Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра электронных вычислительных машин
Дисциплина: Программирование на языках высокого уровня

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

СИСТЕМА ТЕСТИРОВАНИЯ ЗНАНИЙ

БГУИР КП 1-40 02 01 014 ПЗ

Студент: гр. 250541, Р.Е. Власов

Руководитель: ассистент каф. ЭВМ, А.В. Марзалюк

МИНСК 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Образование неразрывно связано с применением информационных технологий, которые играют ключевую роль в повышении качества учебного процесса. Одним из наиболее эффективных инструментов автоматизации обучения является использование систем тестирования знаний. Такие системы позволяют не только объективно оценивать уровень подготовки учащихся, но и значительно упрощают процесс проведения контрольных мероприятий.

Система тестирования знаний представляет собой программное обеспечение, предназначенное для создания, управления и анализа тестов. Она используется для проверки теоретических знаний, навыков и умений студентов в различных образовательных учреждениях. Преимущества таких систем очевидны: автоматизация проверки ответов, снижение человеческого фактора при оценивании, мгновенное предоставление результатов и возможность анализа ошибок.

Кроме того, такие системы способствуют индивидуализации образовательного процесса, предоставляя возможность адаптации тестов под особенности каждого учащегося. Они обеспечивают удобство и гибкость как для преподавателей, так и для обучающихся, позволяя проводить тестирование в удаленном формате или в классе. Внедрение таких технологий значительно экономит время на проверку и обработку данных, а также позволяет преподавателям сосредоточиться на анализе результатов и улучшении учебного процесса.

1 ОБЗОР ЛИТЕРАТУРЫ

Образование играет ключевую роль в развитии общества, определяя его интеллектуальный и профессиональный потенциал. Сегодня доступны многочисленные ресурсы, которые упрощают и автоматизируют проверку знаний. Такие платформы, как Google Forms и Moodle, уже стали незаменимыми инструментами для студентов, преподавателей и образовательных учреждений. Они позволяют создавать тесты, собирать данные, анализировать результаты и предоставляют удобные инструменты для взаимодействия между участниками образовательного процесса. Каждая из этих систем обладает своими преимуществами и особенностями, которые делают их востребованными в образовательной среде.

Google Forms – это бесплатный онлайн-сервис для создания опросов и тестов, интегрированный с экосистемой Google. Он стал одним из наиболее популярных инструментов для проведения тестирования благодаря своей простоте и доступности. Исходя из источника [1] его преимущества включают в себя:

1) Простота использования. Google Forms позволяет создавать тесты минимальными знаниями в области IT, благодаря интуитивно понятному интерфейсу.

2) Результаты автоматически сохраняются в таблице Google Sheets, что упрощает анализ данных.

3) Поддержка множества типов вопросов, включая выбор вариантов, ввод текста, шкалы и другие.

4) Платформа бесплатна и доступна из любой точки мира, где есть интернет.

Moodle – это открытая система управления обучением (LMS), предоставляющая широкий спектр инструментов для организации учебного процесса, включая тестирование знаний. Согласно источнику [2] она была разработана в 2002 году и широко используется в учебных заведениях и компаниях. Преимущества Moodle включают в себя:

1) Система позволяет настраивать тесты под конкретные потребности, включая адаптивное тестирование и случайный выбор вопросов.

2) В тестах можно использовать изображения, видео и аудио для создания интерактивных заданий.

3) Широкий функционал. Помимо тестирования, Moodle предоставляет инструменты для общения, обмена материалами и оценки прогресса.

4) Платформа автоматически собирает данные о результатах тестирования, предоставляя подробные отчеты для анализа.

5) Доступность на разных устройствах.

Moodle внес множество новшеств в области образовательных технологий, включая гибкость настройки тестов и интеграцию мультимедиа. Благодаря своим возможностям Moodle остается одним из ведущих инструментов для тестирования знаний.

1.1 Обзор методов и алгоритмов решения поставленной задачи

Разработка системы тестирования знаний требует применения различных методов и алгоритмов, обеспечивающих эффективность, надежность и удобство использования программы. Для формирования и управления тестами можно использовать несколько подходов.

Файловая система хранения данных предполагает сохранение тестов и их параметров в текстовых или бинарных файлах. Этот метод прост в реализации, но требует проработки структуры данных и алгоритмов обработки. Альтернативой является использование реляционных баз данных, которые обеспечивают высокую скорость поиска и возможность работы с большими объемами информации. Еще одним популярным подходом является применение форматов JSON или XML, которые позволяют хранить данные в структурированном виде, легко читаются и масштабируются. Было выбрано сохранение данных в бинарные файлы, так как этот подход позволяет существенно сократить объем хранимой информации за счет минимизации избыточности. Кроме того, данный метод предоставляет возможность эффективной сериализации сложных объектов, упрощая процесс их восстановления и использования в приложении.

Исходя из источника [3] существует множество видов вопросов, используемых для тестирования знаний, включая вопросы с выбором одного правильного ответа, нескольких правильных вариантов, вопросы с открытым ответом, на установление соответствия и на упорядочивание. Были выбраны три основных типа: вопросы с выбором одного ответа, нескольких вариантов и с открытым ответом. Такой подход обеспечивает универсальность тестирования, позволяет адаптировать задания под разные форматы проверки знаний и способствует более глубокому анализу уровня подготовки участников.

Проблема проверки открытых вопросов связана с тем, что ответы участников могут сильно различаться из-за синонимов, опечаток или вариаций формулировок. Для её решения возможны разные подходы: проверка на полное совпадение, которая требует точности, но не учитывает ошибок; использование алгоритма Левенштейна, позволяющего учитывать небольшие отклонения; анализ на основе ключевых слов, проверяющий наличие важных терминов; и синонимический анализ, оценивающий смысловую близость ответа. Был выбран алгоритм Левенштейна, так как он балансирует между гибкостью и точностью проверки, позволяя объективно оценивать ответы с незначительными ошибками.

Разграничение прав доступа в приложении требует обеспечения разных режимов работы для студентов и администраторов. Студенты не должны иметь возможность редактировать тесты, создавать новые задания или просматривать правильные ответы. Для решения этой задачи можно использовать следующие подходы: разграничение прав доступа с помощью ролей, где администратор имеет полный функционал, а студент — только доступ к прохождению тестов; использование отдельных интерфейсов для

каждой роли; аутентификацию на основе уникальных ключей. Был выбран подход с использованием уникального ключа, который хранится на флешке администратора, а его хэш вычисляется программой для подтверждения уровня доступа. Это гарантирует, что только администратор сможет управлять тестами, сохраняя доступ студентов строго в рамках их роли.

1.2 Обзор информационных технологий решения задачи

В разработанном проекте были применены следующие решения, каждое из которых основано на их преимуществах и соответствии требованиям системы:

1) Языки программирования. Разработка системы тестирования знаний требует выбора языка программирования, который будет подходящим для реализации функционала программы. В данном случае предпочтение отдается языку C++ благодаря его высокой производительности, поддержке объектно-ориентированного программирования и обширной библиотечной базе.

2) Системы управления базами данных (СУБД). Для хранения информации о тестах и результатах тестирования используются СУБД. Они обеспечивают надежное и масштабируемое хранение данных. Наиболее подходящими вариантами являются:

– SQLite – легковесная и простая в использовании встроенная база данных, которая идеально подходит для настольных приложений.

– MySQL и PostgreSQL – мощные реляционные базы данных, которые подходят для работы с большими объемами данных и сложными структурами.

В данном случае системы управления базами данных в ходе разработки приложения не используются.

3) Форматы хранения данных. Для хранения тестов и их структуры можно использовать файлы в форматах JSON или XML. Исходя из источника [4] эти форматы предоставляют удобные средства для структурированного хранения информации:

JSON (JavaScript Object Notation) – простой и легко читаемый формат данных, который широко используется для передачи и хранения структурированной информации.

XML (Extensible Markup Language) – более универсальный формат, поддерживающий расширенные функции, такие как валидация данных с использованием схем.

4) Инструменты для создания пользовательских интерфейсов. Для реализации интерфейса системы используются следующие технологии:

– Консольный интерфейс. Подходит для минималистичных решений, когда требуется простота и функциональность.

– Библиотеки GUI (графический интерфейс) для создания удобного пользовательского интерфейса на C++ широко используются библиотеки, такие как Qt.

В ходе конструирования программы был выбран консольный интерфейс ввиду простоты и лаконичности.

5) Алгоритмы анализа данных. Для проверки знаний и анализа результатов тестирования применяются алгоритмы, которые поддерживаются стандартными библиотеками C++ (например, STL) и специализированными библиотеками:

1. Стандартная библиотека шаблонов (STL): набор контейнеров и алгоритмов для работы с данными, таких как сортировка, поиск и подсчет.

2. Алгоритмы обработки строк для анализа текстовых ответов используются алгоритмы строковых расстояний (например, алгоритм Левенштейна) и алгоритмы поиска подстрок (например, Кнут-Моррис-Пратт).

6) Среды разработки (IDE). Выбор среды разработки играет важную роль в удобстве написания, тестирования и отладки кода. Наиболее популярные среды для разработки на C++:

1. Visual Studio - мощная IDE с инструментами для разработки, отладки и тестирования программ.

2. CLion - кроссплатформенная IDE, поддерживающая современный C++ и предлагающая инструменты для анализа и рефакторинга кода.

3. Code::Blocks - бесплатная и легковесная IDE, подходящая для простых проектов.

Для написания программы была выбрана среда разработки Visual Studio.

