Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра инженерной психологии и эргономики

Дисциплина: Основы алгоритмизации и программирования

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

**КОНСТРУКТОР ЭКЗАМЕНАЦИОННЫХ БИЛЕТОВ**

БГУИР КР 6 - 05 - 06 12 01 029 ПЗ

Выполнил: студент группы 310901 Усов А. М.

Проверил: Кабариха В. А.

Минск 2024

Содержание

[Введение 3](#_Toc165373744)

[1 Требования к программе 4](#_Toc165373745)

[2 Конструирование программы 7](#_Toc165373746)

[2.1 Описание модулей программы 7](#_Toc165373747)

[2.2 Выбор способа организации данных 8](#_Toc165373748)

[2.3 Разработка перечня пользовательских функций программы 10](#_Toc165373749)

[3 Разработка алгоритмов работы программы 12](#_Toc165373750)

[3.1 Алгоритм функции *main* 12](#_Toc165373751)

[3.2 Алгоритм функции поиска 14](#_Toc165373752)

[3.3 Алгоритм функции сортировки 16](#_Toc165373753)

[Раздел 4 19](#_Toc165373754)

[4.1 Авторизация 19](#_Toc165373755)

[4.2 Модуль администрирования 20](#_Toc165373756)

[4.3 Модуль пользователя 21](#_Toc165373757)

[4.4 Исключительные ситуации 25](#_Toc165373758)

[Заключение 28](#_Toc165373759)

[Приложение А 29](#_Toc165373760)

[Листинг кода 29](#_Toc165373761)

Введение

Традиционно подготовка экзаменационных билетов была крайне трудоемким процессом для преподавателей. Им приходилось вручную составлять сотни вопросов и компоновать в билеты. Этот процесс занимал колоссальное количество времени и требовал серьезных усилий.

Современные тенденции заключаются в разработке приложений-конструкторов для автоматизации процесса на всех этапах – от создания шаблонов билетов и банка вопросов до формирования билетов и их печати. Такие решения способны существенно экономить время преподавателей, снижать вероятность ошибок, обеспечивать актуальность материалов.

Однако существующие программные продукты часто обладают ограниченным функционалом, плохо интегрируются с другими системами учебного заведения, имеют неудобный и запутанный интерфейс. Зачастую они решают лишь локальные задачи автоматизации, в то время как комплексный подход отсутствует.

Цель данной курсовой работы – создание программного обеспечения «Конструктор экзаменационных билетов» для автоматизации процесса подготовки экзаменационных материалов в учебных заведениях.

Для достижения поставленной цели были сформулированы следующие задачи:

* провести анализ требований к экзаменационным билетам в различных учебных заведениях;
* изучить существующие программные решения, выявить их достоинства и недостатки;
* спроектировать архитектуру приложения с учетом требований масштабируемости и расширяемости;
* реализовать функционал по созданию билетов и формированию банка вопросов;
* обеспечить возможность генерации комплектов билетов по заданным правилам;
* предусмотреть различные варианты вывода подготовленных билетов (печать, электронный формат).

1 Требования к программе

Учебные компьютерные программы, такие как «Конструктор экзаменационных билетов», позволяют моделировать процесс создания, проверки и экспорта экзаменационных билетов.

В разрабатываемой системе главными объектами являются экзаменационные билеты и тесты, рейтинг и пользователи.

Тест – это объект, включающий в себя следующую информацию:

* название теста;
* имя автора теста;
* сложность теста;
* количество стандартных вопросов;
* набор стандартных вопросов;
* количество вопросов с несколькими вариантами ответа;
* набор вопросов с несколькими вариантами ответа.

Пользователи – это объекты, включающие в себе следующую информацию:

* имя пользователя;
* логин пользователя;
* пароль пользователя;
* список его созданных тестов;
* уровень доступа;
* ID пользователя.

Рейтинг – это объект, включающий следующую информацию:

* имя соответствующего теста;
* количество прохождений;
* список пользователей и их результатов;

При проектировании программного средства для автоматизации создания билетов к экзамену определяются функциональные требования, которые необходимо внедрить в систему:

* возможность регистрации новых пользователей;
* авторизация уже зарегистрированных пользователей;
* авторизация в качестве гостя;
* прохождение выбранного теста;
* возможность просмотра рейтинга пользователей, прошедших определенный тест;
* возможность создания и редактирования теста;
* процесс экспорта созданного теста на печать;
* генерация билетов;
* создание и редактирование вопросов и ответов;
* процесс автоматической оценки после прохождения теста.
* возможность сортировать тесты по сложности;
* гость должен иметь возможность пройти любой тест из списка публичных тестов;
* администратор должен иметь возможность редактировать публичные тесты.

Пользователь должен иметь возможность проходить публичные тесты, а также создавать свои собственные тесты с возможностью редактирования.

Администратор должен иметь возможность делать все вышеперечисленное, а также редактирование любых тестов.

Взаимодействие всех ролей и функций показаны на *Use-case* диаграмме (Рисунок 1.1).

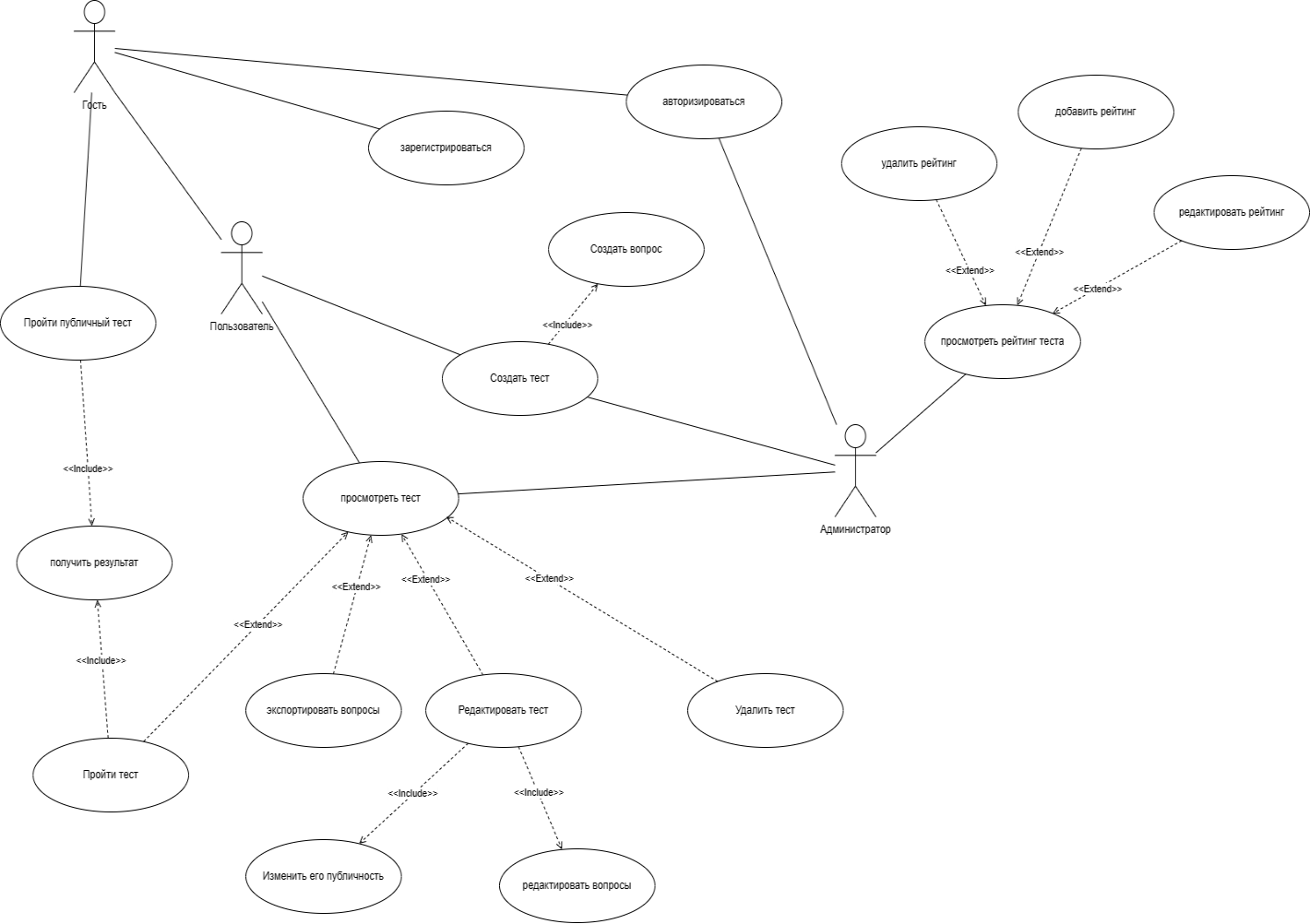


Рисунок 1.1 – *Use-case* диаграмма

При выполнении курсовой работы «Конструктор экзаменационных билетов» для создания безопасного и надежного программного обеспечения были поставлены следующие нефункциональные требования:

* все хранимые данные пользователей должны быть представлены в виде шифра по определенному алгоритму;
* должна существовать документация для пользователя, которая научит его пользоваться программным обеспечением;
* должна существовать возможность экспорта готовых билетов в формате текстового файла;
* должен быть реализован интерфейс в консоли, основанный на вводе данных пользователем и выводе текстовой информации;
* сохранение всех данных пользователей, тестов и рейтинга в файлы на жёстком диске.

