МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГТУ»)

Факультет экономики, менеджмента и информационных технологий

Кафедра систем управления и информационных технологий в строительстве

КУРСОВОЙ ПРОЕКТ

по дисциплине: «Основы программирования и алгоритмизации»

Тема: «Разработка программы мониторинга и планирования маршрута лифта»

Расчетно-пояснительная записка

Разработал студент В.А. Никитин

Подпись, дата Инициалы, фамилия

Руководитель О.В. Минакова

Подпись, дата Инициалы, фамилия

Нормоконтролер О.В. Минакова

Подпись, дата Инициалы, фамилия

Защищена Оценка

дата

Воронеж 2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВА- ТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ВОРОНЕЖ- СКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГТУ»)

Кафедра систем управления и информационных технологий в строительстве

ЗАДАНИЕ

на курсовой проект

по дисциплине Основы программирования и алгоритмизации

Тема: «Разработка программы мониторинга и планирования маршрута лифта»

Студент группы бИЦ-241 Никитин Виталий Алексеевич

Фамилия, имя, отчество

База данных: «Этажи лифта», Признак: Этаж вызова, направление вызова, Вариант сортировки: частота вызова этажа.

Технические условия Windows 10, MicrosoftVisualStudio2019, язык программирования С

Содержание и объем проекта (графические работы, расчеты и прочее): 31 стр.,

9 рисунков, 2 таблиц, 1 приложение

Сроки выполнения этапов анализ и постановка задачи (10.9-5.10.24); разра- ботка пошаговой детализации программы (6.10 -11.11.24); реализация про- граммы (11.11-5.12.24); тестирование программы (6.12-11.12.24); оформле- ние пояснительной записки (11.12-14.12.24).

Срок защиты курсового проекта

Руководитель О.В. Минакова

Подпись, дата Инициалы, фамилия

Задание принял студент В.А. Никитин

Подпись, дата Инициалы, фамилия

Замечания руководителя

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc186070873)

[1. Постановка задачи 6](#_Toc186070874)

[2 Реализация программы 9](#_Toc186070875)

[2.1 Выбор средств реализации 9](#_Toc186070876)

[2.2 Анализ кода программы 10](#_Toc186070877)

[3. Тестирование программы 17](#_Toc186070878)

[Заключение 20](#_Toc186070879)

[Список используемых источников 22](#_Toc186070880)

[ПРИЛОЖЕНИЕ А 23](#_Toc186070881)

# ВВЕДЕНИЕ

В современном мире управление лифтами является важной частью инфраструктуры зданий, особенно в многоэтажных домах и коммерческих центрах. Эффективное управление вызовами лифта не только повышает комфорт пользователей, но и оптимизирует работу лифта, снижая время ожидания и энергозатраты. В связи с этим, разработка программного обеспечения для управления вызовами лифта становится актуальной задачей.

Данная программа предназначена для моделирования работы лифта с возможностью обработки вызовов от пользователей. Программа использует текстовый файл в качестве базы данных для хранения информации о вызовах и предоставляет пользователю удобный интерфейс для взаимодействия.

Целью данной курсовой работы является разработка программного обеспечения для управления вызовами лифта с использованием языка программирования C. Программа должна обеспечивать автоматизированный сбор данных и обработку этих данных, и визуализацию информации для пользователя.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Обосновать выбор структуры данных для хранения отдельных записей в файле, а также формат файла.

2. Реализовать простой и понятный интерфейс для взаимодействия

пользователя с программой, который будет работать пока пользователь не захочет выйти из программы.

3. Обеспечить выполнение программой функции создания записей.

4. Разработать функционал для анализа данных о вызовах, включая подсчет общего количества вызовов, количество вызовов вверх и вниз, а также определение самого часто вызываемого этажа.

# 1. Постановка задачи

Цель курсового проекта: разработать программное обеспечения для управления вызовами лифта с использованием языка программирования C. Программа должна обеспечивать автоматизированный сбор данных и обработку этих данных, и визуализацию информации для пользователя.

Основные задачи проекта:

1. Реализовать функцию для считывания этажей вызова с учетом ограничений (например, с минимального этажа можно вызывать только вверх, а с максимального — только вниз).

2. Определить структуру данных для хранения информации о вызовах лифта, включая этажи и направление движения.

3. Реализовать простой и понятный интерфейс для взаимодействия пользователя с программой, который будет работать до тех пор, пока пользователь не захочет выйти.

4. Обеспечить выполнение программой функции создания записей о вызовах лифта и сохранения их в текстовый файл.

5. Разработать функционал для анализа данных о вызовах, включая подсчет общего количества вызовов, количество вызовов вверх и вниз, а также определение самого часто вызываемого этажа.

Подзадачи проекта:

Подзадача 1. Реализация алгоритма обработки вызовов лифта:

1) Организовать обработку входных данных с учетом правил вызова лифта.

2) Разработать алгоритмы для создания записей о вызовах и их сохранения в файл.