1.3 Постановка задачи

Написать приложение «Система тестирования знаний», в котором должна быть реализована система, позволяющая делать тесты, с вопросами: «один из нескольких вариантов», «несколько из нескольких вариантов», «ввод своего варианта». Реализовать возможность управлять информацией в системе, а также получать статистику пользователю и администратору.

В программе должны быть реализованы следующие основные методы:

1) создание тестов;
2) редактирование тестов;
3) удаление тестов;
4) прохождение тестов;
5) сохранение статистики;
6) удаление статистики;
7) хранение данных в бинарных файлах;
8) верификация администратора с использованием уникального ключа;
9) проверка корректности пользовательского ввода;
10) обработка ошибок при работе с файлами и некорректных входных данных.

2 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

2.1 Структура входных и выходных данных

Файлы тестов сохраняются в каталоге «tests» в бинарном виде и имеют расширение .ktst (Knowledge Testing System Test). Каждый файл представляет собой отдельный тест. В начале файла указывается название теста, после чего друг за другом следуют вопросы. Так вопросы имеют различные структуры, в начале каждого вопроса указывается его тип. Структуры данных «Тест», «Вопрос с выбором одного варианта», «Вопрос с выбором нескольких вариантов» и «Вопрос с открытым ответом» представлены в таблицах 2.1, 2.2, 2.3 и 2.4 соответственно.

Таблица 2.1 – Структура данных «Тест».

№	Данные
1	Уникальный идентификатор
2	Количество вопросов
3	Список вопросов

Таблица 2.2 – Структура данных «Вопрос с выбором одного варианта».

№	Данные
1	Тип вопроса
2	Текст вопроса
3	Количество вариантов ответа
4	Текст каждого варианта
5	Индекс правильного ответа

Таблица 2.3 – Структура данных «Вопрос с выбором нескольких вариантов».

№	Данные
1	Тип вопроса
2	Текст вопроса
3	Количество вариантов ответа
4	Текст каждого варианта
5	Количество правильных ответов
6	Индексы правильных ответов

Таблица 2.4 – Структура данных «Вопрос с открытым ответом».

№	Данные
1	Тип вопроса
2	Текст вопроса
3	Правильный ответ

Статистика прохождения тестов сохраняется в бинарном файле stat.dat. Каждая запись содержит информацию о тесте, включая его структуру, так

как вопросы и варианты ответов перемешиваются при каждом прохождении. Все элементы записываются в бинарном виде с указанием их размера перед содержимым для обеспечения корректности считывания. Структура данных «Статистика» представлены в таблице 2.5.

Таблица 2.5 – Структура данных «Статистика».

№	Данные
1	Тест
2	Имя пользователя
3	Время прохождения теста
4	Количество ответов
5	Ответы пользователя
6	Количество правильных ответов

2.2 Разработка диаграммы классов

Диаграмма классов представлена в приложении А.

2.3 Описание классов

2.3.1 Class Test

Класс представляет собой модель теста с вопросами, которая включает функционал для работы с названием теста, добавлением, удалением и получением вопросов, а также возможности сохранения и загрузки тестов в/из файла.

Поля:

– `string title` – название теста.

– `List<Question*> questions` – список вопросов.

– `string fileName = ""` – имя файла.

Методы:

– `Test(const Test& other)` – копирующий конструктор, копирует данные из другого теста.

– `Test(const string& title)` – конструктор, создающий тест с заданным названием.

– `~Test()` – деструктор класса Test.

– `Test& operator = (const Test& other)` – оператор присваивания, копирует данные из другого теста.

– `const string& getTitle() const` – возвращает название теста.

– `int getQuestionCount() const` – возвращает количество вопросов в тесте.

– `void setFileName(const string& fileName)` – устанавливает имя файла.

– `void setTitle(const string& newTitle)` – устанавливает новое название теста.

– `void addQuestion(Question* question)` – добавляет вопрос в тест.

– `void removeQuestion(int index)` – удаляет вопрос из теста по индексу.

– `Question* getQuestion(int index)` – возвращает указатель на вопрос по указанному индексу.

– `string generateFileName() const` – генерирует имя файла для сохранения теста.

– `void saveToFile()` – сохраняет тест в файл.

– `void deleteTestFile()` – удаляет тест из файла.

– `static Test* loadFromFile(const string& fileName)` – загружает тест из файла.

– `static List<Test*> loadAllTests()` – загружает все тесты из текущей директории.

– `string sanitizeFileName(const string& input) const` –преобразует строку (название теста) в безопасное для файловой системы имя, с транслитерацией кириллицы.

2.3.2 Class Menu

Класс Menu управляет общим интерфейсом приложения, предоставляя пользователю доступ к отображению меню и переход между интерфейсами.

Поля:

– `List<Test*> tests` – список указателей на объекты класса Test.

Методы:

– `Menu()` – конструктор класса Menu.

– `~Menu()` – деструктор класса Menu.

– `void displayRoleSelectionMenu()` – выводит меню выбора роли.

– `void displayAdminMenu()` – отображает меню администратора.

– `void displayStudentMenu(const string& studentName)` – отображает меню ученика.

– `void checkStudentName(const string& studentName)` – проверяет корректность введённого ФИО ученика.

– `static int getValidatedNumber(const string& input, int min, int max)` – проверяет корректность и диапазон введённого числа.

– `static char getInput(const string& validOptions)` – читает пользовательский ввод и проверяет, соответствует ли он допустимым символам.

– `void startTesting()` – позволяет ученику выбрать тест из

списка и начать тестирование.

– `void displayTestList()` – выводит список всех тестов с их заголовками.

– `void addTest()` – добавляет новый тест в список tests.

– `void addQuestionsToTest(Test* newTest)` – добавляет вопросы в новый или существующий тест.

– `void addSingleChoiceQuestion(Test* test)` – добавляет вопрос с одним правильным ответом.

– `void addMultipleChoiceQuestion(Test* test)` – добавляет вопрос с несколькими правильными ответами.

– `void addOpenAnswerQuestion(Test* test)` – добавляет вопрос с открытым (свободным) ответом.

– `void deleteTest()` – удаляет выбранный тест из списка tests.

– `void editTest()` – редактирует существующий тест.

– `void editTestDetails(Test* test)` – предоставляет подробное редактирование выбранного теста, включая изменения вопросов.

– `bool saveTest(Test* test)` – сохраняет тест в файл.

– `void changeQuestion(SingleChoiceQuestion* question)` – редактирует вопрос с одним правильным ответом.

– `void changeQuestion(MultipleChoiceQuestion* question)` – редактирует вопрос с несколькими правильными ответами.

– `void changeQuestion(OpenAnswerQuestion* question)` – редактирует вопрос с открытым (свободным) ответом.

– `bool setQuestionTitle(Question* question)` – устанавливает или изменяет заголовок вопроса.

– `bool setQuestionOptions(ChoiceQuestion* question)` – устанавливает или изменяет варианты ответа для вопроса с выбором.

– `bool handleExistingOption(ChoiceQuestion* question, char& option, unsigned int index)` – обрабатывает уже существующий вариант ответа.

– `bool replaceOption(ChoiceQuestion* question, char option, unsigned int index)` – заменяет текст существующего варианта ответа.

– `bool handleNewOption(ChoiceQuestion* question, char& option)` – обрабатывает добавление нового варианта ответа.

– `bool setAnswer(SingleChoiceQuestion* question)` – устанавливает правильный ответ для вопроса с одним правильным вариантом.

– `bool setAnswer(MultipleChoiceQuestion* question)` – устанавливает правильные ответы для вопроса с несколькими правильными вариантами.

– `bool setAnswer(OpenAnswerQuestion* question)` – устанавливает правильный ответ для вопроса с открытым ответом.

– `void changeTestTitle(Test* test)` – изменяет заголовок теста.

– `void deleteQuestionFromTest(Test* test)` – удаляет вопрос из теста.

– `void editQuestionInTest(Test* test)` – позволяет редактировать выбранный вопрос.

– `int printTestList()` – печатает список тестов и возвращает их количество.

– `void printQuestion(Question* question)` – печатает текст вопроса и его правильный ответ.

– `void sortTests()` – сортирует тесты по названию.

2.3.3 Class StatisticMenu

Класс StatisticMenu управляет взаимодействием с записями статистики, предоставляя пользователю данные о результатах тестирования.

Поля:

– `List<Statistic*> statistics` – список указателей на объекты класса Statistic. Хранит статистику выполнения тестов.

Методы:

– `StatisticMenu()` – конструктор класса StatisticMenu.

– `~StatisticMenu()` – деструктор класса StatisticMenu.

– `void start()` – запускает главное меню статистики.

– `void displayStatisticList()` – отображает список всех записей статистики.

– `void deleteStatistic()` – позволяет удалить выбранную запись статистики.

– `int printStatisticList()` – выводит таблицу со списком записей статистики, включая номера, названия тестов, имена тестируемых и дату.

– `void saveStatistics()` – сохраняет текущий список статистики в файл.

– `void sortStatistics()` – сортирует записи статистики по названию тестов.

2.3.4 Class TestSession

Класс TestSession управляет процессом тестирования.

Поля:

– `Test* test` – указатель на объект Test.

– `string name` – имя тестируемого пользователя.

– `List<string> userAnswers` – список ответов пользователя.

– `int correctAnswers` – количество правильных ответов, данных пользователем.

– `int score` – итоговый результат пользователя в процентах.

Методы:

– `TestSession(Test* test, string name)` – конструктор класса TestSession.

– `void start()` – запускает тестирование.

