОшибка открытия файла: " << dataBaseFilePath <<  
"\nРекомендуется проверить наличие файла и его путь\n";  
        system("pause");  
        exit(-1);  
    }  
  
    try  
    {  
        allTestFile.open(allTestFilePath);  
        allTestFile.seekg(0, ios::end);  
        size = allTestFile.tellg();  
        allTestFile.seekg(0, ios::beg);  
  
        while (allTestFile.tellg() < (size))
```

Продолжение приложения А

```
{
    try
    {
        getline(allTestFile, someTestFilePath);
        listOfTests.push_back(new
Test(someTestFilePath));
    }
    catch (const ifstream::failure&)
    {
        cout << "\nОшибка открытия файла: " <<
someTestFilePath;
        cout << "\nРекомендуется проверить наличие файла
и его путь\nТест не был загружен\n";
        system("pause");
    }
    catch (const runtime_error& ex)
    {
        cout << ex.what();
        system("pause");
    }
}

allTestFile.close();
}
catch (const ifstream::failure&)
{
    cout << "\nОшибка открытия файла: " << allTestFilePath;

    cout << "\nРекомендуется проверить наличие файла и его путь\nТесты не
были загружены\n";
    system("pause");
}

...
}

Student* DataBase::Registration(Student* ptrStudent)
{
    hash<string> hashFunction;
    string fullName,password;
    int id;
    hash<int> hashFunctionInt;
    id = hashFunctionInt(User::GetEntityCount());
    if (id < 0)
        id *= -1;
    cout << "\nВаш уникальный ID = " << id << " пожалуйста, запомните
его!";
}
```

Продолжение приложения А

```
cout << "\nВведите ФИО\n";
cin.ignore();
getline(cin, fullName);
cout << "\nВведите пароль\n";
getline(cin, password);
int hashedPassword = hashFunction(password);

string faculty;
shared_ptr<list<string>> ptrList(new list<string>);
list<SolvedTest*>* ptrSolvedTestList = new list<SolvedTest*>;
int group, course;
cout << "\nВведите факультет\n";
getline(cin, faculty);
cout << "\nВведите группу\n";
cin >> group;
AdditionalFunctions<int>::Check(&group, 100000, 999999);
cout << "\nВведите курс\n";
cin >> course;
AdditionalFunctions<int>::Check(&course, 1, 5);
cout << "\nВведите свою учебную дисциплину\nВыйти - 0\n";
cin.ignore();
while (true)
{
    string subject;
    cout << "\nПредмет: ";
    getline(cin, subject);
    if (subject == "0")
        break;
    ptrList->push_back(subject);
}

ptrStudent = new Student(fullName, hashedPassword, faculty, id,
group, course, ptrList, ptrSolvedTestList);
listOfStudents.push_back(ptrStudent);
return ptrStudent;
}

Teacher* DataBase::Login(Teacher* userTeacher)
{
    int id;
    string password;
    hash<string> hashFunction;
    cout << "\nВведите свой ID ";
    cin >> id;
    AdditionalFunctions<int>::Check(&id, 0, 99999999999);
    cout << "\nВведите свой пароль (без пробелов) ";
    cin >> password;
    int hashedPassword = hashFunction(password);
    for (auto ptrTeacher : listOfTeachers)
```

Продолжение приложения А

```
{
    if (ptrTeacher->Searching(id, hashedPassword))
    {
        cout << "\nПользователь найден\n";
        return ptrTeacher;
    }
    cout << "\nПользователь не найден\n";
    system("pause");
    return nullptr;
}

list<Test*>* DataBase::LoadTestsWithFilter(int course,
shared_ptr<list<string>> ptrSubjList)
{
    list<Test*>* ptrFilterTestList = new list<Test*>;
    for (auto somePtrTestIter = listOfTests.begin(); somePtrTestIter
!= listOfTests.end(); somePtrTestIter++)
    {
        if (course != (*somePtrTestIter)->GetCourse())
            continue;
        else
        {
            string subject = (*somePtrTestIter)->GetSubject();
            for (auto someSubjectIter = ptrSubjList->begin();
someSubjectIter != ptrSubjList->end(); someSubjectIter++)
                if (*someSubjectIter == subject)
                {
                    ptrFilterTestList->push_back(*somePtrTestIter);
                    break;
                }
        }
    }
    return ptrFilterTestList;
}
```

Файл «Teacher.cpp»