3) Реализовать функции для отображения информации о текущих запросах.

Подзадача 2. Создание интерфейса и взаимодействие с пользователем:

1. Разработать интерфейс для ввода этажей и направления вызова.
2. Обеспечить возможность генерации случайных вызовов лифта.

3) Реализовать циклическое выполнение программы до тех пор, пока пользователь не решит выйти.

Подзадача 3. Анализ данных о вызовах лифта:

1. Обеспечить сохранение данных о вызовах в текстовый файл.

2) Реализовать функции анализа данных, включая подсчет общего количества вызовов и определение самого часто вызываемого этажа.

Одной из целей данного проекта является создание структуры данных, которая будет отражать предметную область «Управление вызовами лифта». База данных включает в себя следующие элементы:

1) Этажи вызова: Целое число, представляющее этаж, на который необходимо поднять лифт или с которого его вызывают.

2) Направление движения: Целое число, указывающее направление вызова лифта.

Каждый из этих элементов представляет собой специфическую характеристику и должен быть представлен посредством конкретного типа данных. Таблица 1 содержит перечень типов данных:

Таблица 1 - Типы данных для полей структуры

|  |  |  |
| --- | --- | --- |
| Поле структуры | Описание | Тип данных |
| Этаж вызова | Содержит этаж, на который лифт должен подняться или с которого его вызывают | int |
| Направление | Содержит направление вызова лифта (вверх или вниз) | int |

Информация о добавленных данных хранится в массиве структур ElevatorCall, где каждая структура содержит информацию о конкретном вызове. Размер массива не должен быть ограничен одним числом, для этого необходимо динамическое распределение памяти, которое позволит выделять именно столько памяти, сколько требуется на текущий момент.

Кроме того, для хранения запросов на подъем и спуск используются два отдельных массива в структуре RequestData. Это позволяет удобно управлять запросами и производить их анализ.

Для удобства работы с программой была разработана система меню, включающая основные функции:

1. Включение режима мониторинга. Данная функция отвечает за запуск режима мониторинга вызовов лифта.

2. Выключение режима мониторинга. Этот пункт меню завершает процесс мониторинга и отключает программу от обработки вызовов лифта.

3. Ручной ввод данных. Пользователь может ввести этажи, на которые необходимо поднять лифт или с которых его вызывают. Программа проверяет корректность введенных данных и сохраняет их для дальнейшей обработки.

4. Случайная генерация данных. При выборе этой функции программа генерирует случайные вызовы лифта и сохраняет их в файл «test\_data.txt.».

5. Вывод запросов. Эта функция позволяет пользователю отобразить текущие запросы на подъем и спуск, которые были введены или сгенерированы ранее.

6. Обработка запросов лифта. Программа обрабатывает все запросы на подъем и спуск, моделируя движение лифта по этажам в соответствии с введенными данными.

7. Анализ данных о вызовах. Данная операция предоставляет возможность анализа ранее собранных данных о вызовах лифта. Программа считывает данные из файла и выполняет их анализ, включая подсчет общего количества вызовов, количество вызовов вверх и вниз, а также определение самого часто вызываемого этажа.

8. Выход. Этот пункт завершает работу программы. Функция необходима для корректного завершения работы приложения с возможностью сохранения данных, если режим мониторинга был активен.

При выборе функции «Случайная генерация данных», в консоли появится сообщение о том, что данные успешно сохранены в файл test\_data.txt, который будет содержать информацию о всех вызовах лифта, контрольный пример показан в таблице 2:

Таблица 2 - Контрольный пример сгенерированных данных

|  |  |  |
| --- | --- | --- |
| **№ строки** | **Этаж вызова** | **Направление** |
| 1 | 3 | Вверх |
| 2 | 5 | Вверх |
| 3 | -7 | Вниз |
| 4 | -2 | Вниз |
| 5 | 5 | Вверх |

# 2 Реализация программы

## 2.1 Выбор средств реализации

В данной главе будет подробно рассмотрена реализация программы управления вызовами лифта, разработанной с использованием языка программирования Си. Язык Си был выбран благодаря своей высокой производительности, гибкости и широкому применению в разработке системного и прикладного программного обеспечения. Он предоставляет разработчику низкоуровневый доступ к памяти и ресурсам системы, что позволяет создавать эффективно работающие и оптимизированные решения.

Кроме того, язык Си обладает обширной стандартной библиотекой функций, что значительно упрощает процесс разработки. Его популярность и активное сообщество обеспечивают доступ к множеству ресурсов и документации, что делает его идеальным выбором для образовательных и научных проектов. Использование языка Си позволяет легко управлять памятью и ресурсами, что особенно важно для приложений, требующих высокой скорости обработки данных, таких как управление вызовами лифта.