– `bool prepare()` – проверяет наличие теста и имени пользователя, выводит информацию о тесте, запрашивает подтверждение начала тестирования, перемешивает вопросы в тесте.

– `void askQuestion(Question* question, int number)` – выводит текст текущего вопроса и запрашивает ответ пользователя.

– `void showResult()` – вычисляет итоговый процент правильных ответов.

2.3.5 Class Statistic

Класс Statistic управляет данными о прохождении теста, обеспечивая сбор, хранение и обработку информации о результатах пользователей. Поля:

– `Test* test` – указатель на объект Test, содержащий информацию о тесте.

– `string name` – имя тестируемого пользователя.

– `string time` – дата и время прохождения теста.

– `List<string> answers` – список ответов пользователя на вопросы теста.

– `int correctAnswersCount` – количество правильных ответов, данных пользователем.

– `int score` – итоговый результат пользователя.

Методы:

– `Statistic(const string name, Test* test)` – конструктор класса Statistic.

– `void setCurrentTime()` – устанавливает время прохождения теста вручную.

– `void setAnswers(const List<string>& answers)` – устанавливает список ответов пользователя.

– `void setTime(const string time)` – устанавливает текущее системное время как время прохождения теста.

– `void setCorrectAnswersCount(int count)` – устанавливает количество правильных ответов и вычисляет итоговый результат

– `string getTestTitle()` – возвращает название теста.

– `string getUserName()` – возвращает имя пользователя.

– `string getTime()` – возвращает время прохождения теста.

– `void print()` – выводит подробную информацию о тесте.

– `void saveToFile()` – сохраняет объект статистики в файл.

– `static Statistic* loadFromStream(ifstream& in)` – загружает одну запись статистики из потока данных файла.

– `static List<Statistic*> loadAllFromFile()` – загружает все записи статистики из файла в список объектов Statistic.

2.3.6 Class Question

Класс является базовым классом для всех типов вопросов, таких как вопросы с одним выбором, с множественным выбором и вопросы с открытым ответом.

Поля:

– `string title = ""` – название вопроса.

Методы:

– `Question(const Question& other)` – копирующий конструктор, копирует заголовок из другого вопроса.

– `Question(const string& title)` – конструктор, инициализирует вопрос с заданным заголовком.

– `Question& operator=(const Question& other)` – оператор присваивания, копирует заголовок из другого вопроса.

– `virtual ~Question() = default` – деструктор класса Question.

– `virtual QuestionType getType() const = 0` – возвращает тип вопроса.

– `virtual void printQuestion() = 0` – выводит заголовок и список вариантов ответа.

– `virtual bool checkAnswer(const string answer) const = 0` – проверяет, правильно ли указан ответ пользователя.

– `virtual void saveToStream(ofstream& out) = 0` – сохраняет вопрос в поток.

– `static Question* loadFromStream(ifstream& in)` – загружает вопрос из потока.

– `void setTitle(const string& title)` – устанавливает заголовок вопроса.

– `string getTitle() const` – возвращает заголовок вопроса.

– `static bool isAlpha(unsigned char ch)` – проверяет, является ли символ буквой.

– `static bool isSymbol(unsigned char ch)` – проверяет, является ли символ допустимым специальным символом.

2.3.7 Class SingleChoiceQuestion

Класс представляет собой вопрос с одним правильным вариантом

ответа.

Поля:

– `char correctIndex = 'A'` – буква, указывающая на правильный вариант ответа.

Методы:

– `SingleChoiceQuestion(const SingleChoiceQuestion& other)` – копирующий конструктор, копирует заголовок, варианты ответов и правильный индекс из другого объекта.

– `SingleChoiceQuestion(const string& title, const List<string>& options, char correctIndex)` – конструктор, инициализирует вопрос с заголовком, списком вариантов и правильным индексом.

– `SingleChoiceQuestion& operator=(const SingleChoiceQuestion& other)` – оператор присваивания, копирует данные из другого объекта, включая правильный индекс.

– `QuestionType getType() const override` – возвращает тип вопроса.

– `void printQuestion() override` – выводит заголовок и список вариантов ответа.

– `bool checkAnswer(const string answer) const override` – проверяет, является ли переданный ответ правильным.

– `void setCorrectIndex(char correctIndex)` – устанавливает правильный вариант ответа.

– `void saveToStream(ofstream& out) override` – сохраняет вопрос в поток.

– `static SingleChoiceQuestion loadFromStream(ifstream& in)` – загружает вопрос из потока.

2.3.8 Class MultipleChoiceQuestion

Класс QuestionMultiple представляет собой структуру для хранения и управления вопросом с несколькими правильными ответами.

Поля:

– `List<unsigned char> correctIndices` – список правильных ответов, представленных буквами латинского алфавита.

Методы:

– `MultipleChoiceQuestion(const MultipleChoiceQuestion& other)` – копирующий конструктор, копирует заголовок, варианты ответов и правильные индексы из другого объекта.

– `MultipleChoiceQuestion(const string& title, const List<string>& options, const List<unsigned char>&`

correctIndices) – инициализирует объект с заголовком, списком вариантов и списком правильных индексов, выполняя валидацию.

– MultipleChoiceQuestion& operator=(const MultipleChoiceQuestion& other) – оператор присваивания, копирует данные из другого объекта, включая правильные индексы.

– QuestionType getType() const override – возвращает тип вопроса.

– bool checkAnswer(const string answer) const override – проверяет, правильно ли указаны ответы пользователя.

– void printQuestion() override – выводит заголовок и список вариантов ответа.

– void setCorrectIndices(const List<unsigned char>& options) – устанавливает правильные ответы.

– void saveToStream(ofstream& out) override – сохраняет вопрос в поток.

– static MultipleChoiceQuestion loadFromStream(ifstream& in) – загружает вопрос из потока.

2.3.9 Class OpenAnswerQuestion

Класс представляет вопрос с открытым ответом, где пользователь Должен ввести ответ в свободной форме.

Поля:

– string correctAnswer – правильный ответ на вопрос.

Методы:

– OpenAnswerQuestion(const OpenAnswerQuestion& other) – копирующий конструктор, копирует заголовок и правильный ответ из другого объекта.

– OpenAnswerQuestion(const string& title, const string& correctAnswer) – инициализирует объект с заголовком и правильным ответом, проверяя корректность данных.

– OpenAnswerQuestion& operator = (const OpenAnswerQuestion& other) – оператор присваивания, копирует данные из другого объекта, включая правильный ответ.

– QuestionType getType() const override – возвращает тип вопроса.

– void printQuestion() override – выводит заголовок вопроса.

– bool checkAnswer(const string answer) const override – проверяет, совпадает ли введённый пользователем ответ с правильным ответом.

– void setAnswer(const string answer) – устанавливает правильный ответ.

– `void saveToStream(ofstream& out) override` – сохраняет вопрос в поток.

– `static OpenAnswerQuestion loadFromStream(ifstream& in)` – загружает вопрос из потока.

– `int levenshteinDistance(const string& str1, const string& str2) const` – вычисляет расстояние Левенштейна между двумя строками.

– `unsigned char toLower(unsigned char) const` – приводит символы к нижнему регистру.

2.3.10. Class ChoiceQuestion

Класс ChoiceQuestion управляет вариантами ответа, предоставляет методы для их добавления, удаления и получения.

Поля:

– `List<string> options = {}` – список вариантов ответа.

Методы:

– `ChoiceQuestion(const ChoiceQuestion& other)` – копирующий конструктор, копирует заголовок и все варианты ответа из другого объекта.

– `ChoiceQuestion(const string& title, const List<string>& options)` – инициализирует объект с заголовком и списком вариантов, добавляя каждый вариант с проверкой.

– `ChoiceQuestion& operator=(const ChoiceQuestion& other)` – оператор присваивания, копирует заголовок и варианты ответа из другого объекта, предварительно очищая текущий список.

– `virtual ~ChoiceQuestion() = default` – деструктор класса ChoiceQuestion.

– `List<string> getOptions() const` – возвращает копию списка всех вариантов ответа.

– `string getOption(unsigned int index)` – возвращает вариант ответа по заданному индексу.

– `int getOptionCount() const` – возвращает количество вариантов ответа.

– `void addOption(const string& option, int index = 1)` – добавляет новый вариант ответа с проверкой на корректность.

– `void removeOption(unsigned int index)` – удаляет вариант ответа по заданному индексу, проверяя его корректность.

2.3.11 Class KeyVerifier

Класс предназначен для проверки наличия ключа на съемных устройствах.

Поля: нет.

Методы:

– `static bool verify()` – проверяет наличие ключа на всех доступных съемных устройствах.

– `static bool isRemovableDrive(const string& drivePath)` – проверяет, является ли указанный диск съемным.

– `static uint32_t getHash(const string& data)` – вычисляет хеш для переданных данных.

2.3.12 Class FileError

Класс KeyVerifier предназначен для проверки наличия ключа на съёмных носителях, обеспечивая защиту доступа к административным функциям приложения.

Поля:

–`string message_` – строка, содержащая сообщение об ошибке, которое будет выведено при возникновении исключения.

Методы:

– `explicit FileError(const string& message) : message_(message) {}` – конструктор класса FileError.

– `const char* what() const noexcept override` – возвращает строку с сообщением об ошибке, которое было передано в конструктор.

2.3.13 Class InputError

Класс используется для обработки ошибок, связанных с неправильным вводом данных.

Поля:

–`string message_` – строка, содержащая сообщение об ошибке, которое будет выведено при возникновении исключения.