```
//просмотр необходимой информации
//добавление, редактирование и удаление записей
//поиск, сортировка и фильтрация записей
//стандартные функции
//динамическое выделение памяти

void Teacher::Menu(list<Test*>** ptrFilteredTestList, list<Student*>*
ptrFilteredStudentList)
{
```

Продолжение приложения А

```
while (true)
{
    ...
    AdditionalFunctions<short>::Check(&choice, 0,
SORT_STUDENT_ALPHABETICAL);
    system("cls");
    switch (choice)
    {
    case CREATE_TEST:
    {
        (*ptrFilteredTestList)->push_back(CreateTest());
        break;
    }
    case DELETE_TEST:
    {
        int number;
        if (!PrintAvailableTest(ptrFilteredTestList))
            break;
        cout << "\nВведите номер теста (1 колонка)\n";
        cin >> number;
        auto iter = (*ptrFilteredTestList)->begin();
        advance(iter, number - 1);

        for (auto ptrStudent : *ptrFilteredStudentList)
            ptrStudent->DeleteEditedSolvedTest((*iter)-
>GetID());

        delete* iter;
        (*ptrFilteredTestList)->erase(iter);
        break;
    }
    case VIEW_OWN_INF:
        cout << "\tФИО\t\t\t\tID\t\tПредмет\tГруппы\n";
        this->PrintInformation();
        break;
    case EDIT_TEST:
    {
        if (!PrintAvailableTest(ptrFilteredTestList))
            break;
        Test* test = SearchAvailableTest(ptrFilteredTestList);
        bool isEdit = false;
        EditTest(this, test, &isEdit);
        if (isEdit)
            for (auto ptrStudent : *ptrFilteredStudentList)
                ptrStudent->DeleteEditedSolvedTest(test-
>GetID());

        if (test->GetNumberOfQuestions() == 0)
```

Продолжение приложения А

```
for (auto iter = (*ptrFilteredTestList)->begin(); iter !=
(*ptrFilteredTestList)->end(); iter++)
    if (test->GetID() == (*iter)->GetID())
    {
        cout << "\n\nТест стал пустым и его
пришлось удалить\n";
        (*ptrFilteredTestList)->erase(iter);
        delete test;
    }
    test = nullptr;
    break;
}
case VIEW_ALL_AVAILABLE_TEST:
    PrintAvailableTest(ptrFilteredTestList);
    break;
case VIEW_ALL_STUDENTS:
    PrintOwnStudents(ptrFilteredStudentList);
    break;
case VIEW_ONE_STUDENT:
{
    PrintOwnStudents(ptrFilteredStudentList);
    int number;
    cout << "\nВведите номер студента: ";
    cin >> number;
    AdditionalFunctions<int>::Check(&number, 1,
ptrFilteredStudentList->size());
    auto ptrStudentIter = ptrFilteredStudentList->begin();
    advance(ptrStudentIter, number - 1);
    cout << "\nРешенные тесты студента " <<
(*ptrStudentIter)->GetName()<<"\n";
    (*ptrStudentIter)->PrintAllSolvedTestBriefly();
    break;
}
case VIEW_ONE_TEST_FULLY:
{
    if (!PrintAvailableTest(ptrFilteredTestList))
        break;
    Test* test = SearchAvailableTest(ptrFilteredTestList);
    test->PrintTest();
    break;
}
...
case SORT_STUDENT_ALPHABETICAL:
{
    int size = ptrFilteredStudentList->size();
    if (size == 0)
        cout << "\nСписок пуст\n";
    else if (size == 1)
        cout << "\nВ списке один элемент\n";
```

Продолжение приложения А