Для разработки программы была выбрана среда Microsoft Visual Studio 2019. Эта интегрированная среда разработки (IDE) предлагает мощные инструменты для написания, отладки и тестирования кода, что значительно ускоряет процесс разработки. Удобный интерфейс и поддержка современных стандартов языка Си делают работу в данной среде комфортной и продуктивной. Кроме того, Visual Studio предоставляет возможности для интеграции с системами контроля версий, что облегчает командную работу над проектом.

Таким образом, использование языка Си в сочетании с Microsoft Visual Studio 2019 обеспечило надежную базу для реализации программы управления вызовами лифта. Это позволило создать эффективное решение, способное обрабатывать запросы пользователей и анализировать данные о вызовах в реальном времени.

## 2.2 Анализ кода программы

В данной курсовой работе разработана программа на языке C, предназначенная для управления вызовами лифта и работы с данными, связанными с ними. Программа включает функции для ввода, обработки и анализа данных о вызовах, что позволяет эффективно управлять информацией о лифте.

Реализация программы начинается с подключения необходимых библиотек. Библиотека stdio.h используется для работы с вводом и выводом данных. Она предоставляет функции, такие как printf для вывода информации и scanf для ввода данных от пользователя. Библиотека stdlib.h включает функции для управления памятью и другими утилитами, что важно для динамического распределения памяти в программе. Эта библиотека также предоставляет функции для генерации случайных чисел, что используется при создании случайных вызовов лифта. Библиотека locale.h помогает управлять локализацией.

Следующим шагом становится определение ключевых констант, которые будут использоваться в программе. Здесь создаются несколько констант, таких как MIN\_FLOOR и MAX\_FLOOR, которые определяют минимальный и максимальный этажи, а также MAX\_REQUESTS, задающий максимальное количество запросов лифта. INITIAL\_COUNT начальное значение счетчика вызовов, INITIAL\_COUNT\_UP 0: Начальное значение счетчика для запросов на подъем, INITIAL\_COUNT\_DOWN начальное значение счетчика для запросов на спуск, INITIAL\_COUNT\_DATA счетчик прочитанных данных из файла. Эти ограничения помогают избежать ошибок в программе и улучшают ее стабильность в работе.

Затем в коде определяется структура ElevatorCall, которая предназначена для хранения информации о вызове лифта. Эта структура включает два поля: floor, представляющее этаж вызова (типа int), и direction, указывающее направление движения лифта (типа int). Также определяется структура RequestData, которая содержит два массива для хранения запросов на подъем и спуск, а также счетчики этих запросов.

Далее объявлены функции, которые помогут в работе с данными о вызовах лифта. Функция int readUserInput(ElevatorCall\* calls) отвечает за считывание этажей вызова от пользователя. Она запрашивает ввод целых чисел, представляющих этажи, на которые необходимо вызвать лифт. В процессе работы функция проверяет корректность введенных данных, включая диапазон этажей и специальные условия (например, что с минимального этажа можно вызывать только вверх, а с максимального — только вниз). В конце выполнения функция возвращает количество успешно введенных вызовов.

Функция void displayInput(const ElevatorCall\* calls, int count) предназначена для отображения введенных пользователем данных о вызовах лифта. Она принимает массив, содержащий информацию о каждом вызове, и выводит на экран этажи вызова вместе с направлением движения (вверх или вниз). Это позволяет пользователю увидеть все текущие запросы и удостовериться в их правильности перед обработкой.

Функция void writeToFile(const ElevatorCall\* calls, int count) сохраняет данные о вызовах лифта в текстовый файл «test\_data.txt». Она принимает массив и количество вызовов, которые необходимо записать. В процессе записи функция формирует строки, представляющие каждый вызов, и добавляет их в файл, что позволяет сохранить информацию о вызовах для последующего анализа и обработки.

Функция void generateRandomCalls(ElevatorCall\* calls, int count) генерирует случайные вызовы лифта для заданного количества запросов. Она заполняет массив случайными этажами в диапазоне от минимального до максимального этажа и случайным направлением. Сгенерированные данные могут быть использованы для тестирования программы или для демонстрации ее работы без необходимости ручного ввода.

Функция int\* readFromFile(int\* totalCount) считывает данные о вызовах из текстового файла «test\_data.txt». Она открывает файл для чтения и динамически выделяет память под массив целых чисел, который будет хранить этажи вызова. Функция считывает данные из файла до тех пор, пока не достигнет конца файла или не заполнит массив до максимального размера. После завершения чтения функция возвращает указатель на массив с данными и записывает количество считанных элементов через указатель на переменную.

Функция void selectRequests(int\* total, int totalCount, RequestData\* requests) распределяет запросы по массивам запросов на подъем и спуск. Она принимает массив целых чисел (этажи вызова) и количество элементов в этом массиве, а также указатель на структуру RequestData, которая будет хранить результаты распределения. Функция проходит по каждому элементу массива и добавляет его в соответствующий массив (request\_up или request\_down) в зависимости от направления вызова (положительное значение для подъема и отрицательное для спуска).