Методы:

– `explicit InputError(const string& message) : message_(message) {}` – конструктор класса InputError.

– `const char* what() const noexcept override` – возвращает строку с сообщением об ошибке, которое было передано в конструктор.

2.3.14 Class List<T>

Класс представляет собой двусвязный список, который поддерживает операции вставки, удаления, поиска и другие операции с элементами.

Поля:

– `int size` – количество элементов в списке.

– `Node<T> *head` – указатель на первый элемент списка.

– `Node<T> *tail` – указатель на последний элемент списка.

Методы:

– `List(const T* data, const int size)` – конструктор, который инициализирует список данными из массива.

– `List()` – конструктор по умолчанию, инициализирует пустой список.

– `List(initializer_list<T> data)` – конструктор, который инициализирует список с использованием списка инициализации.

– `~List()` – деструктор класса.

– `T& front() const` – возвращает ссылку на первый элемент списка.

– `T& back() const` – возвращает ссылку на последний элемент списка.

– `T& operator[](const unsigned int index) const` – оператор доступа по индексу для получения элемента списка.

– `void setDefaultValues()` – инициализирует поля size, head и tail значениями по умолчанию.

– `void pushFront(const T& element)` – добавляет элемент в начало списка.

– `void pushBack(const T& element)` – добавляет элемент в конец списка.

– `void popFront()` – удаляет первый элемент списка.

– `void popBack()` – удаляет последний элемент списка.

– `void insert(const T& element, const unsigned int index)` – вставляет элемент в список по указанному индексу.

– `void removeAt(const unsigned int index)` – удаляет элемент по индексу.

– `void remove(const T& element, RemoveMode mode)` – удаляет элемент из списка.

– `int firstIndexOf(const T& element) const` – находит индекс первого вхождения элемента в список.

– `bool contains(const T& element) const` – проверяет, содержится ли элемент в списке.

– `void clear()` – очищает список.

– `bool isEmpty() const` – проверяет, пуст ли список.

– `int getSize() const` – возвращает размер списка.

– `Iterator<T> find(const T& data)` – находит первый элемент в списке, равный переданному значению.

– `Iterator<T> findAt(const unsigned int index)` находит элемент по индексу.

3 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

3.1 Разработка схем алгоритмов

В приложениях Б и В представлены схемы методов levenshteinDistance() класса OpenAnswerQuestion и verify() класса KeyVerifier соответственно.

3.2 Разработка алгоритмов (описание алгоритмов по шагам)

3.2.1 Алгоритм загрузки тестов из каталога loadAllTests()

Данный метод отвечает за загрузку всех тестов из указанного каталога.

Входные данные: каталог, указанный в PATH.

Выходные данные: список объектов типа Test*, содержащий успешно загруженные тесты.

1. Начало.
2. Объявить переменную tests типа List<Test*> и инициализировать ее пустым списком.
3. Объявить переменную extension типа string и присвоить значение ".ktst".
4. Объявить переменную dir типа DIR*.
5. Объявить переменную entry типа struct dirent*.
6. Открыть каталог PATH с помощью функции opendir() и присвоить результат переменной dir.
7. Если каталог не открылся, перейти к шагу 8. Если каталог открылся, перейти к шагу 9.
8. Создать каталог с помощью _mkdir(). Если создание не удалось, выбросить исключение FileError и вернуть пустой список tests. Перейти к шагу 12.
9. Прочитать запись из каталога с помощью readdir() до тех пор, пока не достигнут конец каталога.
10. Если имя файла заканчивается на .ktst, перейти к шагу 11. Если не заканчивается, перейти к шагу 9.
11. Попытаться загрузить тест из файла и добавить его в список tests. Перейти к шагу 9.
12. Закрыть каталог с помощью closedir().
13. Вернуть список tests.
14. Конец.

3.2.2 Алгоритм пирамидальной сортировки тестов sortTests()

Данный метод сортирует список тестов tests по заголовкам в лексикографическом порядке.

Входные данные: список tests, содержащий объекты тестов.

Выходные данные: список tests, отсортированный по возрастанию заголовков.

1. Начало.
2. Получить размер списка tests и сохранить его в переменной n.
3. Определить лямбда-функцию heapify, которая перестраивает список tests.

Входные данные: список tests, размер списка n, индекс текущего элемента i.

Выходные данные: перестроенный список tests.

3.1 Начало
3.2 Установить индекс как индекс наибольшего элемента: largest = i.
3.3 Рассчитать индекс левого потомка: left = 2 * i + 1.
3.4 Рассчитать индекс правого потомка: right = 2 * i + 2.
3.5 Если left < n и tests[left] > tests[largest], то largest = left.
3.6 Если right < n и tests[right] > tests[largest], то largest = right.
3.7 Если largest == i перейти к шагу 3.10.
3.8 Поменять местами элементы tests[i] и tests[largest].
3.9 Рекурсивно вызвать heapify(tests, n, largest).
3.10 Конец.
4. Определить переменную i для хранения текущего индекса.
5. Установить индекс последнего узла с потомками (i = n / 2 - 1).
6. Если i < 0, перейти к шагу 10.
7. Вызвать функцию heapify(tests, n, i).
8. Уменьшить значение i на 1.
9. Перейти к шагу 6.
10. Установить индекс на последний элемент списка tests (i = n - 1).
11. Если i < 0, перейти к шагу 16.
12. Поменять местами элементы tests[0] и tests[i].
13. Уменьшить значение i на 1.
14. Вызвать функцию heapify(tests, i, tests[0]).
15. Перейти к шагу 11.
16. Конец.

4 РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

В результате работы было написано приложение, главное меню которого представлено на рисунке 4.1. Для запуска программы необходимо запустить файл TestingSystem.exe, в папке “tests” храняться файлы с тестами.

Рисунок 4.1 – Меню выбора роли

После выбора роли "Ученик" пользователю предлагается ввести свои фамилию, имя и отчество. Пример интерфейса для ввода данных представлен на рисунке 4.2.

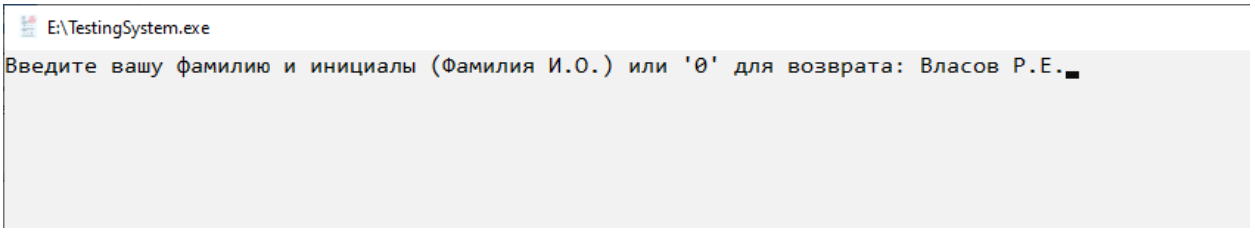

Рисунок 4.2 – Интерфейс ввода ФИО

После ввода ФИО отображается меню ученика, предоставляющее доступ к прохождению тестов, как показано на рисунке 4.3.

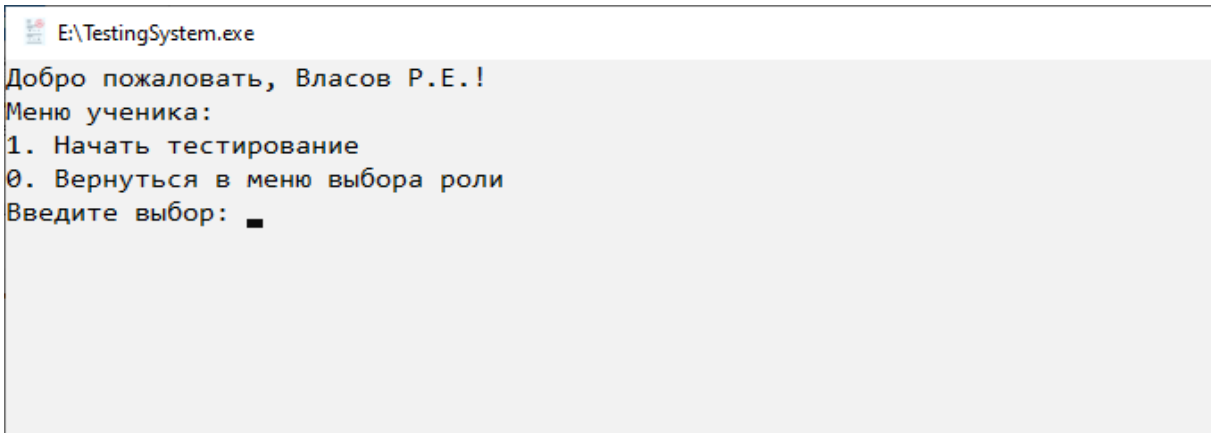

Рисунок 4.3 – Меню ученика

После выбора “Начать тестирование” пользователю предлагается выбрать тест из доступного списка. После выбора требуется подтвердить свой выбор для начала тестирования, как показано на рисунке 4.4.