```
else
    {
        ptrFilteredStudentList-
>sort(sort::SortStudentAlphabetic);
        cout << "\nСписок отсортирован\n";
    }
    break;
}
default:
    return;
}
cout << "\n\n";
system("pause");
}
}

Test* Teacher::CreateTest()
{
    string shortDiscription, subject = this->subject;
    int course, uniqueID, answersNumber;
    hash<int> hashFunction;
    list<Question*>* ptrQuestionList = new list<Question*>;
    cin.ignore();
    uniqueID = hashFunction(Test::GetEntityCount());
    cout << "\nВведите краткое описание теста: ";
    getline(cin, shortDiscription);
    cout << "\nВведите курс: ";
    cin >> course;
    AdditionalFunctions<int>::Check(&course, 1, 6);
    cout << "\nВведите количество ответов на вопрос (для каждого
вопроса будет установлено это значение количества ответов): ";
    cin >> answersNumber;
    AdditionalFunctions<int>::Check(&answersNumber, 1, 5);
    int isContinue = 1;
    do
    {
        ptrQuestionList->push_back(CreateQuestion(answersNumber));
        system("cls");
        cout << "\nЖелаете ввести еще один вопрос? 1 - да, 0 -
нет\n";
        cin >> isContinue;
        AdditionalFunctions<int>::Check(&isContinue, 0, 1);
    } while (isContinue);
    return new Test(uniqueID, shortDiscription, subject, course,
ptrQuestionList);
}

bool Teacher::PrintAvailableTest(list<Test*>** ptrFilteredTestList)
```

Продолжение приложения А

```
{
    if (!(*ptrFilteredTestList)->size())
    {
        cout << "\nСписок доступных тестов пуст\n";
        return false;
    }
    int i = 1;
    cout << "№\tID\t\tПредмет\tКурс\tКраткое описание\n";
    for (auto ptrTest : **ptrFilteredTestList)
    {
        cout << "\n" << i++ << " ) ";
        ptrTest->PrintTestBriefly();
    }
    return true;
}

void EditTest(Teacher* ptrTeacher, Test* ptrTest, bool* isEdit)
{
    while (true)
    {
        system("cls");
        string password;
        hash<string> hashFunction;
        cout << "\nВведите ваш пароль: ";
        cin >> password;
        int hashedPassword = hashFunction(password);
        if (hashedPassword != ptrTeacher->GetHashedPassword())
        {
            cout << "\nВ доступе отказано\n";
            return;
        }
        int value;
        cout << "\nВыберите поле изменения:\n1 - Краткое описание\n2 - Вопрос\n0 - Выйти\n";
        cin >> value;
        AdditionalFunctions<int>::Check(&value, 0, 2);
        system("cls");
        switch (value)
        {
            case 1:
                cin.ignore();
                cout << "\nВведите новое краткое описание\n";
                getline(cin, ptrTest->shortDescription);
                *isEdit = true;
                break;
            case 2:
                {
                    while (true)
```


Продолжение приложения А

```
{
    system("cls");
    int i = 1, number;
    for (auto q : *(ptrTest->ptrQuestionList))
        cout << "\nВопрос № " << i++ << ": " << q-
>question;

    cout << "\n\nВведите номер вопроса (0 - выйти):
";

    cin >> number;
    AdditionalFunctions<int>::Check(&number, 0,
ptrTest->ptrQuestionList->size());
    if (!number)
        break;
    auto iter = ptrTest->ptrQuestionList->begin();
    advance(iter, number - 1);
    system("cls");
    cout << "\nЧто вы хотите сделать с вопросом?\n1 -
удалить\n2 - изменить вопрос\n3 - изменить вариант ответа";
    cout << "\n4 - изменить правильный вариант
ответа\n5 - изменить количество баллов за вопрос\n6 - Добавить новый
вопрос\n0 - Выйти\n\n";
    cin >> value;
    AdditionalFunctions<int>::Check(&value, 0,
Teacher::ADD_NEW_QUESTION);
    if (value)
        *isEdit = true;
    switch (value)
    {
    case Teacher::DELETE:
        delete* iter;
        ptrTest->ptrQuestionList->erase(iter);
        break;
    case Teacher::CHANGE_QUESTION:
        cout << "\nВведите новый вопрос: ";
        cin.ignore();
        getline(cin, (*iter)->question);
        break;
    case Teacher::CHANGE_ANSWER_OPTION:
        for (int i = 0; i < (*iter)-
>numberOfAnswers; i++)
            cout << "\nОтвет №" << i + 1 << " : "
<< (*iter)->answerOptions[i];
        cout << "\n\nВведите номер ответа для
изменения: ";