Реализация данных нефункциональных требований позволит достичь при создании программного обеспечения «Конструктор экзаменационных билетов» удобного взаимодействия пользователя с программной, а также простое обучение пользователя, будет обеспечена удобная реализация хранения, а также безопасность всех важных файлов посредством шифрования.

2 Конструирование программы

2.1 Описание модулей программы

При проектировании программного обеспечения «Конструктор экзаменационных билетов» была выбрана модульная архитектура проекта, которая позволит удобно организовать все необходимые функции и классы, а также позволит в перспективе реализовать поддержку, что позволит разрабатывать приложение в команде.

Все основные модули программного обеспечения «Конструктор экзаменационных билетов» представлены на таблице 2.1.

Связь между модулями наглядно показана на диаграмме модулей (Рисунок 2.1).



Рисунок 2.1 – модульная диаграмма

Таблица 2.1 – Описание основных модулей программного средства конструктор экзаменационных билетов

|  |  |  |
| --- | --- | --- |
| Название модуля | Описание | Функционал |
| Модуль регистрации | Обеспечивает процесс регистрации и авторизации пользователей. | * регистрация новых читателей; * авторизация уже зарегистрированных пользователей; * авторизация как гость. |
| Модуль управления тестами | Отвечает за управления тестами. | * создание тестов; * редактирование тестов; * поиск тестов. |
| Модуль работы с билетами | Отвечает за создание, редактирование и удаление билетов. | * создать билеты; * редактировать билеты; * удалить билеты. |
| Модуль прохождения тестов | Отвечает за возможность прохождения теста | * прохождение теста; * просмотр теста; * получение оценки. |
| Модуль администрирования | Обеспечивает процесс управления пользователями. | * управление пользователями. |
| Модуль для работы с файлами | Обеспечивает взаимодействие программы с файлами. | * запись в файлы; * чтение файлов. |

2.2 Выбор способа организации данных

При разработке программного обеспечения «Конструктов экзаменационных билетов» для удобного хранения и быстрого доступа была выбрана файловая структура хранения данных, где некоторые важные поля (например пароль пользователя) зашифрованы для обеспечения безопасности пользователей.

Для реализации такого способа хранения данных была разработана следующая структура файлов:

* файл всех пользователей, хранящий ID, имя, логин, пароль, уровень доступа и количество созданных тестов;
* файл со всеми тестами с пометкой «публичный», такие тесты могут проходить незарегистрированные пользователи;
* набор файлов, где каждый файл имеет в названии ID пользователя (для удобного поиска). Каждый такой файл хранит данные всех созданных этим пользователем тестов;
* файл пользовательской документации, для удобного представления документации на выводе и упрощенного редактирования;
* файл рейтинга, хранящий в себе данные о каждом пройденном теста, количество прохождений, а также имя пользователя, прошедшего тест и результат прохождения.

Для более удобной для разработчика структуры, файлы, хранящие данные тестов, находятся в отдельно созданной папке (кроме файла рейтингов), файлы, которые хранят информацию про пользователей также находятся в отдельной папке. Для файла документации была также выделена отдельная папка.

Все сохраняемые программной файлы имеют схожую структуру, которая представляет собой набор переменных, которые записываются с новой строки. Пример представлен на рисунке 1.2.



Рисунок 1.2 – представление файла тестов

Остальные файлы имеют такую же структуру. Она позволяет иметь удобный доступ к информации из программы, но усложняет восприятие при чтении файла при несанкционированном доступе.

2.3 Разработка перечня пользовательских функций программы

Необходимый перечень функций, которые будут задействованы в реализации консольной программы на *C++* «Конструктор экзаменационных билетов» представлен в таблице 2.2.

Таблица 2.2 – Перечень основных пользовательских функций

|  |  |  |  |
| --- | --- | --- | --- |
| № | Функция | Аргументы | Описание |
| 1 | void Menu(User& user); | – User &user (принимает ссылку на пользователя который вызвал меня) | Запускает стартовое меню. |
| 2 | void RegisterModule(); | Ничего не принимает | Вызывает окно регистрации. |
| 3 | string customHash(const string& input); | * String input(входная строка) | Принимает строку и возвращает ее хэш. |
| 4 | void PersonUI(User user); | * User user (пользователь который вошел в систему) | Вызывает меню для пользователя. |
| 5 | void GuestUI(User user); | * User user (пользователь который вошел в систему) | Вызывает меню для гостя. |
| 6 | void AdminUI(User user); | * User user (пользователь который вошел в систему) | Вызывает меню для администратора. |
| 7 | int getLastId(string path); | * path (путь) | Возвращает количество пользователей в системе |

Продолжение таблицы 2.2

|  |  |  |  |
| --- | --- | --- | --- |
| 8 | Test EditTest(Test myTest); | * Test myTest (ранее созданный тест). | Редактирует выбранный тест и возвращает его. |
| 9 | User Login(); | Ничего не принимает | Запускает взаимодействие через консоль и проверяет введенные данные пользователя и разрешает доступ к определенным функциям программы. |
| 10 | Test CreateTest(); | Ничего не принимает. | Отвечает за создание теста. |
| 11 | int startTest(Test test); | Test test (ранее созданный тест). | Реализует прохождение теста и возвращает оценку за него. |
| 12 | void ShowTest(Test test,bool isPublic); | * Test test (ранее созданный тест);   Bool isPublic ( публичный ли тест) | Показывает тест с ответами. |
| 13 | Test Sort(Test test, string rule); | * Test \* tests (массив тестов); * string rule (правило сортировки). | Сортирует исходный массив тестов по определенному переданному правилу (переменной). |
| 14 | void ShowTests(TestsContainer\* tests) | * TestsContainer test ( список тестов) | Вызывает меню для показа списка тестов. |

Продолжение таблицы 2.2

|  |  |  |  |
| --- | --- | --- | --- |
| 15 | void ShowUsers(); | Ничего не принимает | Показывает всех зарегистрированных пользователей. |
| 16 | UsersContainer\* ReadUsers(); | Ничего не принимает | Возвращает список зарегистрированных пользователей. |
| 17 | int WriteToFile(string file\_path, User user); | * String file\_path (путь к файлу); * User user (пользователь который вызвал функцию). | Записывает в файл пользователя который зарегистрировался и возвращает статус записи. |
| 18 | int WriteToFile(string file\_path, Test test); | * String file\_path (путь к файлу); * Test test (созданный тест). | Записывает созданный тест в файл тестов. |
| 19 | int WriteToFile(string file\_pass, UserData user); | * String file\_path (путь к файлу); * UserData user (данные зарегистрированного пользователя). | Записывает в файл данные зарегистрированного пользователя |
| 20 | int WriteToFile(string file\_pass, string test\_name, string user\_name, int result); | * String file\_path (путь к файлу); * String test\_name (название теста); * String user\_name (имя пользователя); * int result (результат прохождения). | Записывает в файл результат прохождения теста пользователем. |

3 Разработка алгоритмов работы программы

3.1 Алгоритм функции *main*

Функция *main* играет ключевую роль в консольном приложении «Конструктор экзаменационных билетов». Это основная точка входа, где начинается выполнение программы. Её задача - инициализировать консоль, предложить пользователю выбор действий и перенаправить его в соответствующий интерфейс в зависимости от его уровня доступа.

Приложение начинает с очистки консоли и вывода приветственного сообщения на русском языке. Далее, через вызов функции меню (*Menu*()), предоставляется выбор действий пользователю. Это позволяет разделить логику работы с приложением на более мелкие и управляемые куски, обеспечивая понятность и легкость сопровождения кода. После выбора действия пользователем, программа направляет его на соответствующий интерфейс: для гостя, пользователя или администратора.