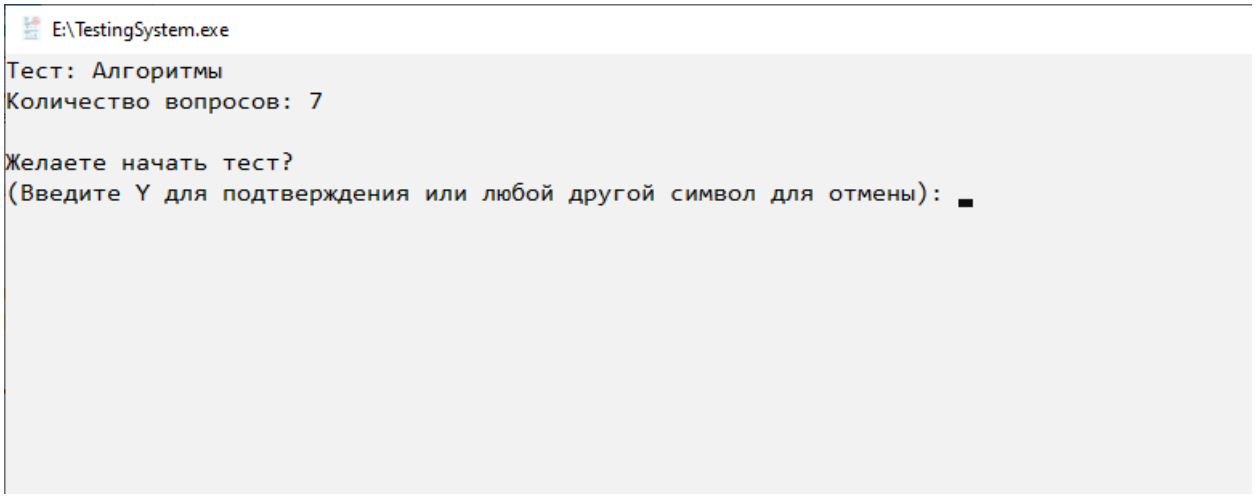

Рисунок 4.4 – Подтверждение выбора теста

Пример вопроса, отображаемого в процессе тестирования, включает текст задания, который представляет собой четко сформулированный вопрос или инструкцию для пользователя. Под текстом располагается список вариантов ответа, представленных в виде буквенных обозначений с пояснениями или описаниями возможных решений. Дополнительно предусмотрено поле для ввода ответа, позволяющее пользователю ввести правильные варианты или оставить его пустым при необходимости. Такой формат обеспечивает логичное представление информации и способствует удобству взаимодействия с системой тестирования. Графическое отображение подобного вопроса представлено на рисунке 4.5.

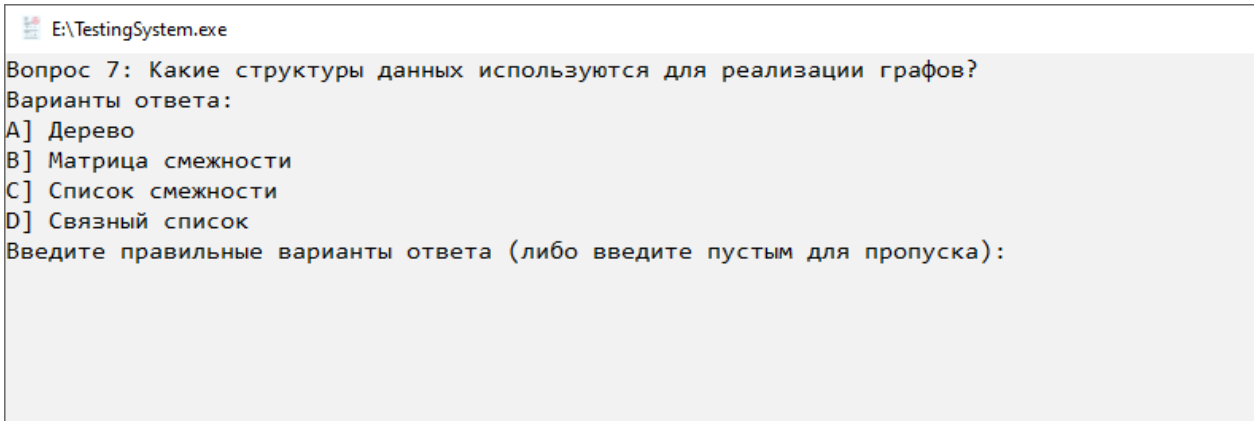

Рисунок 4.5 – Вопрос для ученика

После завершения тестирования система отображает итоговый результат, который включает общее количество вопросов, количество правильных ответов и процент успешности прохождения. Такой формат вывода позволяет пользователю оценить свой уровень выполнения задания и

понять, насколько успешно он справился с тестом. Графическое представление данного результата приведено на рисунке 4.6.

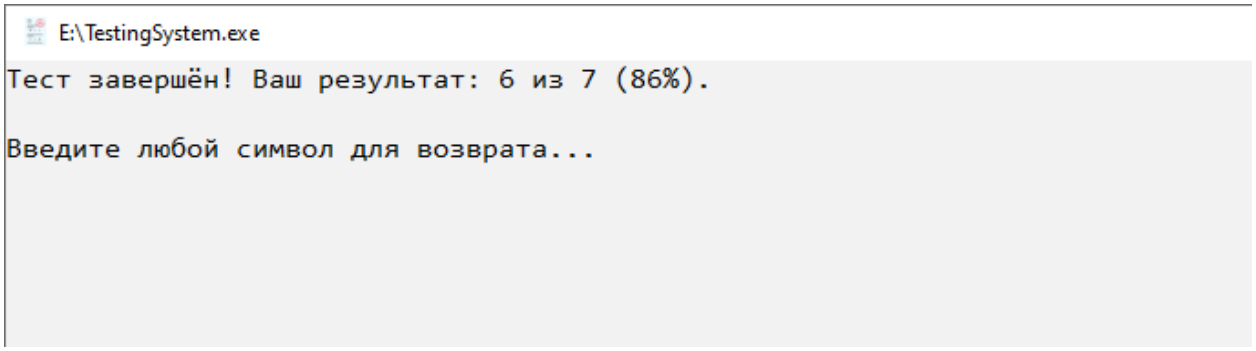

Рисунок 4.6 – Результаты теста

Если в начале работы программы была выбрана роль "Администратор", пользователю предоставляется доступ к специальному меню с функциями управления тестами и статистикой. Для активации режима администратора необходимо подключить съемный накопитель, на котором в корневом каталоге должен находиться файл kts_key.dat с заранее сгенерированным ключом доступа. Меню администратора содержит функции для просмотра списка тестов, добавления, удаления, редактирования тестов, а также анализа статистики прохождения тестов. При необходимости можно вернуться в главное меню выбора роли. Пример интерфейса меню администратора приведен на рисунке 4.7.

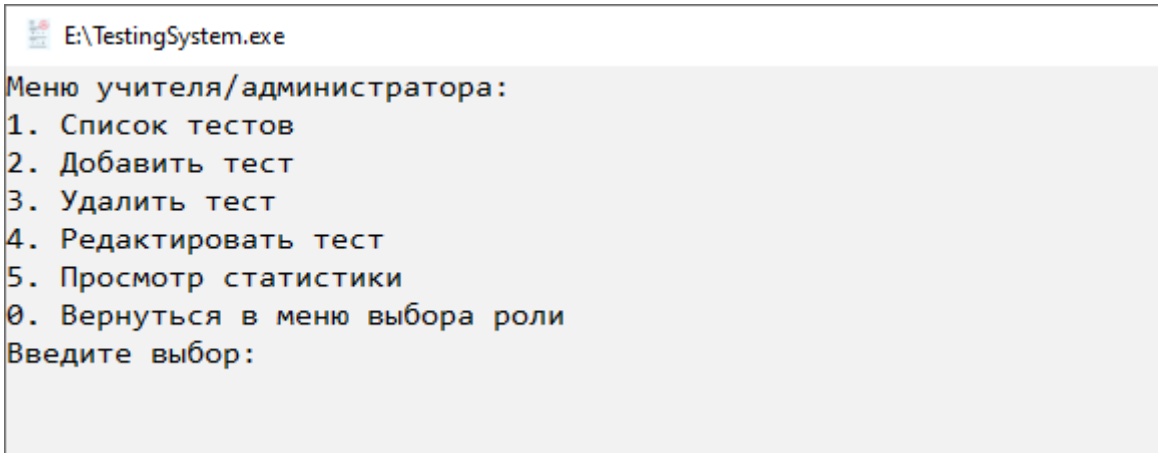

Рисунок 4.7 – Меню администратора

При выборе пункта "Список тестов" в меню администратора программа отображает перечень доступных тестов с их порядковыми номерами и названиями. Пользователю предлагается выбрать номер теста для дальнейшего просмотра его содержимого или управления им. Также предусмотрена возможность возврата в предыдущее меню путем ввода числа 0. Такой интерфейс обеспечивает наглядность и удобство взаимодействия при управлении тестами. Пример отображения списка тестов приведен на рисунке 4.8.