Функция void displayRequests(const RequestData\* requests) отображает текущие запросы на подъем и спуск, хранящиеся в структуре RequestData. Она выводит на экран все этажи, на которые необходимо поднять лифт, а также все этажи, с которых лифт должен спуститься. Это позволяет пользователю видеть все активные запросы перед их обработкой.

Функция void sortRequests(int\* requests, int count, int (\*compare)(int a, int b)) сортирует массив запросов по заданному критерию (возрастание или убывание) с использованием переданной функции сравнения. Она принимает указатель на массив запросов, его размер и указатель на функцию   
сравнения как аргументы. Представим функцию sortRequests   
в виде блок-схемы (Рис. 1).

Функция int compareAscending(int a, int b) является функцией сравнения, используемой для сортировки массива этажей по возрастанию. Она принимает два целых числа как аргументы и возвращает разницу между ними. Если первое число меньше второго, возвращается отрицательное значение; если они равны — ноль; если больше — положительное значение.

Функция int compareDescending(int a, int b) выполняет аналогичную задачу, но сортирует массив этажей по убыванию. Она также принимает два целых числа и возвращает разницу между ними в обратном порядке: если первое число больше второго, возвращается отрицательное значение; если они равны — ноль; если меньше — положительное значение.

Функция void processRequests(int\* request\_up, int upCount, int\* request\_down, int downCount) обрабатывает запросы на подъем и спуск лифта, моделируя его движение по этажам согласно введенным данным. Она принимает два массива (для подъема и спуска) и их размеры как аргументы. Функция последовательно обрабатывает запросы: сначала выполняет все подъемы до самого высокого этажа из списка запросов вверх, затем спускается до всех запрашиваемых этажей вниз. Представим функцию processRequests   
в виде блок-схемы (Рис. 2).

Функция void analyzeData(int\* total, int totalCount) выполняет анализ собранных данных о вызовах лифта из массива целых чисел. Она подсчитывает общее количество вызовов, определяет количество   
вызовов вверх и вниз, а также находит самый часто вызываемый этаж   
среди всех записей. Результаты анализа выводятся пользователю   
в удобной форме. Представим функцию analyzeData в виде   
блок-схемы (Рис.3).

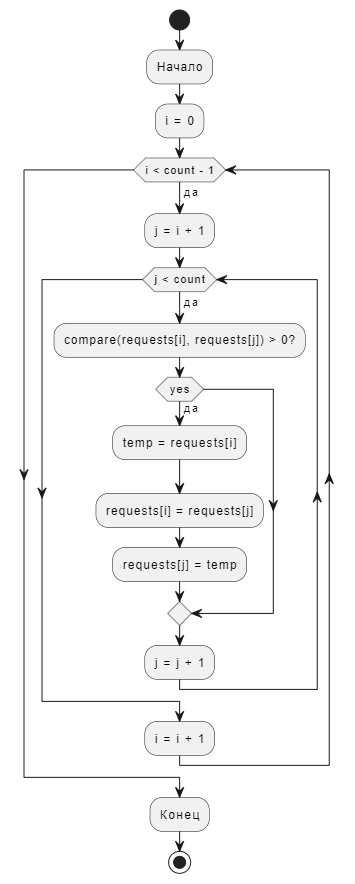
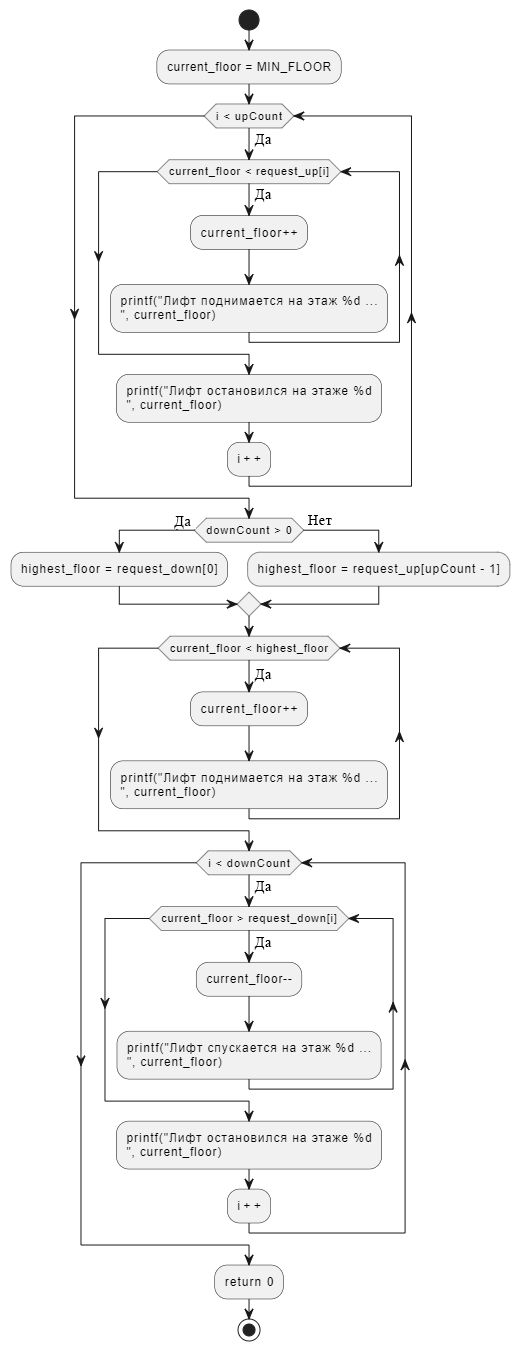
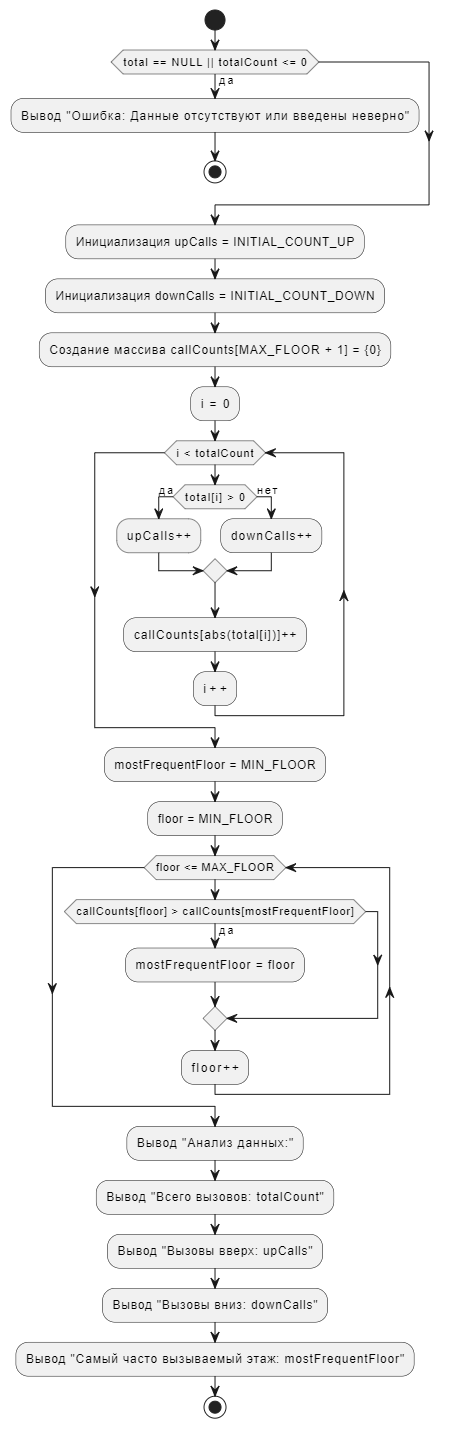
  