Пошаговый словесный алгоритм:

1. Устанавливаются кодировки консоли для корректного отображения символов на кириллице;
2. Очищается экран консоли;
3. Выводится приветственное сообщение;
4. Создается объект типа *User*;
5. Устанавливается начальное значение поля *ruleLevel* объекта *User*;
6. Запускается функция Menu, которая отвечает за меню выбора действий. После завершения этой функции значение поля *ruleLevel* объекта *User* изменяется в зависимости от выбранного действия пользователя;
7. Выполняется оператор *switch* по значению поля *ruleLevel* объекта *User*:

* Если *ruleLevel* равен 0, к шагу 8;
* Если *ruleLevel* равен 1, к шагу 9;
* Если *ruleLevel* равен 2, к шагу 10;

1. Вызов функции *GuestUI*, которая отвечает за интерфейс гостя, далее к шагу 11;
2. Вызов функции *AdminUI*, которая отвечает за интерфейс администратора. После чего выводится значение поля *ruleLevel* объекта *User*, далее к шагу 11;
3. Вызов функции *PersonUI*, которая отвечает за интерфейс пользователя, далее к шагу 11;
4. значение 0, обозначающее успешное завершение программы

Более наглядно алгоритм функции *main* представлен на блок-схеме (Рисунок 3.1).

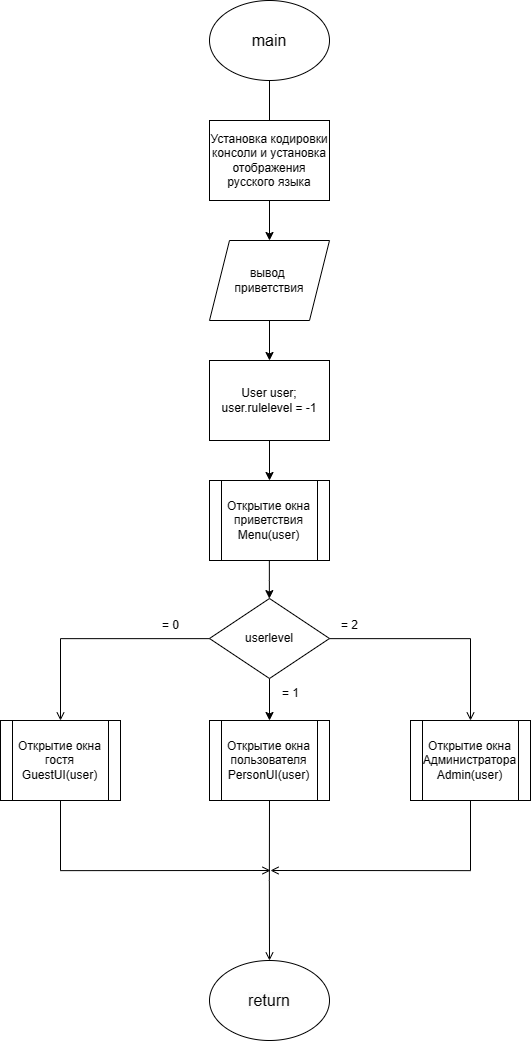


Рисунок 3.1 – Блок-схема алгоритма функции *main*

Такая реализация функции *main* позволит легко разбивать программу на модули, а также уменьшит объем функции *main* и улучшит ее читаемость

3.2 Алгоритм функции поиска

Функция *LinearFind* является важным компонентом в обработке данных оценок в консольном приложении «Конструктор экзаменационных билетов». Её задача - провести линейный поиск максимальной оценки, полученной определенным пользователем.

Этот алгоритм выбран из-за его простоты и понятности реализации. Он эффективен для небольших объемов данных, что позволяет быстро найти максимальную оценку в списке результатов экзаменов. После завершения работы функции, программа может использовать найденное значение для дальнейшего анализа и предоставления соответствующей информации пользователю.

Пошаговый словесный алгоритм:

1. Принимаются указатели на объекты *raiting* (оценки) и *user* (пользователь), для которого производится поиск.
2. Инициализируется переменная *max*, которая будет хранить максимальное значение результата.
3. Создается временный указатель *temp*, указывающий на начало связного списка оценок.
4. Начинается цикл, который проходит по всем элементам связного списка, пока указатель temp не станет равным *nullptr*.
5. Внутри цикла проверяется условие: если идентификатор пользователя элемента списка совпадает с идентификатором пользователя, для которого выполняется поиск, и результат текущего элемента больше, чем текущее максимальное значение *max*, то *max* обновляется значением результата текущего элемента.
6. Указатель *temp* перемещается к следующему элементу списка.
7. По окончании цикла возвращается найденное максимальное значение *max*.

Более наглядно алгоритм функции *LinearFind* представлен на блок-схеме (Рисунок 3.2).

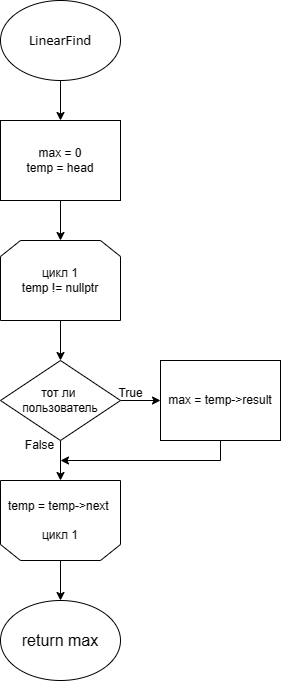


Рисунок 3.2 – Блок-схема алгоритма *LinearFind*

Алгоритм линейного поиска позволяет в очень простой и понятной форме найти нужный элемент списка или массива, но при этом при данной ситуации его использование является оптимальным, так как массив на входе не гарантированно упорядочен.

3.3 Алгоритм функции сортировки

При разработке программного обеспечения были выбраны проверенные сортировки, такие как: слиянием, вставками, шейкерная. Сортировка слиянием была выбрана из-за своей понятности при реализации и стабильной быстрой скорости работы. Вставками была выбрана в качестве одной из базовых для упрощения и не загромождения кода. шейкерная сортировка была выбрана как самая простая сортировка для небольших объемов данных, но более сложнее чем сортировка пузырьком.

Пошаговый алгоритм функции сортировки слиянием:

1. Разбиваем массив пополам, пока не останется по одному элементу в каждой части;
2. Сортируем каждую половинку рекурсивно, вызывая сортировку слиянием для каждой из них;
3. Сливаем две отсортированные половинки в один отсортированный массив, сравнивая элементы поочередно;
4. Возвращаем отсортированный массив.

Более наглядно алгоритм функции сортировки слиянием представлен на блок-схеме (Рисунок 3.3).



Рисунок 3.3 – Блок-схема алгоритма функции сортировки слиянием

Пошаговый алгоритм функции сортировки вставками:

1. Начинаем с элемента под индексом 1, считая его уже отсортированным;
2. Для каждого элемента находим правильную позицию в отсортированной части массива, сравнивая его с предыдущими элементами;
3. Пока текущий элемент меньше предыдущего и пока не достигнут начало массива, сдвигаем предыдущий элемент вправо;
4. Вставляем текущий элемент на свою правильную позицию;
5. Повторяем шаги 2-4 для всех оставшихся элементов массива.

Более наглядно алгоритм функции сортировки слиянием представлен на блок-схеме (Рисунок 3.4).

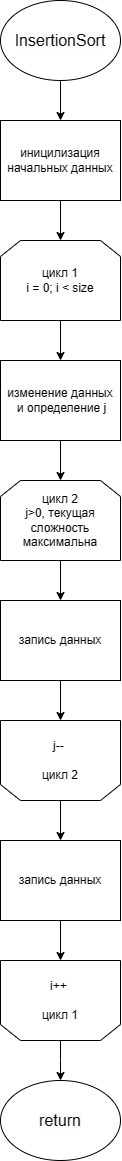


Рисунок 3.5 – Блок-схема алгоритма функции сортировки вставками

Пошаговый алгоритм функции шейкерной сортировки:

1. Устанавливаем начальные значения указателей *prev* и *curr* на начало списка.
2. Проводим проход по списку слева направо, сравнивая соседние элементы и обменивая их местами, если текущий больше следующего.
3. После завершения прохода, уменьшаем значение указателя *curr* на 1, а *prev* устанавливаем в *nullptr*.
4. Проводим проход по списку справа налево, обменивая элементы, если текущий меньше предыдущего.
5. Повторяем процесс до тех пор, пока массив не будет полностью отсортирован.