        cin >> number;
        cin.ignore();
        cout << "\nВведите новый ответ: ";
        getline(cin, (*iter)->answerOptions[number -
1]);
    }
```

```

        break;
    case Teacher::CHANGE_CORRECT_ANSWER:
        cout << "\n" << (*iter)->question;
        for (int i = 0; i < (*iter)-
>numberOfAnswers; i++)
            cout << "\nОтвет №" << i + 1 << " : "
<< (*iter)->answerOptions[i];
            cout << "\nВведите новый правильный вариант
ответа: ";

            cin >> (*iter)->correctAnswerOption;
            break;
    case Teacher::CHANGE_POINTS:
        cout << "\nВведите новое количество баллов
за вопрос: ";

        cin >> (*iter)->pointsPerQuestion;
        break;
    case Teacher::ADD_NEW_QUESTION:
        ptrTest->ptrQuestionList-
>push_back(ptrTeacher->CreateQuestion((*iter)->numberOfAnswers));
        break;
    default:
        break;
    }

    }
    break;
}
default:
    return;
}
}

}

void Teacher::Unload(string path)
{
    ofstream file;
    file.open(path);
    if (!file.is_open())
        exit(-11);
    User::Unload(file);
    file << this->subject << "\n" << this->ptrGroupList->size();
    for (auto iter = ptrGroupList->begin(); iter != ptrGroupList-
>end(); iter++)
        file << " " << (*iter);
    file.close();
}

```

Файл «SolvedTest.cpp»

//перегрузка операторов ввода/вывода

Продолжение приложения А

```
#include "SolvedTest.h"

ofstream& operator<<(ofstream& file, const SolvedTest& obj)
{
    file << obj.uniqueID << "\n" << obj.subject << "\n" <<
obj.shortDiscription << "\n";
    file << obj.receivedPoints << " " << obj.maxPoints << "\n";
    file << obj.answers->size();
    for (auto iter = obj.answers->begin(); iter != obj.answers-
>end(); iter++)
        file << " " << (*iter);
    return file;
}

ostream& operator<<(ostream& out, const SolvedTest& obj)
{
    out << obj.uniqueID << "\n" << obj.subject << "\n" <<
obj.shortDiscription << "\n";
    out << obj.receivedPoints << " " << obj.maxPoints << "\n";
    out << obj.answers->size();
    for (auto iter = obj.answers->begin(); iter != obj.answers-
>end(); iter++)
        out << " " << (*iter);
    return out;
}

bool sort::SortByAnswerPercentage(SolvedTest* ptr1, SolvedTest* ptr2)
{
    return (ptr1->GetPercent() > ptr2->GetPercent());
}

bool sort::SortSolvedBySubject(const SolvedTest* ptr1, const
SolvedTest* ptr2)
{
    return (ptr1->subject.compare(ptr2->subject) < 0);
}
```

Файл «AdditionalFunctions.h»

```
//шаблонный класс и метод
//контролирование работы с потоком

#pragma once
#include "Question.h"
#include <conio.h>
template <class T>
class AdditionalFunctions
{

```

Продолжение приложения А