Рисунок 1 – Блок-схема функции sortRequests  
Рисунок 2 – Блок-схема функции processRequests

Рисунок 3 – Блок-схема функции analyzeData

# 3. Тестирование программы

В этой главе будет представлено детальное описание процесса проверки корректности работы разработанного программного обеспечения для управления вызовами лифта. Ключевым этапом тестирования является наблюдение за тем, как программа обрабатывает вводимые данные и какие результаты она выводит на экран.

Основная цель тестирования заключается в выявлении возможных ошибок или недочетов в логике работы приложения, а также в проверке его функциональности на контрольном примере. Тестирование позволяет не только подтвердить правильность выполнения программы в стандартных условиях, но и оценить ее устойчивость к некорректным и крайним значениям, что является важным аспектом для обеспечения надежности и стабильности программного обеспечения.

При запуске программы открывается консольное меню, где пользователю предлагается выбрать одно из доступных действий (см. рис. 4)

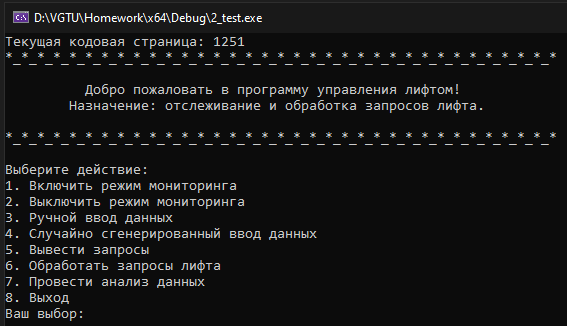


Рисунок 4 – Главное меню и приветствие пользователя

Рассмотрим контрольный пример, который демонстрирует основные функции программы. Начнем с включения режима мониторинга, что позволит программе начать обработку вызовов и взаимодействие с пользователем. Затем перейдем к генерации данных, где создадим случайные вызовы лифта, что упростит тестирование. После этого мы выполним ввод количества вызовов, что даст пользователю возможность задать необходимое количество запросов, которые будут обработаны программой. Далее, осуществим передачу данных в обработку, что позволит программе организовать и распределить запросы по этажам в соответствии с заданными параметрами.