Более наглядно алгоритм функции шейкерной сортировки представлен на блок-схеме (Рисунок 3.5).

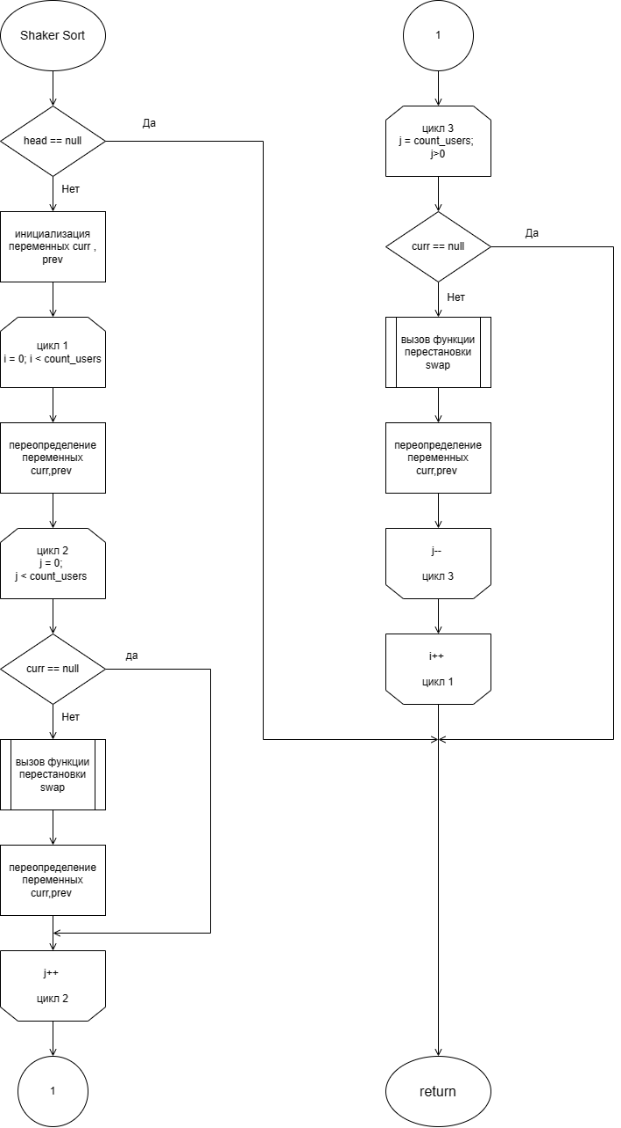


Рисунок 3.5 – Блок-схема алгоритма функции шейкерной сортировки

Таким образом реализация данных сортировок позволит решить задачи с сортировкой данных для пользователя наиболее лучшим способом. Выбор как способом будут сортироваться данные зависит от контекста использования и при оптимальном распределении будут выбраны наиболее подходящие сортировки.

Раздел 4

4.1 Авторизация

После старта программы пользователю выводится окно приветствия, которое представлено на рисунке 4.1

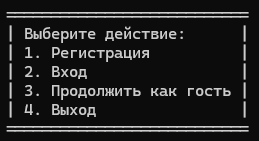


Рисунок 4.1 – Стартовое окно

Далее в зависимости от выбора пользователя, управление переходит в блок регистрации в качестве регистрации, логина. (продолжить как гость тоже является логином).

При выборе пункта «Вход» запускается соответствующая функция *Login*, которая выводит окно как на рисунке 4.2.

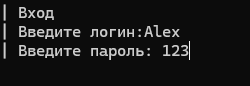


Рисунок 4.2 – Окно функции *Login*

Далее после ввода соответствующих данных происходит хеширование введенного пароля и проверка на существование пользователя, и в результате выводит либо Успешный вход, либо неверный логин или пароль. При успешном входе вызывается *UI* который зависит от уровня доступа пользователя.

В случае выбора регистрации вызывается соответствующая функция, которая выводит окно, представленное на рисунке 4.3.

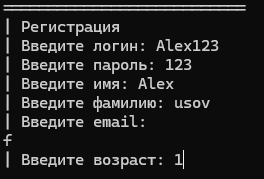


Рисунок 4.3 – Окно регистрации

В случае если все данные были введены корректно, происходит хеширование пароля вызывается функция, которая записывает пользователя в файл если это возможно (такого пользователя не существует). После регистрации вызывается стартовое окно.

При выпоре пункта «Продолжить как гость» сразу вызывается соответствующая *UI* функция.

4.2 Модуль администрирования

Для того, чтобы зайти в панель администрирования нужно в окне авторизации ввести заранее определенные в программе логин и пароль, тогда будет вызвана функция, которая выведет окно администрирования (представлено на рисунке 4.4), в котором можно просмотреть всех зарегистрированных пользователей, а также редактировать любой публичный тест.

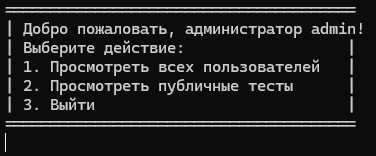


Рисунок 4.4 – окно администрирования

При выборе пункта «Просмотреть всех пользователей» будет открыто окно, в котором будет выведена вся информация о пользователях, такие как логин, имя, фамилия, почта, возраст, *ID*. Администратор не должен иметь возможность посмотреть на пароль пользователя

При выборе пункта «Просмотреть публичные тесты» будет открыто такое же окно, как и для всех пользователей, за исключением возможности редактировать любой тест.

Таким образом администратору дается право на просмотр и изучение информации о пользователях, за исключением пароля для обеспечения безопасности системы, также есть возможность для редактирования публично открытых тестов, что позволит администратору удалять тесты, содержащие не валидную, не корректную или оскорбительную информацию, что позволить программному обеспечения конструктор экзаменационных билетов быть доступной и приятной для широкого круга людей.

4.3 Модуль пользователя

При старте *UI* пользователя выводится окно, представленное на рисунке 4.5, в котором выводится приветствие с указанием логина пользователя и список возможных действий.

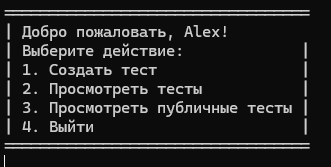


Рисунок 4.5 – Окно вывода *UI* пользователя

При выборе каждого пункта вызывается соответствующая функция, которая представляет свое окно для управления.

При выборе пункта «Выйти» программа завершается.

При выборе пункта «Создать тест» вызывается окно, представленное на рисунке 4.6, в котором пользователю дается возможность создать свой тест и далее при корректном вводе данных и успешном создании выводится соответствующее окно и вызывается функция, которая запишет созданный тест в файл, а также создаст файл для хранения рейтинга.

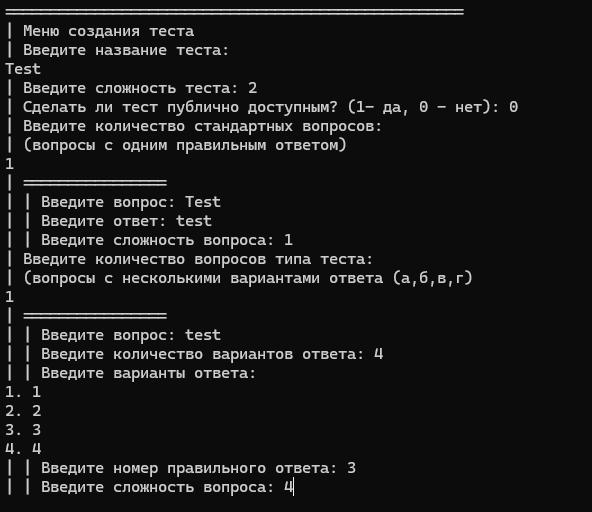


Рисунок 4.6 – Окно создания теста

При выборе пункта «Просмотреть тесты» вызывается функция, которая отобразит окно, представленное на рисунке 4.7, в котором отобразятся все тесты пользователя с выводом имени теста и сложности, и возможностью сортировать их по разным параметрам. При выборе соответствующего параметра сортировки происходить упорядочивание массива по переданному параметру. Также программа на входе ждет номер теста и списка, при выборе которого вызывается меню отображения теста, в которую параметром передается сам тест и пользователь, который вызывает функцию.

