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](#_Toc164682096)

[1 Требования к программе 4](#_Toc164682097)

[2 Конструирование программы 7](#_Toc164682098)

[2.1 Описание модулей программы 7](#_Toc164682099)

[2.2 Выбор способа организации данных 8](#_Toc164682100)

[2.3 Разработка перечня пользовательских функций программы 10](#_Toc164682101)

[3 Разработка алгоритмов работы программы 12](#_Toc164682102)

[3.1 Алгоритм функции *main* 12](#_Toc164682103)

[3.2 Алгоритм функции поиска 14](#_Toc164682104)

[3.3 Алгоритм функции сортировки 15](#_Toc164682105)

[Раздел 4 19](#_Toc164682106)

[4.1 Авторизация 19](#_Toc164682107)

[4.2 Модуль администрирования 19](#_Toc164682108)

[4.3 Модуль пользователя 19](#_Toc164682109)

[4.4 Исключительные ситуации 19](#_Toc164682110)

[Заключение 20](#_Toc164682111)

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;
* возможность фильтровать тесты по разным параметрам;
* гость должен иметь возможность пройти любой тест из списка публичных тестов;
* администратор должен иметь возможность редактировать рейтинг.

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 и возможность менять рейтинг лютого теста.

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

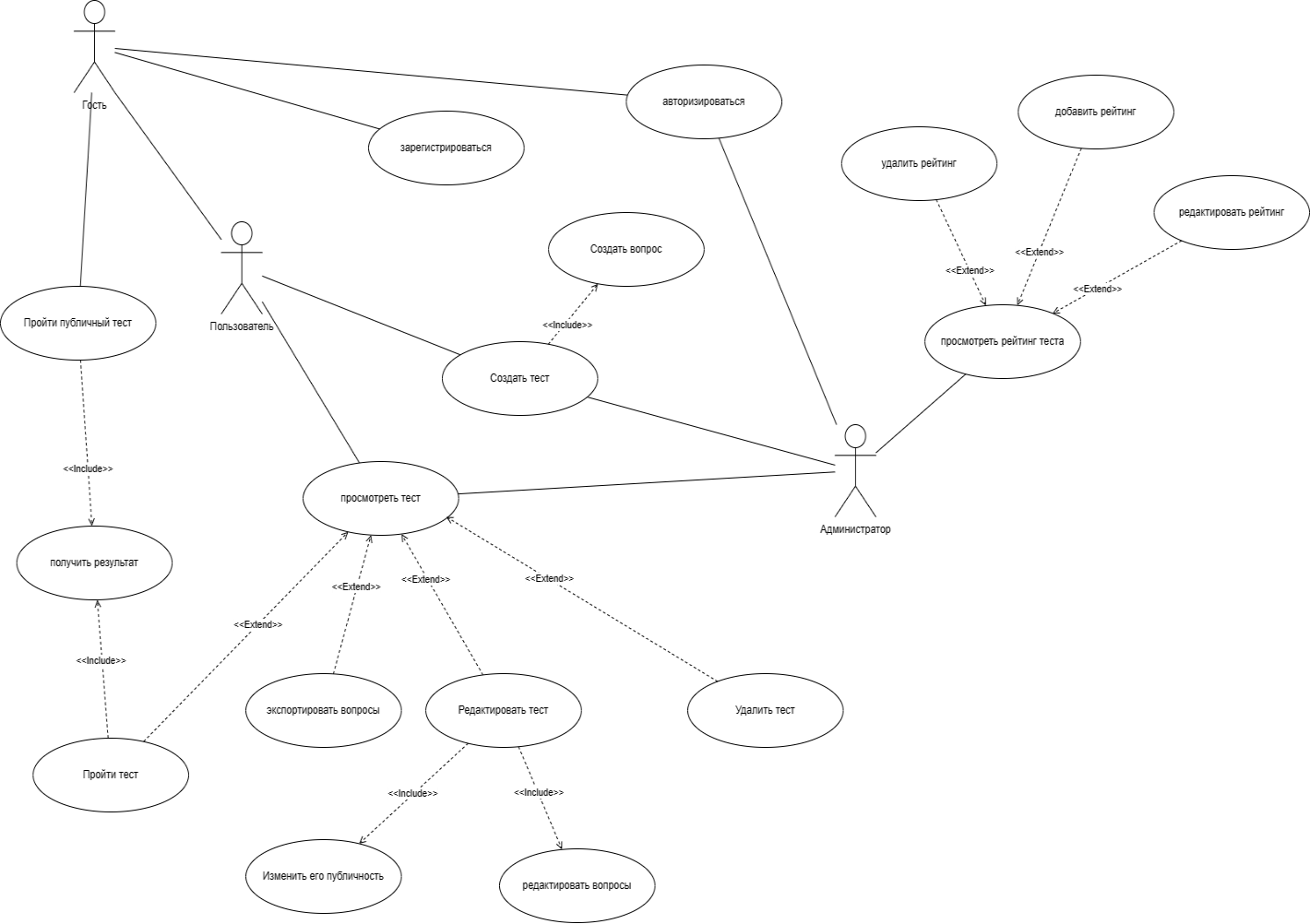


Рисунок 1.1 – *Use-case* диаграмма

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 | Test GetChoice() | Ничего не принимает. | Возвращает тест, который пользователь выбрал из перечня. |
| 2 | BiletsContainer CreateExamBilets(int num\_questions, int num\_bilets) | * int num\_questions (количество вопросов в билете); * int num\_bilets (Количество билетов). | Возвращает сгенерированные экзаменационные билеты. |