```
public:
    static void Check(T*, double, double);
};

template<class T>
void AdditionalFunctions<T>::Check(T* value, double min , double max )
{
    if (cin.good() && (*value) >= min && (*value) <= max)
        return;
    cin.clear();
    cin.ignore(500, '\n');
    do
    {
        cout << "\nНеверный ввод!Вводить можно только цифры и в
разумных пределах!\nВведите новое значение: ";
        char x = 0;
        *value = 0;
        bool isMinus = false, isAppear = false;
        do
        {
            x = _getch();
            if (x == '-' && !isAppear)
            {
                cout << x;
                isMinus = true;
                isAppear = true;
            }
            if (x >= '0' && x <= '9')
            {
                isAppear = true;
                cout << x;
                *value *= 10;
                if (!isMinus) *value += (x - 48);
                else *value -= (x - 48);
            }
            if (x == 8)
            {
                cout << "\b \b";
                *value /= 10;
            }
        } while (x != 13);
    } while (*value < min || *value > max);
}
```

ПРИЛОЖЕНИЕ Б
(обязательное)
Диаграмма классов

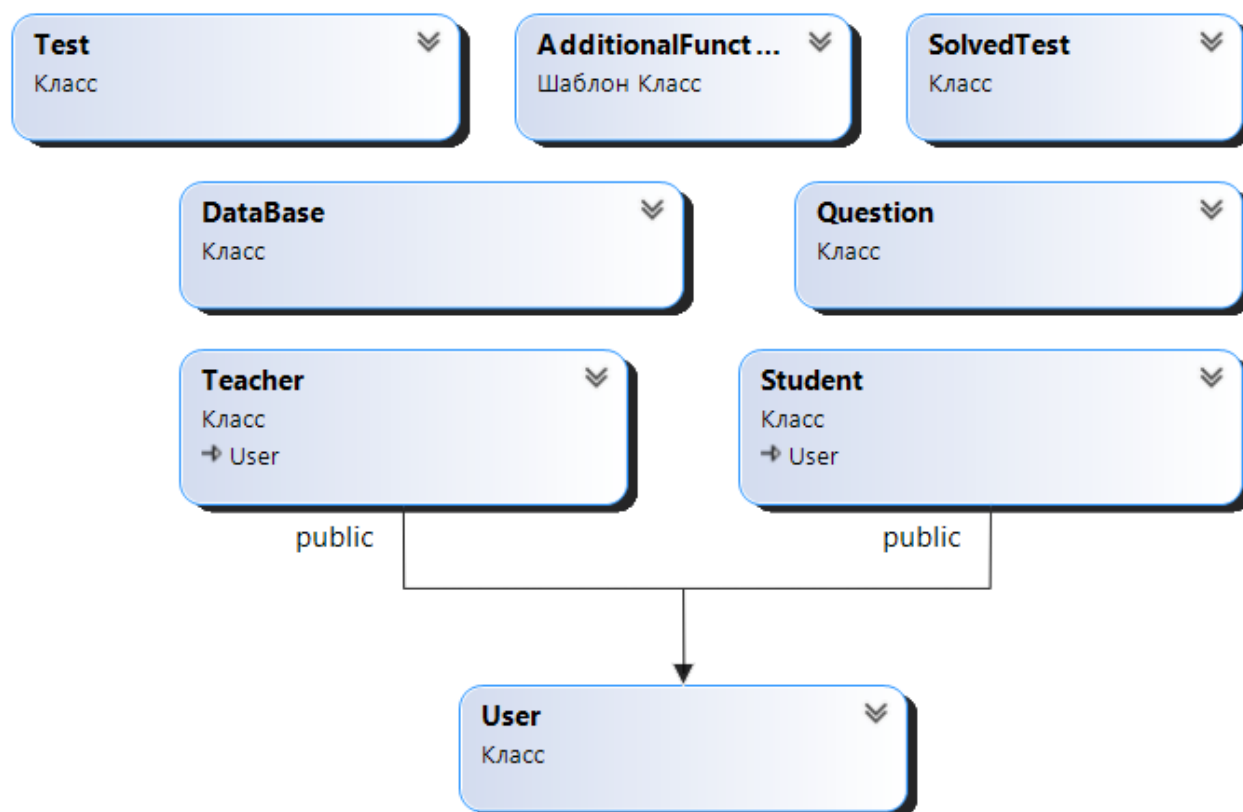


Рисунок Б1 – диаграмма классов автоматизированной системы тестирования
по различным темам

ПРИЛОЖЕНИЕ В **(обязательное)** **Диаграмма вариантов использования**