При выборе пункта «Посмотреть публичные тесты» вызывается такая же функция, но только для отображения публичных тестов.

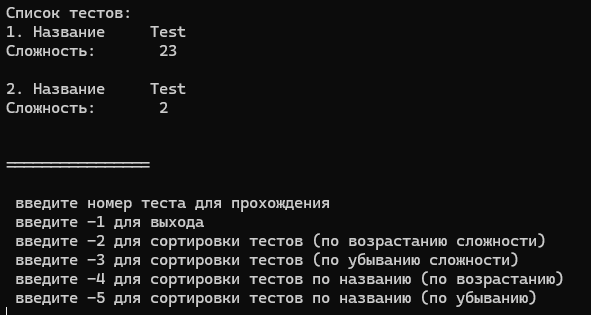


Рисунок 4.7 – Окно вывода тестов

Далее при выборе теста отображается его окно, представленное на рисунке 4.8, которое отображает название теста и список возможных действий с ним, при выборе соответствующего пункта вызывается выбранная функция.

Если логин пользователя совпадает с логином пользователя, который вызвал функцию то выводится дополнительный пункт «Редактировать тест»

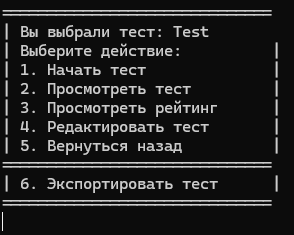


Рисунок 4.8 – Окно теста

При выборе пункта «Начать тест» вызывается функция *StartTest,* которая возвращает оценку и записывает в файл нужного рейтинга результат.

При выборе пункта «Просмотреть тест» вызывается функция *ShowTest* которая выводит на экран тест без ответов.

При выборе пункта «Просмотреть рейтинг» открывается окно рейтинга, представленное на рисунке 4.9, в котором отображается сам рейтинг (отсортированный по убыванию), наивысший результат текущего пользователя и средний результат всех пользователей. При нажатии любой кнопки кроме *Space* открывается предыдущее окно (меню выбранного теста).

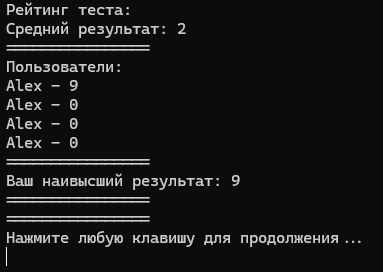


Рисунок 4.9 – Окно просмотра рейтинга

При выборе пункта «Редактировать тест» в окне просмотра выбранного теста вызывается функция *EditTest* которое отображает окно, представленное на рисунке 4.10, в котором выводится название текущего теста, и список возможных действий. Также при открытии этого окта создается локальная копия теста.

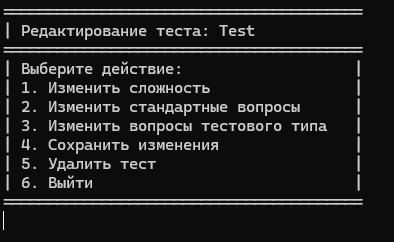


Рисунок 4.10 – Окно редактирования теста

При выборе пунктов изменения, программа предложит пользователю ввести новые данные, и, если они корректна, обновит локальный тест.

При выборе пункта «Сохранить изменения» нужный тест из файла будет удален, и запишется его локальная измененная копия.

При выборе пункта «Выйти» откроется предыдущее окно и, если результаты не были сохранены, они будут удалены.

При выборе пункта «Экспортировать тест» в окне теста будет вызвана соответствующая функции для открытия окна экспорта, представленного на рисунке 4.11, в котором будет отображаться название теста и варианты экспорта.

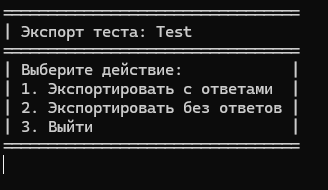


Рисунок 4.11 – Окно экспорта теста

Пример экспорта текущего теста с ответами представлен на рисунке 4.12.

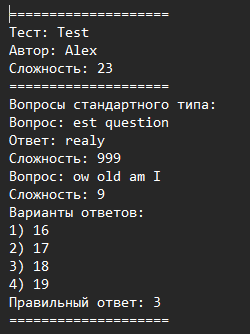


Рисунок 4.12 – Запись экспортированного теста в файл

При выборе пункта «Выйти» отобразится предыдущее окно.

Для публичных тестов все окна сохраняют свою структуру и поля, за исключением возможности редактирования для иных пользователей.

Реализация такого интерфейса должна упростить понимание для пользователя самого программного обеспечения конструктор экзаменационных билетов и улучшить возможность быстрого освоения интерфейса и функционала программы, что хорошо скажется на удобстве.

4.4 Исключительные ситуации

Исключительные ситуации – это непредвиденные события или условия, которые могут возникнуть в процессе работы программного обеспечения. Они могут включать в себя ошибки ввода данных пользователем или неправильное функционирование программы. Решение таких ситуаций обычно требует специальных механизмов обработки ошибок, чтобы обеспечить корректное выполнение программы.

Первый тип таких ситуаций – неверный ввод пользователя, обычно возникает, когда программа ждет от пользователя целочисленное значение, а пользователь по какой-либо причине ввел строку. Такие ситуации решаются встроенными функциями и конструкциями. На рисунке 4.13 представлена такая ситуация, когда пользователю нужно ввести свой возраст при регистрации, и он по ошибке может ввести строку.

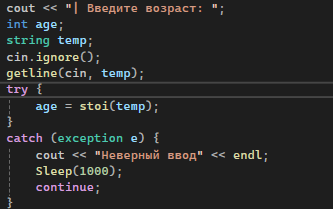


Рисунок 4.13 – Исключительная ситуация первого типа

Чтобы обойти эту проблему используется временная переменная, в которую пользователь вводит нужные данные, далее программа пробует преобразовать полученную строку в число, и, если это не выходит сделать, выводит ошибку ввода и перезапускает цикл.

Второй тип исключительных ситуаций происходит в файловой системе. Если у пользователя нет тестов, а он пытается их просмотреть, необходима проверка (представлена на рисунке 4.14), которая будет следить чтобы массив тестов не был пустым. Если же тестов у пользователя нет, то произойдет ошибка чтение файла тестов пользователя и функция вернет *nullptr*, далее, в ходе проверки если массив равен *nullptr* выведет ошибку, что у пользователя нет тестов и откроет предыдущее окно. Такая же реализация и для массива рейтинга.

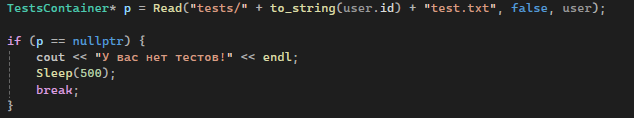


Рисунок 4.14 – Исключительная ситуация второго типа

Исключительная ситуация третьего типа связана она с тем, что при попытке обновить файл рейтинга его может не существовать, а для обновления вначале нужно чтение, что и вызовет ошибку. Чтобы такая ошибка не могла произойти, нужно при создании теста также создавать файл рейтинга. Для этого была введена новая функция *CreateRaiting* (представлена на рисунке 4.15)

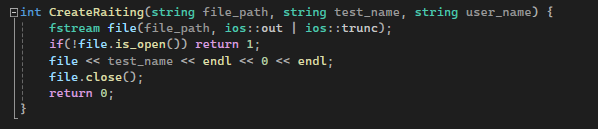


Рисунок 4.15 – код функции *CreateRaiting*

Реализовав решение для всех этих исключительных ситуаций, можно получить программное обеспечение, которое сможет обрабатывать ошибки не завершая работу.

Заключение

В рамках выполнения данной курсовой работы было разработано программное обеспечение конструктор экзаменационных билетов. Целью этого программного продукта является обеспечение удобства и эффективности в процессе создания, экспорта и прохождения тестов. Были разработаны и успешно реализованы все необходимые функции, что позволяет сделать использование программы интуитивно понятным и простым для потенциальных пользователей.

Проведенный сравнительный анализ с аналогичными продуктами на рынке, такими как *Online Test Pad* и *Google Forms*, показал, что программное обеспечение конструктор экзаменационных билетов обладает основным функционалом, который является конкурентоспособным. Однако, чтобы выделиться среди аналогов и привлечь больше пользователей, необходимо уделить внимание реализации графического интерфейса и добавлению дополнительных функций, способных привлечь внимание пользователей.