| 3 | Test EditTest(Test myTest) | * Test myTest (ранее созданный тест). | Редактирует выбранный тест и возвращает его. |
| 4 | Void RegisterModule() | * string username (Имя пользователя); * string password (Пароль). | Запускает модуль регистрации нового пользователя, сохраняя данные в файл. |
| 5 | User Login() | Ничего не принимает | Запускает взаимодействие через консоль и проверяет введенные данные пользователя и разрешает доступ к определенным функциям программы. |

Продолжение таблицы 2.2

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | string encrypt(string str) | * -string password (Пароль). | Функция, которая принимает строку, шифрует ее и возвращает зашифрованную строку. |
| 7 | Test CreateTest() | Ничего не принимает. | Отвечает за создание теста. |
| 8 | Int TryTest(Test test) | * Test test (ранее созданный тест). | Реализует прохождение теста и возвращает оценку за него. |
| 9 | Void ViewTest(Test test) | * Test test (ранее созданный тест). | Показывает тест с ответами. |
| 10 | User EditUser(User user) | * User user (зарегистрированный пользователь). | Отвечает за редактирование данных пользователя и его прав доступа (только для администратора). |
| 11 | Raiting EditRating(Test test) | * Test test (опубликованный тест). | Отвечает за редактирование рейтинга определенного теста (только для создателя теста). |
| 12 | Test Sort(Test test, string rule) | * Test \* tests (массив тестов); * string rule (правило сортировки). | Сортирует исходный массив тестов по определенному переданному правилу (переменной). |

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).

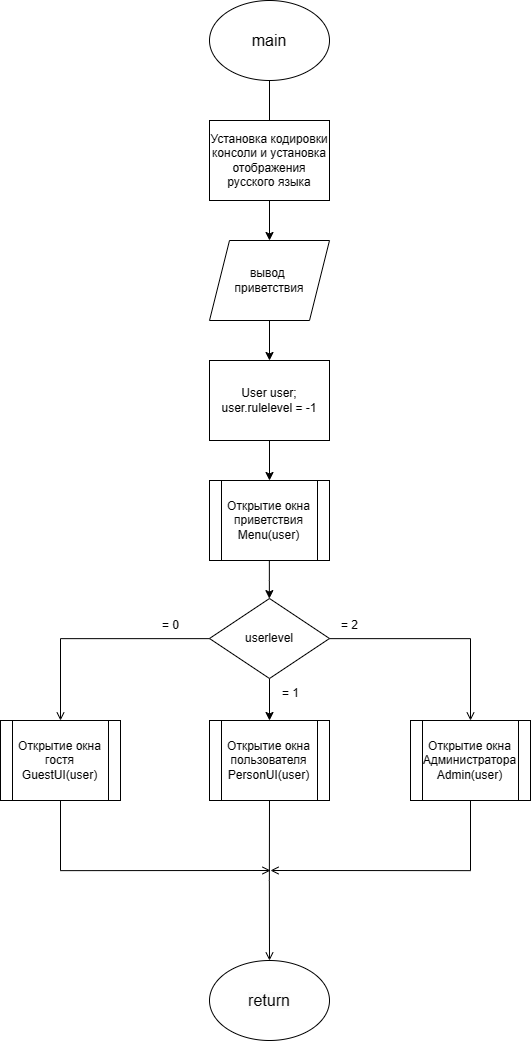


Рисунок 3.1 – Блок-схема алгоритма функции *main*

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).