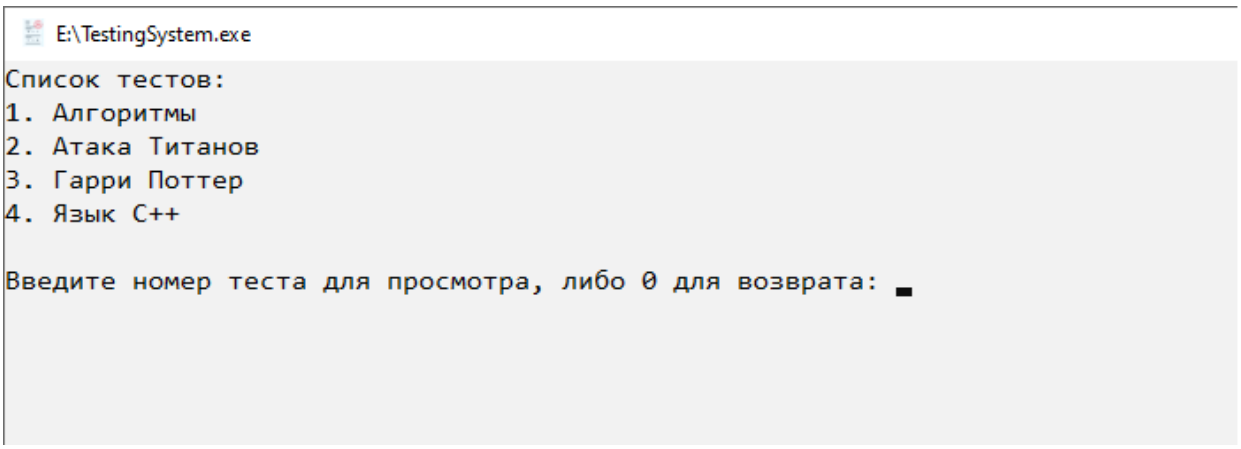

Рисунок 4.8 – Список тестов

В этом разделе "Список тестов" можно выбрать тест для просмотра его содержимого. Пример интерфейса для выбора и просмотра теста показан на рисунке 4.9.

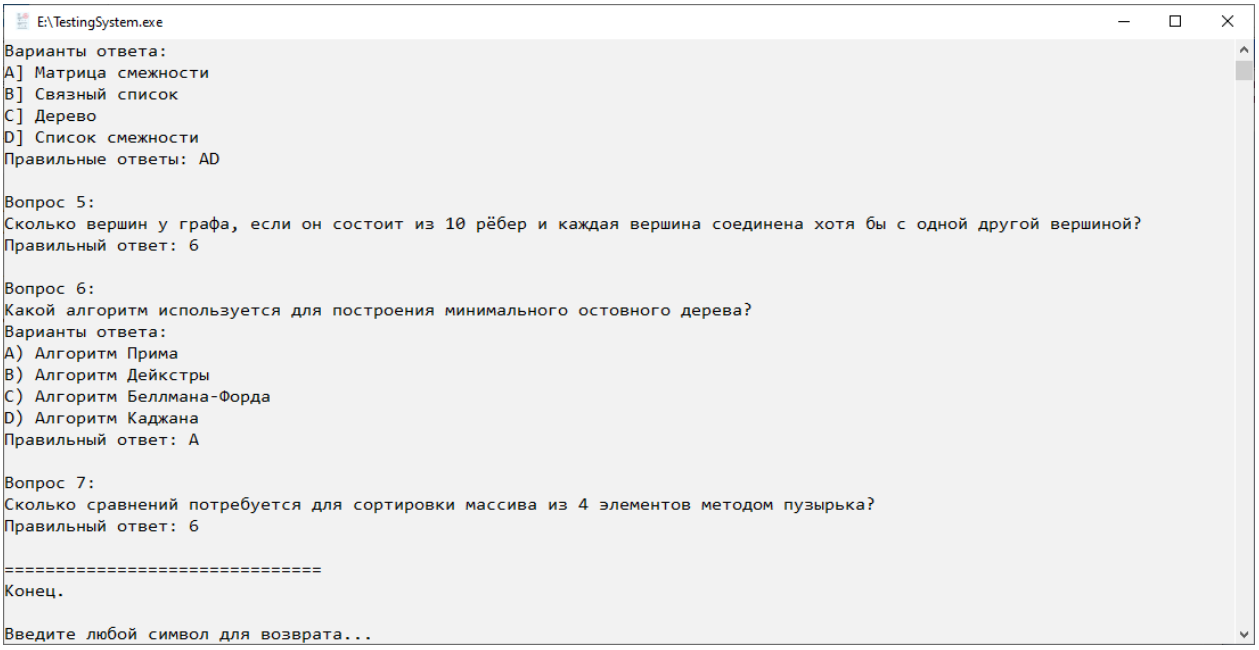

Рисунок 4.9 – Просмотр теста

В меню администратора предусмотрена возможность добавления новых тестов. Пример интерфейса для добавления названия теста, представлен на рисунке 4.10

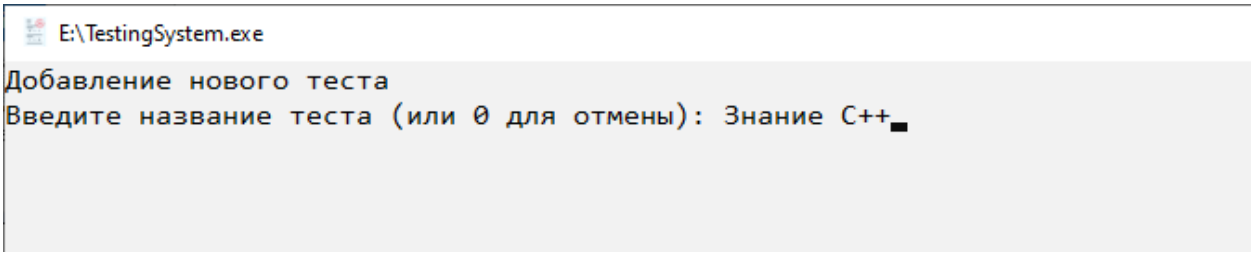

Рисунок 4.10 – Просмотр теста

В процессе добавления теста доступно меню добавления вопроса, где администратор может выбрать тип вопроса (с выбором одного варианта, нескольких вариантов или с открытым ответом). Пример интерфейса меню добавления вопроса показан на рисунке 4.11.

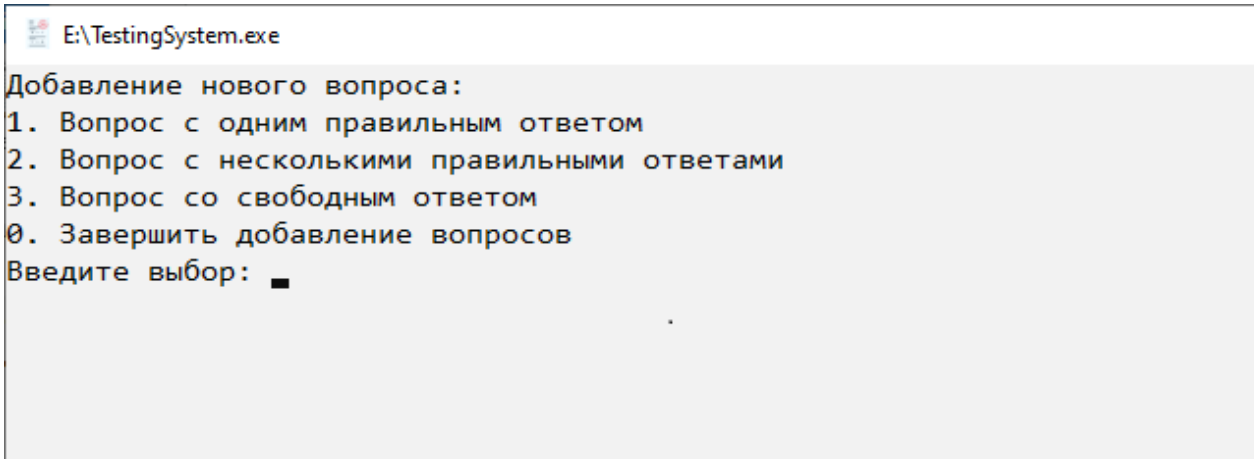

Рисунок 4.11 – Меню добавления вопроса

Для добавления вопроса с одним правильным ответом администратор вводит текст задания, формулирует несколько вариантов ответов и отмечает один из них как правильный. В процессе тестирования пользователю необходимо выбрать верный вариант, введя соответствующую букву. Пример интерфейса для создания и отображения вопроса с одним правильным ответом представлен на рисунке 4.12.

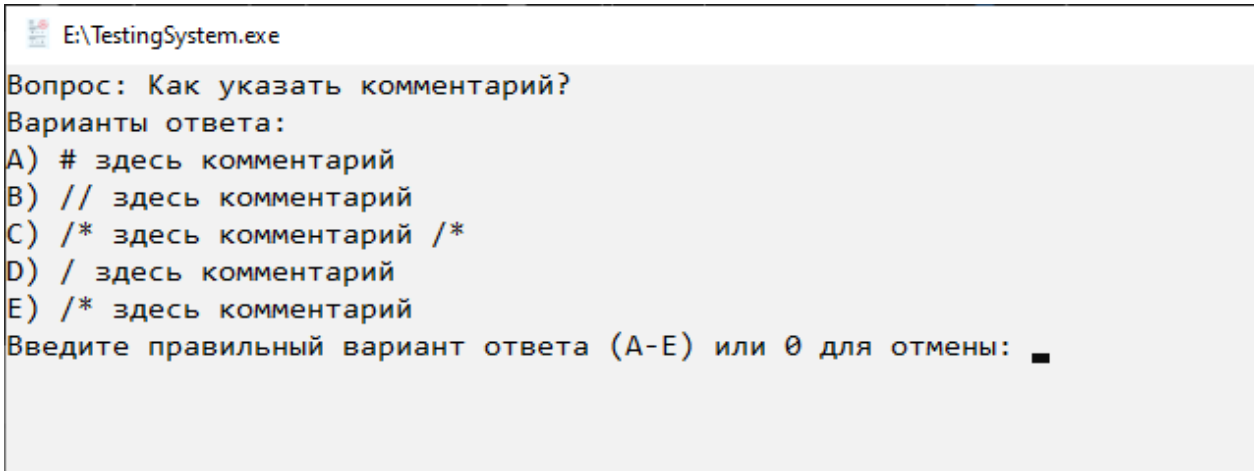

Рисунок 4.12 – Добавление вопроса с одним правильным ответом

Для добавления вопроса с несколькими правильными ответами администратор вводит текст задания, формулирует варианты ответов и указывает правильные. Пользователю предлагается выбрать комбинацию ответов, вводя соответствующие буквенные обозначения, либо отказаться от ответа, введя значение 0. Пример интерфейса для создания и отображения такого вопроса приведен на рисунке 4.13.