В перспективе развития программного обеспечения конструктор экзаменационных предлагается несколько направлений. Во-первых, можно расширить функционал программы путем добавления новых типов вопросов, что сделает её более универсальной и привлекательной для широкого круга пользователей. Во-вторых, важным шагом будет улучшение механизмов экспорта результатов тестирования для повышения удобства работы с полученной информацией. И, наконец, создание веб-версии программы станет важным шагом для увеличения её доступности и удобства использования.

Таким образом, данная курсовая работа не только позволила разработать функциональное программное обеспечение для создания экзаменационных билетов, но и определить перспективы его дальнейшего развития, что представляет значимый вклад в область развития образовательных технологий и инструментов.

Приложение А

Листинг кода

Файл ControlTestModule.h:

#pragma once

#include "Structures.h"

#include <iostream>

void PersonUI( User user);

void GuestUI(User user);

void AdminUI(User user);

void CreateTest(UserData user);

int startTest(Test test);

float getMark(int result);

void ShowTest(Test test,bool isPublic);

void ShowTests(TestsContainer\* tests);

void showTestMenu(Test test, User\*user,bool isPublic);

void showRaiting(Raiting\* raiting,User\*user, string user\_name,bool isPublic);

Test\* sortTests(Test\* tests,int size, int choice);

Test\* merge(Test\* left, Test\* right, int l, int r, int choice);

Test\* InsertionSort(Test\* tests, int size);

int LinearFind(Raiting\* raiting, User\* user);

string customHash(const string& input);

void EditTest(Test\*test, User user);

Test edit\_Q\_standart(Test temp);

Test edit\_Q\_guest(Test temp);

void ShowUsers();

Файл ControlTestModule.cpp

// Code for the ControlTestModule

#include <iostream>

#include <string>

#include <windows.h>

#include "Structures.h"

#include "FileModule.h"

#include "ControlTestModule.h"

#include "ExportModule.h"

using namespace std;

// показать тесты (массива)

void ShowTests(TestsContainer\* tests) {

system("cls");

cout << "Список тестов: " << endl;

for (int i = 0; i < tests->count\_tests; i++) {

cout << i + 1 << ". Название\t" << tests->tests[i].name << "\nСложность:\t " << tests->tests[i].difficulty << endl << endl;

}

cout << endl <<"================" << endl << endl;

}

// создание теста

void CreateTest(UserData user) {

//

};

// меню пользователя

void PersonUI(User user) {

//

};

// меню гостя

void GuestUI(User user) {

//

};

// меню администратора

void AdminUI(User user) {

//

}

void ShowUsers() {

UsersContainer\* data = ReadUsers();

if (data == nullptr) {

cout << "Список пользователей пуст" << endl;

return;

}

UserNode\* temp = data->head;

while (temp != nullptr) {

cout << "| ID " << temp->data.id<<"." << endl;

cout << "| Login: " << temp->data.login << endl;

cout << "| Name: " << temp->data.name << endl;

cout << "| Surname: " << temp->data.surname << endl;

cout << "| Email: " << temp->data.email << endl;

cout << "| Age: " << temp->data.age << endl;

cout << "==========================" << endl;

temp = temp->next;

}

}

// показать тест

void ShowTest(Test test,bool isPublic) {

system("cls");

cout << "Название теста: " << test.name << endl;

cout << "Автор: " << test.author << endl;

cout << "Сложность: " << test.difficulty << endl;

cout << "================" << endl;

cout << "Стандартные вопросы: " << endl;

for (int i = 0; i < test.count\_q\_standart; i++) {

cout << i + 1 << ". " << test.q\_standart[i].question << endl;

}

cout << "================" << endl;

cout << "Вопросы тестого типа: " << endl;

for (int i = 0; i < test.count\_q\_guest; i++) {

cout << i + 1 << ". " << test.q\_guest[i].question << endl;

}

cout << "================" << endl;

cout << "Нажмите любую клавишу для продолжения..." << endl;

cin.clear();

cin.get();

}

// показать меню теста

void showTestMenu(Test test, User\*user, bool isPublic) {

//

}

// начать тест

int startTest(Test test) {

cin.clear();

cin.ignore(32767, '\n');

int result = 0;

system("cls");

cout << "================" << endl;

cout << "Тест "<< test.name<< " начался!" << endl;

cout << "================" << endl;

cout << "Стандартные вопросы: " << endl;

for (int i = 0; i < test.count\_q\_standart; i++) {

cout << i + 1 << ". " << test.q\_standart[i].question << endl;

string answer;

cout << "Введите ответ: ";

getline(cin, answer);

if (answer == test.q\_standart[i].answer) {

result += test.q\_standart[i].difficulty;

}

}

cout << "================" << endl;

cout << "Вопросы тестого типа: " << endl;

for (int i = 0; i < test.count\_q\_guest; i++) {

cout << i + 1 << ". " << test.q\_guest[i].question << endl;

cout << "Варианты ответа: " << endl;

for (int j = 0; j < test.q\_guest[i].count\_answers; j++) {

cout << j + 1 << ". " << test.q\_guest[i].answer[j] << endl;

}

int answer;

cout << "Введите номер ответа: ";

string temp\_cin;

getline(cin, temp\_cin);

try {

answer = stoi(temp\_cin);

}

catch (const exception) {

answer = 0;

}

if (answer == test.q\_guest[i].right\_answer) {

result += test.q\_guest[i].difficulty;

}

}

cout << "================" << endl;

return result;

}

// получить оценку

float getMark(int result){

if (result == 0) {

return 0;

}

return result % 10;

}

// merge sort

Test\* sortTests(Test\* tests,int \_size, int choice) {

int size = \_size;

if (size == 1) {

return tests;

}

int mid = size / 2;

Test\* left = new Test[mid];

Test\* right = new Test[size - mid];

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

left[i] = tests[i];

}

for (int i = mid; i < size; i++) {

right[i - mid] = tests[i];

}

left = sortTests(left, mid,choice);

right = sortTests(right,size-mid,choice);

return merge(left, right, mid, size - mid, choice);

}

// merge

Test\* merge(Test\* left, Test\* right, int l, int r, int choice) {

//

}

void showRaiting(Raiting\* raiting, User \* user,string user\_name, bool isPublic) {

system("cls");

cout << "Рейтинг теста: " << raiting->test\_name << endl;

cout << "Средний результат: " << raiting->middle\_result << endl;

cout << "================" << endl;

raiting->BubleSort();

cout << "Пользователи:" << endl;

bool isGoten = false;

Raiting\_node\* temp = raiting->head;

while (temp != nullptr) {

cout << temp->user\_name<< " - " << temp->result << endl;

if (temp->user\_name == user\_name) {

isGoten = true;

}

temp = temp->next;

}

cout << "================" << endl;

if (!isPublic) {

if (isGoten) {

cout << "Ваш наивысший результат: " << LinearFind(raiting, user) << endl;

cout << "================" << endl;

}

else {

cout << "Вы не проходили данный тест" << endl;

}

}

cout << "================" << endl;

cout << "Нажмите любую клавишу для продолжения..." << endl;

if (isPublic) {

cin.ignore();

}

cin.get();

}

int LinearFind(Raiting\* raiting, User\* user) {

int max = 0;

Raiting\_node\* temp = raiting->head;

while (temp != nullptr) {

if (temp->user\_name == getUserData(user->id, "users/usersData.txt").name && temp->result > max) {

max = temp->result;

}

temp = temp->next;

}

return max;

}

void EditTest(Test\* test, User user) {

//

}

Test edit\_Q\_standart(Test temp1){

//

}

Test edit\_Q\_guest(Test temp1) {

//

}

// insertion sort tests by difficulty

Test\* InsertionSort(Test\* tests, int size) {

Test temp;

for (int i = 1; i < size; i++) {

temp = tests[i];

int j = i - 1;

while (j >= 0 && tests[j].difficulty > temp.difficulty) {

tests[j + 1] = tests[j];

j--;

}

tests[j + 1] = temp;

}

return tests;

}

Файл ExportModule.h

#pragma once

#include "Structures.h"

void ExportTestMenu(Test test, User\* user);

Файл ExportModule.cpp

#include "ExportModule.h"

#include <iostream>

#include <string>

#include "Structures.h"