Затем мы проведем анализ данных, который включает подсчет общего количества вызовов, определение направлений движения лифта и выявление самого часто вызываемого этажа. Наконец, завершив все операции, мы выполним выход из программы, что обеспечит корректное завершение работы приложения и сохранение всех необходимых данных. Этот контрольный пример наглядно продемонстрирует функциональность программы и ее способность эффективно управлять вызовами лифта.

Таблица 3 – Контрольный пример

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Действия | Ожидаемый результат | Фактический результат | Pass/Fail |
| Включение мониторинга | 1. Ввести 1 2. Нажать   Enter | Режим мониторинга включен. | Режим мониторинга включен. | Pass |
| Генерация данных | 1. Ввести 4 2. Нажать   Enter | Введите количество случайных вызовов для генерации: | Введите количество случайных вызовов для генерации: | Pass |
| Ввод количества | 1. Ввести 5 2. Нажать   Enter | Вызов с этажа 3. Направление: Вверх.  Вызов с этажа 5. Направление: Вверх.  Вызов с этажа 7. Направление: Вниз.  Вызов с этажа 2. Направление: Вниз.  Вызов с этажа 5. Направление: Вверх. | Вызов с этажа 3. Направление: Вверх.  Вызов с этажа 5. Направление: Вверх.  Вызов с этажа 7. Направление: Вниз.  Вызов с этажа 2. Направление: Вниз.  Вызов с этажа 5. Направление: Вверх. | Pass |

Продолжение таблицы 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Передача данных в обработку | 1. Ввести 6 2. Нажать   Enter | Лифт поднимается на этаж 2 ...  Лифт поднимается на этаж 3 ...  Лифт остановился на этаже 3  Лифт поднимается на этаж 4 ...  Лифт поднимается на этаж 5 ...  Лифт остановился на этаже 5  Лифт остановился на этаже 5  Лифт поднимается на этаж 6 ...  Лифт поднимается на этаж 7 ...  Лифт остановился на этаже 7  Лифт спускается на этаж 6 ...  Лифт спускается на этаж 5 ...  Лифт спускается на этаж 4 ...  Лифт спускается на этаж 3 ...  Лифт спускается на этаж 2 ...  Лифт остановился на этаже 2 | Лифт поднимается на этаж 2 ...  Лифт поднимается на этаж 3 ...  Лифт остановился на этаже 3  Лифт поднимается на этаж 4 ...  Лифт поднимается на этаж 5 ...  Лифт остановился на этаже 5  Лифт остановился на этаже 5  Лифт поднимается на этаж 6 ...  Лифт поднимается на этаж 7 ...  Лифт остановился на этаже 7  Лифт спускается на этаж 6 ...  Лифт спускается на этаж 5 ...  Лифт спускается на этаж 4 ...  Лифт спускается на этаж 3 ...  Лифт спускается на этаж 2 ...  Лифт остановился на этаже 2 | Pass |
| Анализ данных | 1. Ввести 7 2. Нажать   Enter | Анализ данных:  Всего вызовов: 5  Вызовы вверх: 3  Вызовы вниз: 2  Самый часто вызываемый этаж: 5 | Анализ данных:  Всего вызовов: 5  Вызовы вверх: 3  Вызовы вниз: 2  Самый часто вызываемый этаж: 5 | Pass |

Продолжение таблицы 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Выход из программы | 1. Ввести 7 2.Нажать   Enter | Выход из программы. | Выход из программы. | Pass |

# Заключение

В ходе выполнения этой курсовой работы была проведена комплексная разработка программы для управления вызовами лифта. Это позволило не только углубить знания в данной области, но и применить теоретические навыки на практике, а также улучшить навыки программирования.

В разработанной программе реализованы следующие функции:

1. Управление вызовами лифта: программа позволяет пользователю включать и отключать режим мониторинга вызовов, обеспечивая контроль за состоянием системы в режиме реального времени.

2. Ввод данных: пользователь может вручную вводить этажи вызова, что обеспечивает гибкость в управлении данными. Также предусмотрена функция автоматической генерации случайных вызовов, что упрощает процесс тестирования программы.

3. Сохранение данных: все введённые или сгенерированные данные сохраняются в текстовый файл «test\_data.txt», что позволяет вести учёт и анализировать информацию о вызовах лифта.

4. Сортировка запросов: реализована функция сортировки запросов по этажам, что упрощает их обработку и делает работу с данными более эффективной.

5. Анализ данных: программа предоставляет возможность анализировать собранные данные, позволяя пользователю получать информацию о количестве вызовов, количестве вызовов вверх и вниз, а также определять наиболее часто вызываемый этаж.

6. Проверка корректности ввода: введённые данные проверяются на соответствие установленным правилам, что предотвращает ошибки и повышает надёжность работы программы.

В заключение можно сказать, что этот работа стала важным этапом на пути к глубокому пониманию темы управления вызовами лифта и разработки качественного программного обеспечения. Все полученные знания и опыт окажут положительное влияние на дальнейшую учёбу и профессиональную деятельность.