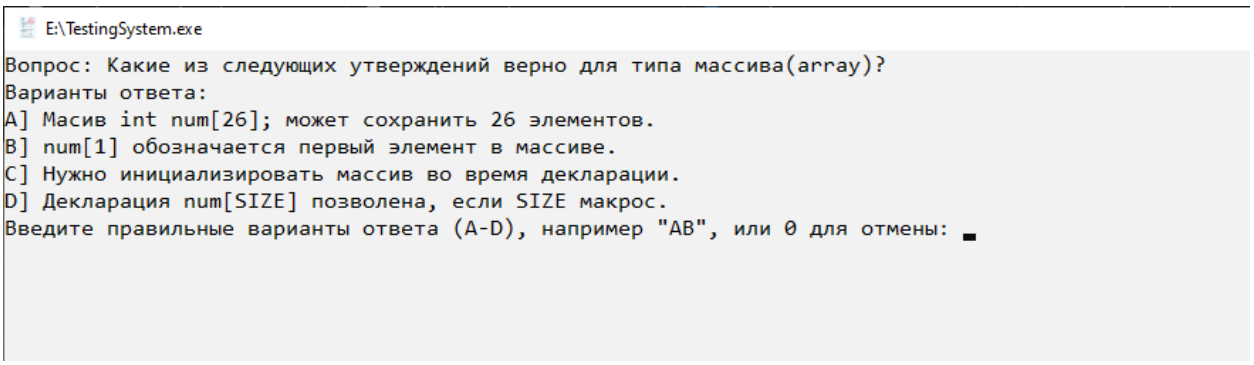

Рисунок 4.13 – Добавление вопроса с несколькими правильными ответами

Для добавления вопроса с открытым ответом администратор вводит текст вопроса и указывает правильный ответ, который будет использоваться для проверки. Пример интерфейса для ввода такого вопроса представлен на рисунке 4.14.

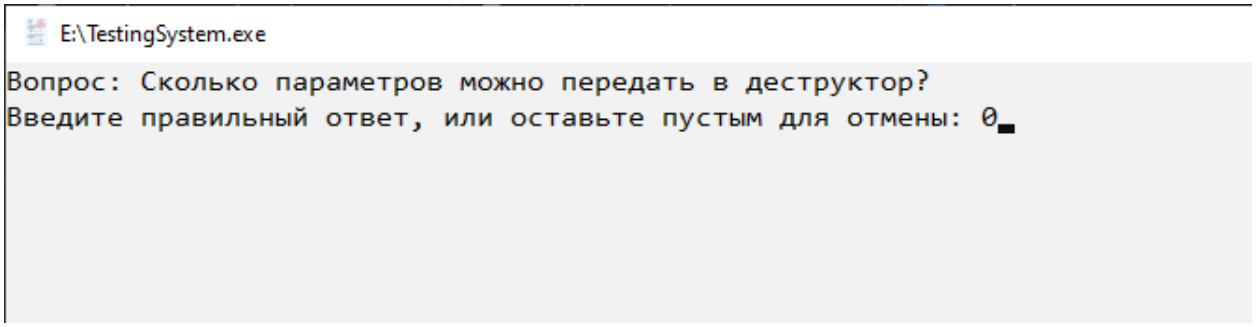

Рисунок 4.14 – Добавление вопроса с открытым ответом

Возвратившись в меню администратора, можно выбрать пункт "Удалить тест". После выбора отображается список доступных тестов, из которого администратор может выбрать тест для удаления. Пример интерфейса для удаления теста показан на рисунке 4.15.

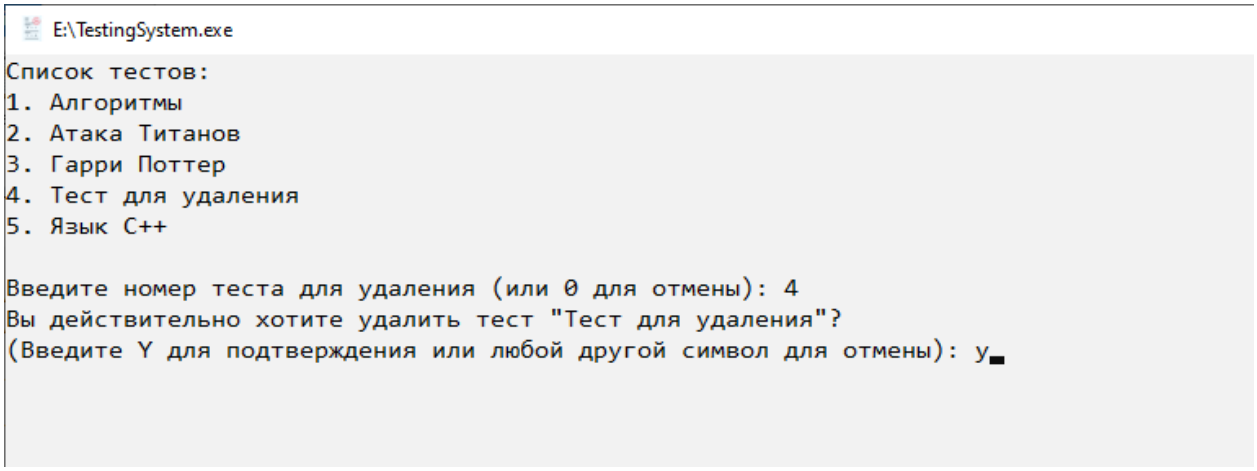

Рисунок 4.15 – Интерфейс удаления теста

В меню администратора доступен пункт "Редактирование теста". После выбора отображается список тестов, из которого администратор выбирает

тест для редактирования. Возможности редактирования включают изменение названия теста, добавление новых вопросов, удаление существующих и изменение параметров вопросов. Пример интерфейса для редактирования теста представлен на рисунке 4.16.

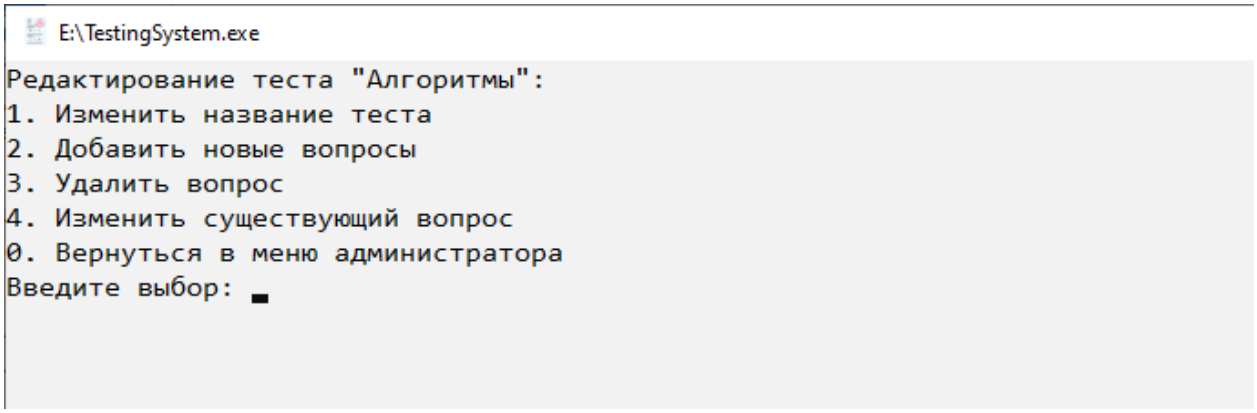

Рисунок 4.16 – Интерфейс удаления теста

В меню редактирования теста доступна функция "Редактирование вопросов". После выбора отображается список вопросов теста, где администратор может выбрать нужный вопрос для изменения. Возможности включают редактирование текста вопроса, вариантов ответов, а также корректировку правильных ответов. Пример интерфейса для редактирования вопросов представлен на рисунке 4.17.

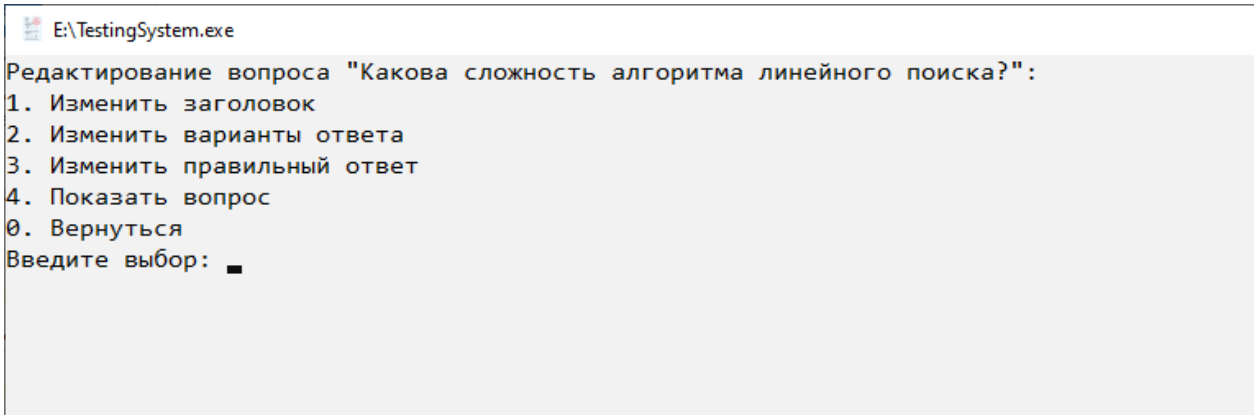

Рисунок 4.17 – Интерфейс удаления теста

В меню администратора доступен пункт "Просмотр статистики", который позволяет получить информацию о результатах тестирования пользователей. При выборе этого пункта открывается меню статистики, где администратор может просмотреть текущие результаты, удалить статистику или вернуться в предыдущее меню для дальнейшего управления тестами. Такой интерфейс обеспечивает эффективный контроль и анализ данных о прохождении тестов. Пример интерфейса меню статистики представлен на рисунке 4.18.