#include <Windows.h>

#include "FileModule.h"

using namespace std;

void ExportTestMenu(Test test, User\* user) {

}

Файл FileModule.h

#pragma once

#include "Structures.h"

int WriteToFile(string file\_path, User user);

int WriteToFile(string file\_path, Test test);

int WriteToFile(string file\_pass, UserData user);

int WriteToFile(string file\_pass, string test\_name, string user\_name, int result, float mark);

int UpdateTest(Test\*test, Test test1,User user, string file\_path);

int DeleteTest(Test\* test, User user, string file\_path);

int CreateRaiting(string file\_path, string test\_name, string user\_name);

UserData getUserData(int id, string file\_path);

User FindUser(string file\_path, string login);

int getLastId(string path);

TestsContainer\* Read(string test\_path, bool isPublic,User user);

Raiting\* Read(string test\_path, Test\* test, User\* user);

int ExportTest(Test test, User\* user, string file\_path, bool withAnsvers);

UsersContainer\* ReadUsers();

Файл FileModule.cpp

#include <iostream>

#include <string>

#include <fstream>

#include "Structures.h"

#include "ControlTestModule.h"

using namespace std;

// запись в файл пользователя

int WriteToFile(string file\_path, User user){

ofstream file(file\_path, ios::app);

if (!file.is\_open()) return 1;

file << user.id << " " << user.login << " " << user.password << " " << user.ruleLevel << endl;

return 0;

}

// запись в файл теста

int WriteToFile(string file\_path, Test test) {

ofstream file(file\_path,ios::app);

if (!file.is\_open()) return 1;

file << test.name << endl << test.author<< endl << test.difficulty<< endl << test.isPublic << endl;

file << test.count\_q\_standart<< endl;

for (int i = 0; i < test.count\_q\_standart; i++) {

file << test.q\_standart[i].question << endl;

file << test.q\_standart[i].answer << endl;

file << test.q\_standart[i].difficulty<< endl;

}

file << test.count\_q\_guest<< endl;

for (int i = 0; i < test.count\_q\_guest; i++) {

file << test.q\_guest[i].question<< endl;

file << test.q\_guest[i].right\_answer<< endl;

file << test.q\_guest[i].count\_answers << endl;

for (int j = 0; j < test.q\_guest[i].count\_answers; j++) {

file << test.q\_guest[i].answer[j]<< endl;

}

file << test.q\_guest[i].difficulty << endl;

}

file.close();

return 0;

}

// запись в файл рейтинга

int WriteToFile(string file\_pass, string test\_name, string user\_name, int result,float mark) {

fstream file(file\_pass, ios::in);

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

cout << "Ошибка открытия файла" << endl;

return 1;

}

string test\_name1;

int count\_user;

string user\_name1;

int result1;

getline(file,test\_name1) ;

if (test\_name1 == test\_name) {

file >> count\_user;

RaitingFileModule\* raiting = new RaitingFileModule[count\_user];

for (int i = 0; i < count\_user; i++) {

file >> user\_name1 >> result1;

raiting[i] = RaitingFileModule(user\_name1, result1);

}

file.close();

file.open(file\_pass, ios::out | ios::trunc);

file << test\_name<< endl << count\_user + 1 << endl;

for (int i = 0; i < count\_user; i++) {

file << raiting[i].user\_name << endl << raiting[i].result << endl;

}

file << user\_name << endl << result << endl;

file.close();

return 0;

}

else {

cout << "Тест не найден" << endl;

file.close();

file.open(file\_pass, ios::out | ios::trunc);

file << test\_name << endl << 1 << endl;

file << user\_name << endl << result << endl;

file.close();

return 0;

}

}

// запись в файл данных пользователя

int WriteToFile(string file\_pass, UserData user) {

ofstream file(file\_pass, ios::app);

if (!file.is\_open()) return 1;

file << user.id << " " << user.name << " " << user.surname << " " << user.login << " " << user.age << " " << user.count\_tests ;

for (int i = 0; i < user.count\_tests; i++) {

file << "name";

}

file << endl;

return 0;

}

// find user in file (login)

User FindUser(string file\_path, string login) {

ifstream file(file\_path);

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

cout << "Ошибка открытия файла" << endl;

return User();

}

User user;

while (!file.eof()) {

file >> user.id >> user.login >> user.password >> user.ruleLevel;

if (user.login == login) {

file.close();

return user;

}

}

file.close();

return User();

}

// find user in file (id)

UserData getUserData(int id,string file\_path) {

ifstream file(file\_path);

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

cout << "Ошибка открытия файла" << endl;

return UserData();

}

UserData userData;

while (!file.eof()) {

file >> userData.id >> userData.name >> userData.surname >> userData.login >> userData.age >> userData.count\_tests;

if (userData.id == id) {

file.close();

return userData;

}

}

return UserData();

}

// find test in file

int getLastId(string path) {

ifstream file(path);

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

cout << "Ошибка открытия файла" << endl;

return 0;

}

int id = 0;

User user;

while (!file.eof()) {

file >> user.id >> user.login >> user.password >> user.ruleLevel;

id = user.id;

}

return id;

}

//read test from file

TestsContainer\* Read(string test\_path,bool isPublict,User user) {

//

}

//read raiting from file

Raiting\* Read(string test\_path, Test\* test, User\* user) {

Raiting\* raiting = new Raiting();

fstream file(test\_path, ios::in);

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

cout << "Ошибка открытия файла рейтинга" << endl;

return raiting;

}

string test\_name;

int count\_user;

string user\_name;

int result;

getline(file, test\_name);

file >> count\_user;

if (test\_name == test->name) {

for (int i = 0; i < count\_user; i++) {

file >> user\_name >> result;

raiting->push(user\_name, getMark(result), result);

}

}

else {

cout << "Тест не найден" << endl;

}

raiting->print();

file.close();

return raiting;

}

int UpdateTest(Test\* test, Test test\_modified,User user,string file\_path) {

TestsContainer \* tests = Read(file\_path, true, user);

int index = -1;

for (int i = 0; i < tests->count\_tests; i++) {

if (tests->tests[i].name == test->name) {

index = i;

break;

}

}

tests->tests[index] = test\_modified;

fstream file(file\_path, ios::out | ios::trunc);

if(!file.is\_open()) return 1;

for (int i = 0; i < tests->count\_tests; i++) {

WriteToFile(file\_path, tests->tests[i]);

}

file.close();

return 0;

}

int DeleteTest(Test\* test, User user, string file\_path) {

TestsContainer\* tests = Read(file\_path, true, user);

int index = -1;

for (int i = 0; i < tests->count\_tests; i++) {

if (tests->tests[i].name == test->name) {

index = i;

break;

}

}

if (index == -1) return 1;

Test\* temp = new Test[tests->count\_tests - 1];

int j = 0;

for (int i = 0; i < tests->count\_tests; i++) {

if (i == index) continue;

temp[j++] = tests->tests[i];

}

fstream file(file\_path, ios::out | ios::trunc);

if (!file.is\_open()) return 1;

for (int i = 0; i < tests->count\_tests - 1; i++) {

WriteToFile(file\_path, temp[i]);

}

file.close();

return 0;

}

int CreateRaiting(string file\_path, string test\_name, string user\_name) {

fstream file(file\_path, ios::out | ios::trunc);

if(!file.is\_open()) return 1;

file << test\_name << endl << 0 << endl;

file.close();

return 0;

}

int ExportTest(Test test,User\* user, string file\_path,bool withAnsvers) {

//

}

UsersContainer\* ReadUsers() {

fstream file("users/usersData.txt", ios::in);

if (!file.is\_open()) return nullptr;

UsersContainer\* data = new UsersContainer();

int id;

string name;

string surname;

string login;

int age;

string email;

string temp\_cin;

while (!file.eof()) {

file >> temp\_cin;

try {

id = stoi(temp\_cin);

}

catch (const exception) {

cout << "Ошибка чтения пользователей";

return nullptr;

}

file >> name >> surname >> login;

file >>temp\_cin;

try {

age = stoi(temp\_cin);

}

catch (const exception) {

cout << "Ошибка чтения пользователей";

return nullptr;

}

file >> email;

data->add(new UserNode(UserData(id,name, login, surname, email, age)));

}

file.close();

return data;

}

Файл RegisterModule.h

#pragma once

void RegisterModule();

string customHash(const string& input);

User Login();

Файл RegisterModule.cpp

#include <iostream>

#include "Structures.h"

#include "FileModule.h"

#include <string>

#include <windows.h>

using namespace std;

string customHash(const string& input) {

// Шаг 1: Инициализация начального значения хеша

unsigned long long hash = 0;