Ссылка на код, размещенный на github: https://github.com/Mininils/KP/

# Список используемых источников

1. О.В. Минакова, О.В. Курипта. // Практикум по СИ URL: https://sites.google.com/view/course-of-study1-c/главная

2. M. УЭИТ С. ПРАТА Д. МАРТИН // Язык Си руководство для начинающих

3. Перри, Г. Программирование на C для начинающих // Г. Перри, Д. Миллер. - М.: Эксмо, 2015

4. Г. Шилдт // Полный справочник по CИ

5. Онлайн справочник программиста на C и C++ // URL: <http://www.c-cpp.ru/books>

6. Конспект лекций по C и C++ //   
URL: <https://natalia.appmat.ru/c&c++/index.php>

7. Глушков, В. Программирование на C // В. Глушков, А. Кузнецов. - М.: Бином. Лаборатория знаний, 2016

8. Беляев, А. Программирование на C для начинающих // А. Беляев, И. Сидоров. - М.: Питер, 2018

9. Савин, А., Гусев, И. Основы программирования на C // А. Савин, И. Гусев. - М.: Наука и техника, 2020

10. Керниган, Б., Ричи, Д. Язык программирования C // Б. Керниган, Д. Ричи. - М.: Эксмо, 2015

# ПРИЛОЖЕНИЕ А

(обязательное)

Листинг программы

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#define MIN\_FLOOR 1 // Минимальный этаж

#define MAX\_FLOOR 10 // Максимальный этаж

#define INITIAL\_COUNT 0 // Начальное значение счетчика

#define INITIAL\_COUNT\_UP 0 // Начальное значение счетчика вверх

#define INITIAL\_COUNT\_DOWN 0 // Начальное значение счетчика вниз

#define INITIAL\_COUNT\_DATA 0 // Cчетчик прочитанных данных

#define MAX\_REQUESTS 100 // Максимальное количество запросов

// Структура для хранения информации о вызове лифта

typedef struct {

int floor; // Этаж вызова

int direction; // Направление

} ElevatorCall;

// Структура для хранения запросов

typedef struct {

int request\_up[MAX\_REQUESTS]; // Массив для хранения этажей запросов вверх

int request\_down[MAX\_REQUESTS]; // Массив для хранения этажей запросов вниз

int upCount; // Количество запросов на подъем

int downCount; // Количество запросов на спуск

} RequestData;

/\*\*

\* Функция считывает данные от пользователя.

\*

\* @param calls Указатель на массив, в который будут записаны введенные данные.

\* @return Количество успешно введенных вызовов.

\*/

int readUserInput(ElevatorCall\* calls);

/\*\*

\* Функция отображает данные о вызовах лифта.

\*

\* @param calls Указатель на массив, содержащий данные о вызовах.

\* @param count Количество вызовов, которые необходимо отобразить.

\*/

void displayInput(ElevatorCall\* calls, int count);

/\*\*

\* Функция записывает данные о вызовах лифта в файл.

\*

\* @param calls Указатель на массив, содержащий данные о вызовах.

\* @param count Количество вызовов, которые необходимо записать в файл.

\*/

void writeToFile(ElevatorCall\* calls, int count);

/\*\*

\* Функция генерирует случайные вызовы лифта.

\*

\* @param calls Указатель на массив, в который будут записаны сгенерированные данные.

\* @param count Количество случайных вызовов, которые необходимо сгенерировать.

\*/

void generateRandomCalls(ElevatorCall\* calls, int count);

/\*\*

\* Функция читает данные из файла.

\*

\* @param totalCount Указатель на переменную, в которую будет записано количество прочитанных данных.

\* @return Указатель на массив целых чисел, содержащий данные о вызовах лифта.

\*/

int\* readFromFile(int\* totalCount);

/\*\*

\* Функция распределяет запросы по массивам запросов.

\*

\* @param total Указатель на массив целых чисел, содержащий данные о вызовах лифта.

\* @param totalCount Количество элементов в массиве total.

\* @param requests Указатель на структуру RequestData, в которую будут записаны распределенные запросы.

\*/

void selectRequests(int\* total, int totalCount, RequestData\* requests);

/\*\*

\* Функция отображает запросы.

\*

\* @param requests Указатель на структуру RequestData, содержащую запросы лифта.

\*/

void displayRequests(const RequestData\* requests);

/\*\*

\* Функция обрабатывает запросы лифта.

\*

\* @param request\_up Указатель на массив целых чисел с запросами на подъем.

\* @param upCount Количество запросов на подъем.

\* @param request\_down Указатель на массив целых чисел с запросами на спуск.

\* @param downCount Количество запросов на спуск.

\*/

void processRequests(int\* request\_up, int upCount, int\* request\_down, int downCount);

/\*\*

\* Функция сортирует массив запросов.

\*

\* @param requests Указатель на массив целых чисел с запросами для сортировки.