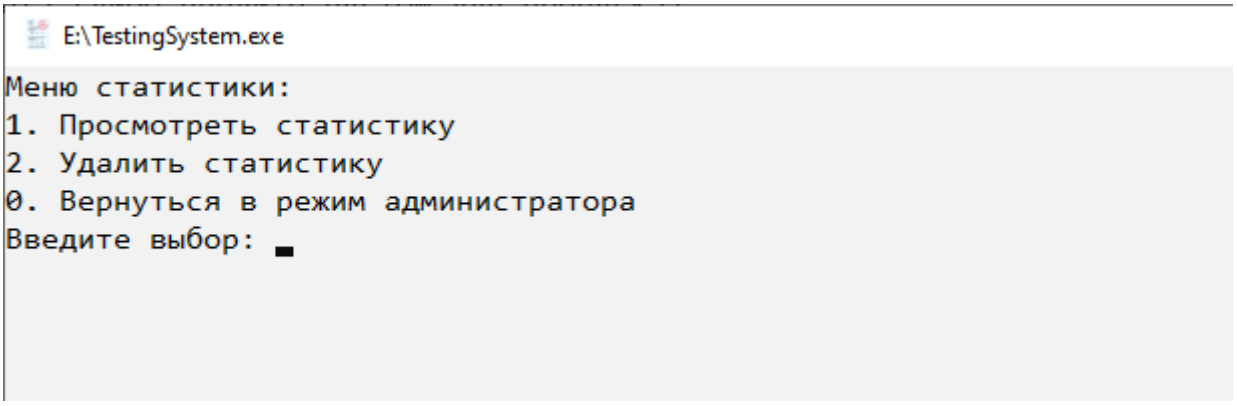

Рисунок 4.18 – Интерфейс меню статистики

При выборе пункта "Просмотр статистики" отображается таблица с данными о статистике. Таблица содержит три колонки: название теста, имя студента и время прохождения. При выборе строки из таблицы открывается полная статистика, включающая детальную информацию о тесте, ответы студента и правильные ответы. Пример интерфейса таблицы представлен на рисунке 4.19, а интерфейс с полной статистикой — на рисунке 4.20.

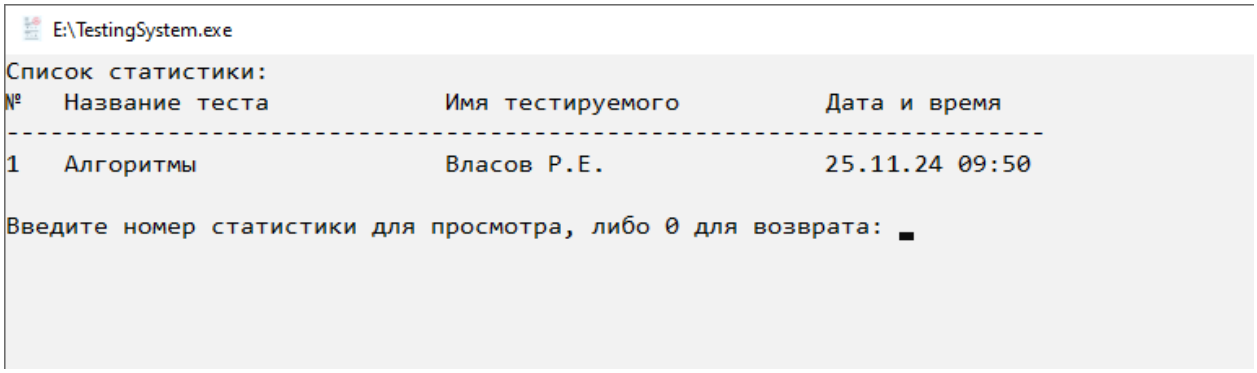

Рисунок 4.19 – Интерфейс выбора статистики

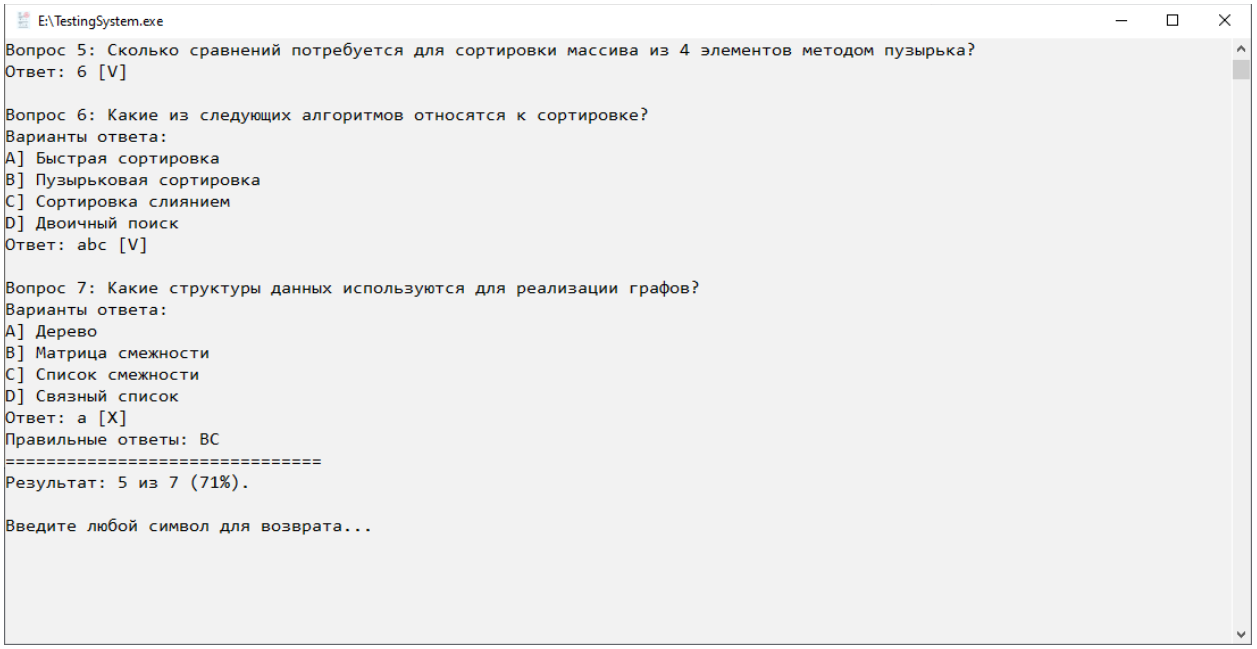

Рисунок 4.20 – Интерфейс статистики

ЗАКЛЮЧЕНИЕ

В результате выполнения курсового проекта была создана работающая программа «Система тестирования знаний» с полным набором необходимых функций. Все задачи, поставленные перед проектом, были успешно решены, а функционал реализован в полном объеме.

В процессе разработки были изучены особенности языка программирования C++, включая объектно-ориентированный подход. Работа над проектом включала несколько ключевых этапов: анализ существующих аналогов и литературы, формулировка требований к программному обеспечению, проектирование функционала, конструирование приложения, разработка отдельных модулей и тестирование готового решения.

Результатом последовательного выполнения этих этапов стало создание исправно работающего приложения. В перспективе планируется модернизация программы, включая добавление графического интерфейса и расширение функциональных возможностей.

ЛИТЕРАТУРА

[1] Google Forms [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.google.ru/forms/about/ – Дата доступа: 10.10.2024

[2] Moodle [Электронный ресурс]. – Электронные данные. – Режим доступа: https://moodle.org/ – Дата доступа: 10.10.2024

[3] Типы вопросов и их применение в тестировании [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.edutopia.org/assessment-guide – Дата доступа: 10.10.2024

[4] Форматы JSON и XML [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.w3schools.com/xml/ – Дата доступа: 10.10.2024

[5] Страуструп, Б. Программирование. Принципы и практика использования C++/ Б. Страуструп. – Вильямс, 2018 г. – 991 с.

[6] Скляров, В. А. Язык C++ и объектно-ориентированное программирование: справ. пособие / В. А. Скляров. – Минск : Выш. шк., 1997.– 540 с.

[7] Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. – Санкт-Петербург, 2014. – 366 с.

[8] Саттер Г., Александреску А. Стандарты программирования на C++: 101 правило, рекомендация и лучшие практики / Саттер Г., Александреску А. – Бостон, 2004. – 240 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Диаграмма классов

ПРИЛОЖЕНИЕ Б

(обязательное)

Схема алгоритма вычисления расстояния Левенштейна

ПРИЛОЖЕНИЕ В

(обязательное)

Схема алгоритма проверки ключа администратора