// Шаг 2: Перемешивание символов входной строки

for (char c : input) {

hash ^= static\_cast<unsigned long long>(c);

hash = (hash << 7) | (hash >> 57); // сдвиг влево на 7 бит и сдвиг вправо на 57 бит

hash += 0xabcdef1234567890; // добавляем произвольное число

}

// Шаг 3: Применяем простую хеш-функцию

hash ^= (hash << 13);

hash ^= (hash >> 47);

hash ^= (hash << 23);

// Шаг 4: Повторяем преобразования несколько раз

for (int i = 0; i < 10; ++i) {

hash = ((hash << 5) ^ (hash >> 59)) \* 2654435761;

}

// Шаг 5: Преобразуем значение хеша в строку и возвращаем

return to\_string(hash);

}

void RegisterModule() {

while(true){

system("cls");

cout << "===========================" << endl;

cout << "| Регистрация" << endl;

cout << "| Введите логин: ";

string login;

cin >> login;

cout << "| Введите пароль: ";

string password;

cin.ignore();

getline(cin, password);

// Шифрование пароля

password = customHash(password);

cout << "| Введите имя: ";

string name;

cin >> name;

cout << "| Введите фамилию: ";

string surname;

cin >> surname;

cout << "| Введите email: ";

string email;

cin >> email;

cout << "| Введите возраст: ";

int age;

string temp;

cin.ignore();

getline(cin, temp);

try {

age = stoi(temp);

}

catch (exception e) {

cout << "Неверный ввод" << endl;

Sleep(1000);

continue;

}

string UserPath = "users/users.txt";

string UserDataPath = "users/usersData.txt";

User::count\_users = getLastId(UserPath);

User user = User(login, password, 1);

UserData userData = UserData(name, login, surname, email, age);

int status = WriteToFile(UserPath, user);

if (status == 0)

cout << "Пользователь успешно добавлен в базу" << endl;

else

cout << "Ошибка в работе в файлами" << endl;

status = WriteToFile(UserDataPath, userData);

if (status == 0)

cout << "Успешная регистрация" << endl;

else

cout << "Ошибка в работе в файлами" << endl;

Sleep(1000);

break;

}

}

User Login() {

while (true) {

system("cls");

cout << "===========================" << endl;

cout << "| Вход " << endl;

string login;

string password;

cout << "| Введите логин:";

cin >> login;

cout << "| Введите пароль: ";

cin >> password;

// Шифрование пароля

password = customHash(password);

string UserPath = "users/users.txt";

string UserDataPath = "users/usersData.txt";

User user = FindUser(UserPath, login);

if (password != user.password) {

cout << "Неверный логин или пароль" << endl;

Sleep(1000);

continue;

}

else {

UserData usd = getUserData(user.id, UserDataPath);

cout << "Вход выполнен" << endl;

Sleep(1000);

cout << "Вы вошли как " << usd.name << endl;

cout << "=========================" << endl;

return user;

}

}

}

Файл Structures.cpp

#include "Structures.h"

#include <string>

#include <iostream>

using namespace std;

Q\_guest::Q\_guest() {

this->question = "";

this->answer = new string[0];

this->count\_answers = 0;

this->right\_answer = 0;

this->difficulty = 0;

}

Q\_guest::Q\_guest(string question, string\* answer,int count\_answers, int right\_answer, int difficulty) {

this->question = question;

this->answer = answer;

this->count\_answers = count\_answers;

this->right\_answer = right\_answer;

this->difficulty = difficulty;

}

Q\_standart::Q\_standart() {

this->question = "";

this->answer = "";

this->difficulty = 0;

}

Q\_standart::Q\_standart(string question, string answer, int difficulty) {

this->question = question;

this->answer = answer;

this->difficulty = difficulty;

};

Test::Test() {

}

Test::Test(int size) {

this->count\_q\_standart = size;

this->q\_standart = new Q\_standart[size];

this->count\_q\_guest = size;

this->q\_guest = new Q\_guest[size];

this->isPublic = false;

this->difficulty = 0;

this->name = "";

this->author = "";

}

Test::Test(string name, string author, bool isPublic, int difficulty, int count\_q\_standart, Q\_standart\* q\_standart, int count\_q\_guest, Q\_guest\* q\_guest) {

this->name = name;

this->author = author;

this->isPublic = isPublic;

this->difficulty = difficulty;

this->count\_q\_standart = count\_q\_standart;

this->q\_standart = q\_standart;

this->count\_q\_guest = count\_q\_guest;

this->q\_guest = q\_guest;

}

User::User() {

this->ruleLevel = 0;

this->login = "";

this->password = "";

this->id = -1;

}

User::User(string login, string password, int ruleLevel) {

this->login = login;

this->password = password;

this->ruleLevel = ruleLevel;

this->id = ++count\_users;

}

int User::count\_users = 0;

UserData::UserData() {

this->name = "";

this->surname = "";

this->login = "";

this->age = 0;

this->email = "";

this->count\_tests = 0;

this->tests = new Test[count\_tests];

this->id = User::count\_users;

}

UserData::UserData(string name, string login, string surname, string email ,int age) {

this->name = name;

this->surname = surname;

this->login = login;

this->age = age;

this->email = email;

this->count\_tests = 0;

this->tests = new Test[count\_tests];

this->id = User::count\_users;

}

void UserData::addtest(Test test) {

tests[this->count\_tests++] = test;

};

UserData::UserData(int id, string name, string login, string surname, string email, int age) {

this->name = name;

this->surname = surname;

this->login = login;

this->age = age;

this->email = email;

this->count\_tests = 0;

this->tests = new Test[count\_tests];

this->id = id;

}

Raiting::Raiting() {}

Raiting::Raiting(string test\_name) {

this->test\_name = test\_name;

this->count\_users = 0;

this->middle\_result = 0;

this->head = nullptr;

}

void Raiting::push(string user\_name, int mark, int result) {

Raiting\_node \* new\_node = new Raiting\_node(user\_name, mark, result, this->head);

this->head = new\_node;

this->count\_users++;

this->middle\_result = (middle\_result\*(count\_users-1) + result) / count\_users;

}

void Raiting::swap(Raiting\_node\* prev, Raiting\_node\* curr, Raiting\_node\* next, Raiting\_node\* H) {

if (curr != nullptr && next != nullptr && curr->result < next->result) {

if (prev == nullptr) {

Raiting\_node\* temp = curr;

curr->next = next->next;

next->next = temp;

head = next;

}

else {

Raiting\_node\* temp = curr;

prev->next = curr->next;

temp->next = temp->next->next;

prev->next->next = temp;

}

}

else {

return;

}

}

// шейкерная сортировка

void Raiting::BubleSort() {

if (head == nullptr) return;

Raiting\_node\* prev;

Raiting\_node\* curr;

for (int i = 0; i < count\_users; i++) {

prev = nullptr;

curr = head;

for (int j = 0; j < count\_users; j++) {

if (curr == nullptr) break;

swap(prev, curr, curr->next, head);

prev = curr;

curr = curr->next;

}

for(int j = count\_users; j > 0; j--) {

if (curr == nullptr) break;

swap(prev, curr, curr->next, head);

prev = curr;

curr = curr->next;

}

}

}

void Raiting::print() {

Raiting\_node\* temp = head;

while (temp != nullptr) {

cout << temp->user\_name << " " << temp->result << " " << temp->mark << endl;

temp = temp->next;

}

}

Raiting\_node::Raiting\_node() {}

Raiting\_node::Raiting\_node(string user\_name, int result, int mark) {

this->user\_name = user\_name;

this->result = result;

this->mark = mark;

this->next = nullptr;

}

Raiting\_node::Raiting\_node(string user\_name, int mark , int reuslt, Raiting\_node \*next) {

this->user\_name = user\_name;

this->mark = mark;

this->result = reuslt;

this->next = next;

}

TestsContainer::TestsContainer() {}

TestsContainer::TestsContainer(Test\* tests, int size) {

this->tests = tests;

this->count\_tests = size;

}

RaitingFileModule::RaitingFileModule() {}

RaitingFileModule::RaitingFileModule(string user\_name, int result) {

this->user\_name = user\_name;

this->result = result;

}

UsersContainer::UsersContainer() {

this->head = nullptr;

}

void UsersContainer::add(UserNode\* node) {

if (this->head == nullptr) {

this->head = node;

}

else {

UserNode\* temp = this->head;

while (temp->next != nullptr) {

temp = temp->next;

}

temp->next = node;

}

}

UserNode::UserNode() {}

UserNode::UserNode(UserData data) {

this->data = data;

this->next = nullptr;

}