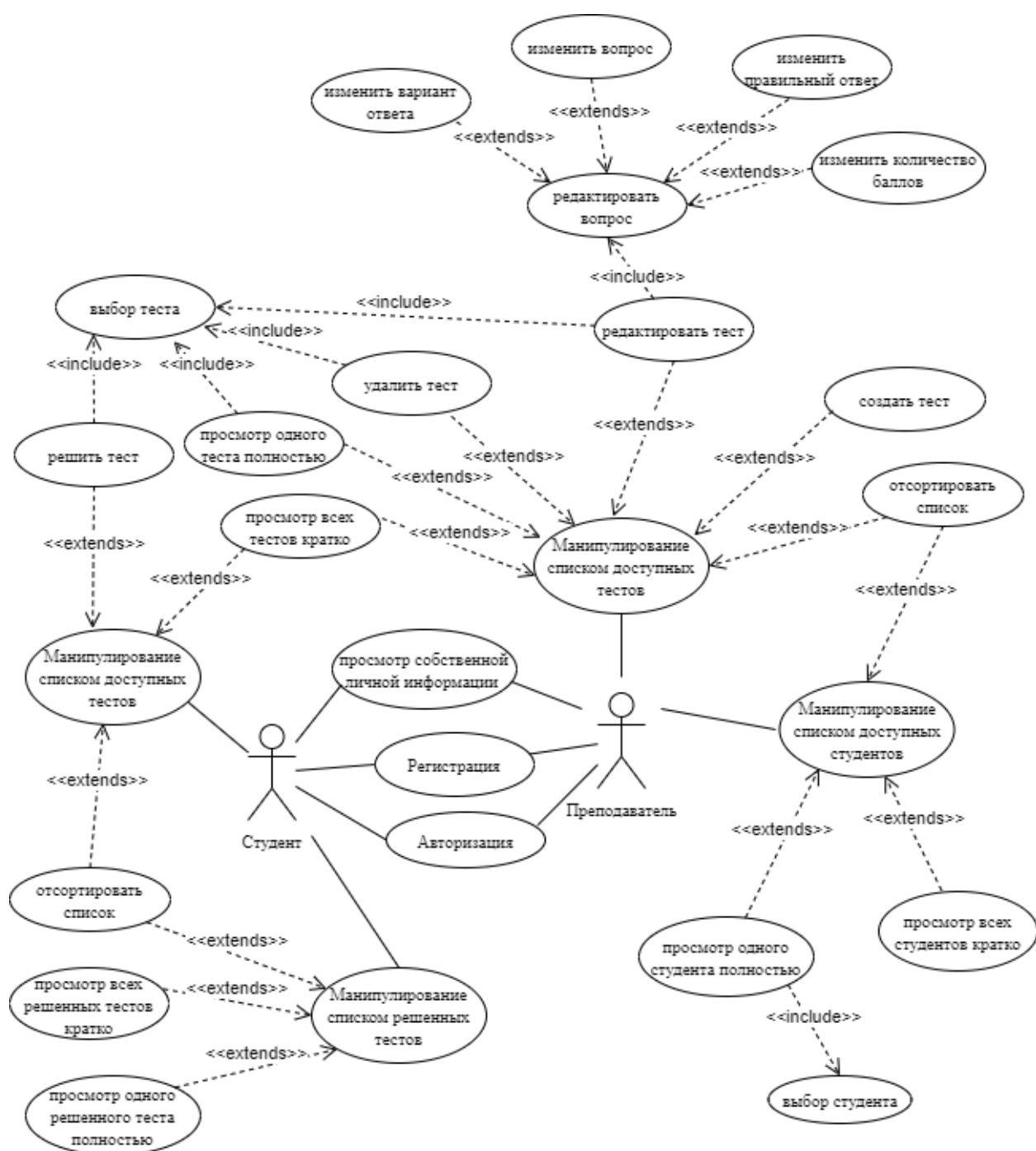


Рисунок В1 – диаграмма классов автоматизированной системы тестирования по различным темам

ПРИЛОЖЕНИЕ Г
(обязательное)
Схема алгоритмов

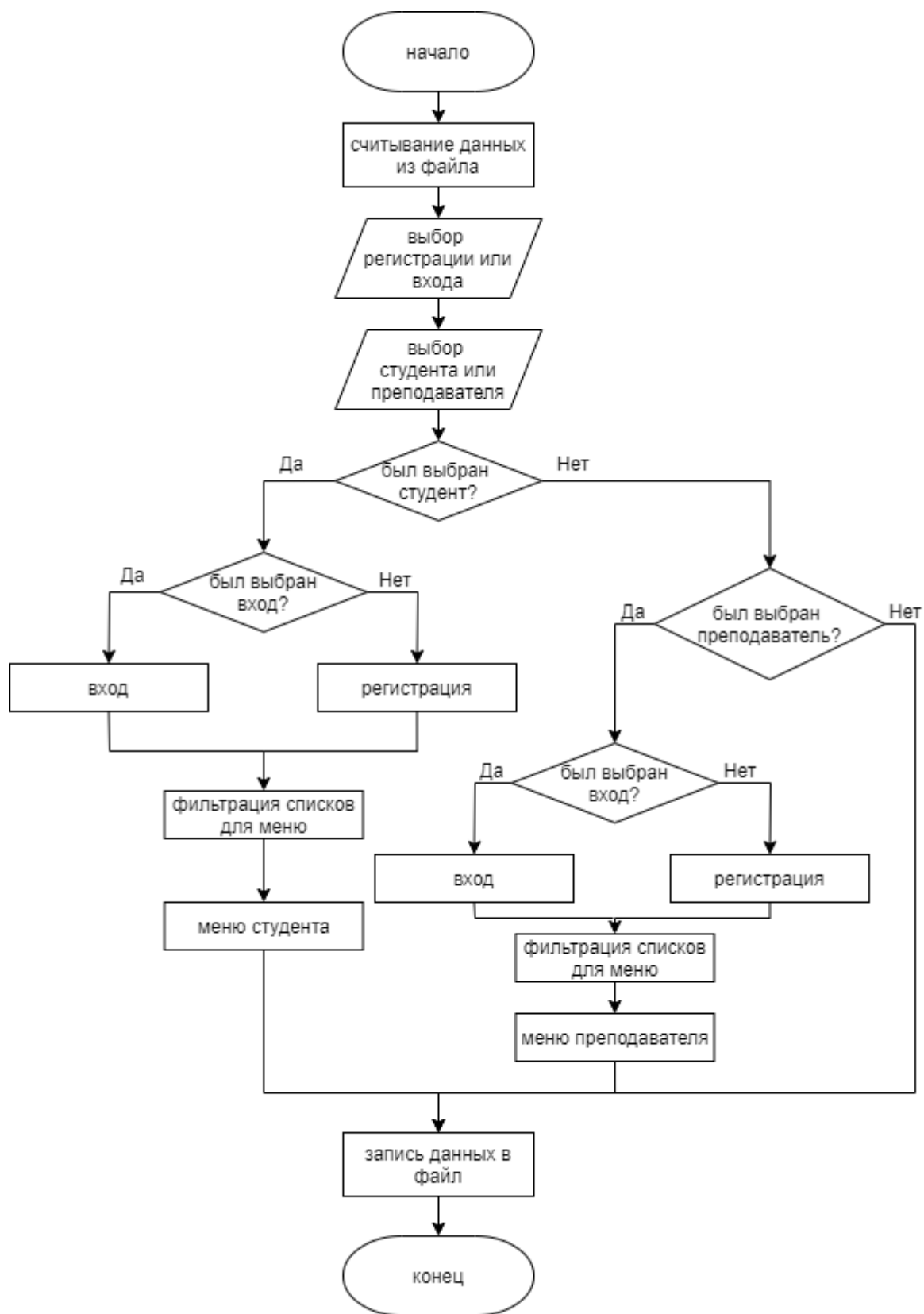


Рисунок Г1 – схема алгоритма работы всей программы

Продолжение приложения Г

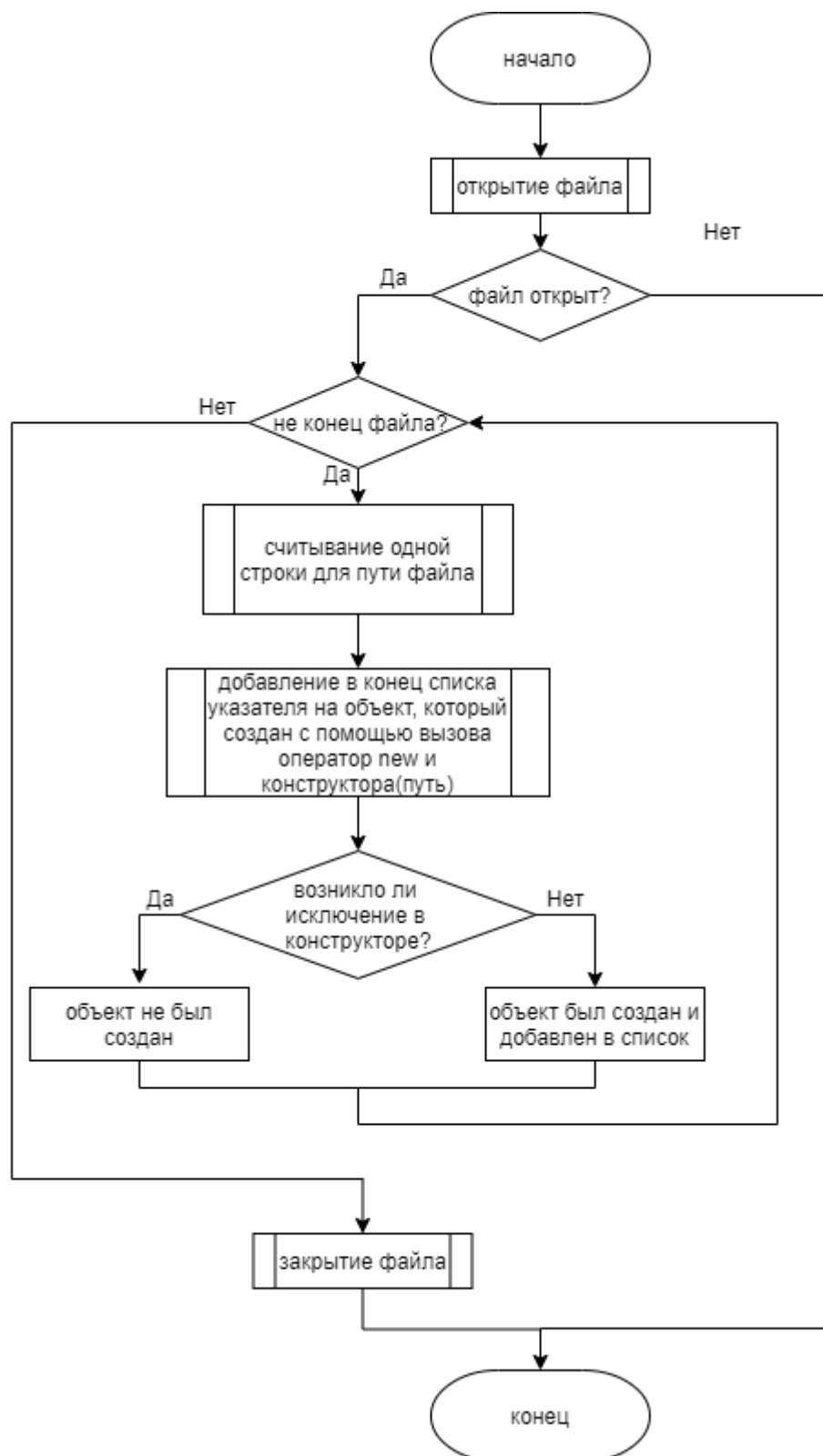


Рисунок Г2 – схема алгоритма чтения из файла

Продолжение приложения Г



Рисунок ГЗ – схема алгоритма записи в файл