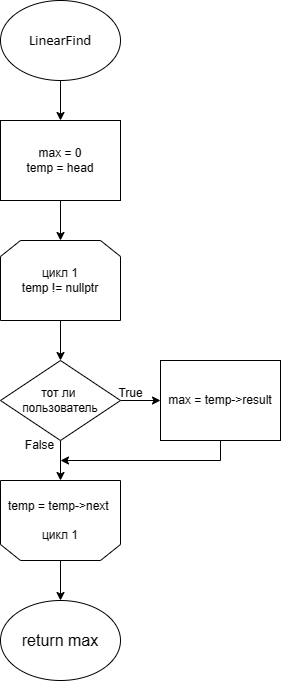


Рисунок 3.2 – Блок-схема алгоритма *LinearFind*

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).

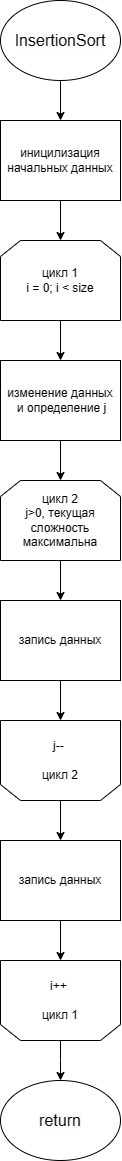


Рисунок 3.5 – Блок-схема алгоритма функции сортировки вставками

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).

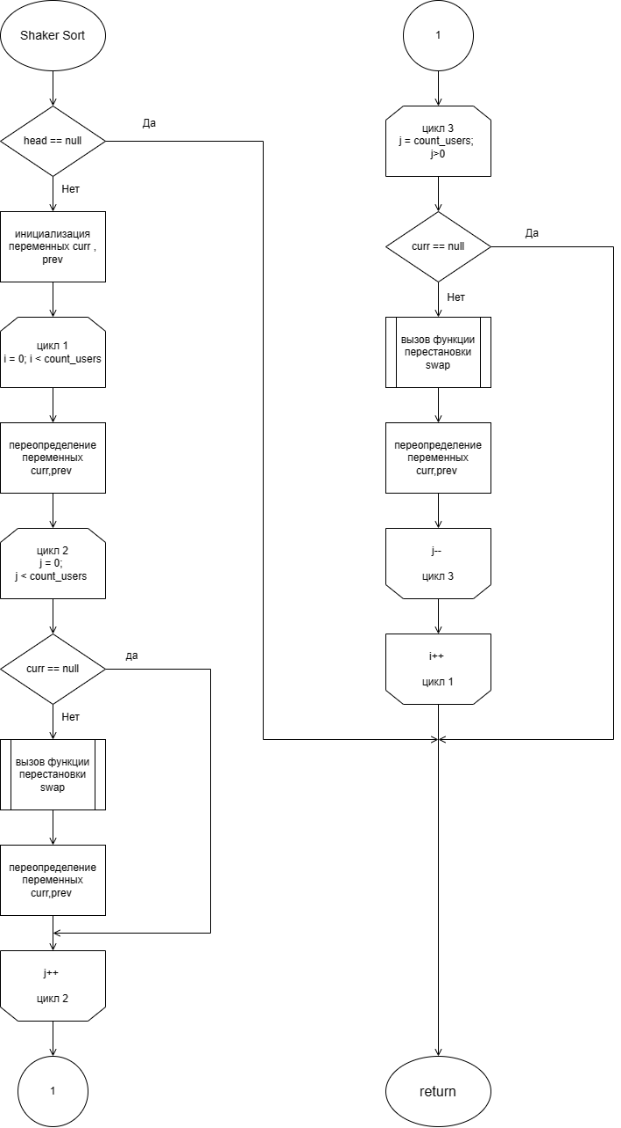


Рисунок 3.5 – Блок-схема алгоритма функции шейкерной сортировки

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

4.2 Модуль администрирования

4.3 Модуль пользователя

4.4 Исключительные ситуации

Заключение