\* @param count Количество элементов в массиве requests.

\* @param compare Указатель на функцию сравнения для определения порядка сортировки (возрастание или убывание).

\*/

void sortRequests(int\* requests, int count, int (\*compare)(int a, int b));

/\*\*

\* Функция анализирует данные о вызовах лифта.

\*

\* @param total Указатель на массив целых чисел с данными о вызовах лифта.

\* @param totalCount Количество элементов в массиве total.

\*/

void analyzeData(int\* total, int totalCount);

/\*\*

\* Функция сравнения для сортировки по возрастанию.

\*

\* @param a Первое значение для сравнения.

\* @param b Второе значение для сравнения.

\* @return Отрицательное значение, если a < b; 0 если a == b; положительное значение если a > b.

\*/

int compareAscending(int a, int b);

/\*\*

\* Функция сравнения для сортировки по убыванию.

\*

\* @param a Первое значение для сравнения.

\* @param b Второе значение для сравнения.

\* @return Отрицательное значение, если a < b; 0 если a == b; положительное значение если a > b.

\*/

int compareDescending(int a, int b);

int main() {

setlocale(LC\_ALL, "RUS");

system("chcp 1251");

srand(time(NULL)); // Инициализация генератора случайных чисел

// Приветсвие пользователя (шапка программы)

puts("\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*");

printf("\n\t Добро пожаловать в программу управления лифтом!\n");

printf("\tНазначение: отслеживание и обработка запросов лифта.\n\n");

puts("\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*\_\*");

int monitoringMode = 0;

char choice;

ElevatorCall calls[MAX\_REQUESTS]; // Массив для хранения вызовов

do {

// Главное меню

printf("\nВыберите действие:\n");

printf("1. Включить режим мониторинга\n");

printf("2. Выключить режим мониторинга\n");

printf("3. Ручной ввод данных\n");

printf("4. Случайно сгенерированный ввод данных\n");

printf("5. Вывести запросы\n");

printf("6. Обработать запросы лифта\n"); // Новый пункт меню

printf("7. Провести анализ данных\n"); // Новый пункт меню

printf("8. Выход\n");

printf("Ваш выбор: ");

scanf(" %c", &choice);

switch (choice) {

case '1': // Включение мониторинга

monitoringMode = 1;

printf("Режим мониторинга включен.\n");

break;

case '2': // Выключение мониторинга

monitoringMode = 0;

printf("Режим мониторинга выключен.\n");

break;

case '3': // Ручной ввод данных

if (monitoringMode) {

int count = readUserInput(calls); // Считывание данных от пользователя

displayInput(calls, count); // Вывод введенных данных

writeToFile(calls, count); // Запись данных в файл

}

else {

printf("Режим мониторинга выключен. Включите его для ввода данных.\n");

}

break;

case '4': { // Генерация данных

if (monitoringMode) {

int count; // Считывание данных от пользователя

printf("Введите количество случайных вызовов для генерации: ");

scanf("%d", &count);

if (count > MAX\_REQUESTS) { // Проверка на превышение максимального количества запросов

printf("Ошибка: Максимальное количество вызовов - %d.\n", MAX\_REQUESTS);

break;

}

generateRandomCalls(calls, count); // Генерация случайных вызовов

displayInput(calls, count); // Вывод введенных данных

writeToFile(calls, count); // Запись данных в файл

}

else {

printf("Режим мониторинга выключен. Включите его для генерации данных.\n");

}

break;

}

case '5':

if (monitoringMode) {

int totalCount; // Массив запросов

int\* total = readFromFile(&totalCount); // Чтение данных из файла

if (total != NULL) {

RequestData requests = { .upCount = 0, .downCount = 0 }; // Инициализация структуры запросов

selectRequests(total, totalCount, &requests); // Распределение запросов

displayRequests(&requests); // Вывод запросов

free(total); // Освобождение памяти

}

}

else {

printf("Режим мониторинга выключен. Включите его для отображения запросов.\n");

}

break;

case '6':

if (monitoringMode) {

int totalCount; // Массив запросов

int\* total = readFromFile(&totalCount); // Чтение данных из файла

if (total != NULL) {

RequestData requests = { .upCount = 0, .downCount = 0 }; // Инициализация структуры запросов

selectRequests(total, totalCount, &requests); // Распределение запросов

sortRequests(requests.request\_up, requests.upCount, compareAscending); // Сортируем по возрастанию для вверх

sortRequests(requests.request\_down, requests.downCount, compareDescending); // Сортируем по убыванию для вниз

processRequests(requests.request\_up, requests.upCount, requests.request\_down, requests.downCount); // Обработка запросов

free(total); // Освобождение памяти

}

}

else {

printf("Режим мониторинга выключен. Включите его для обработки запросов лифта.\n");

}

break;

case '7':

if (monitoringMode) {

int totalCount; // Массив запросов

int\* total = readFromFile(&totalCount); // Чтение данных из файла

if (total != NULL) {

analyzeData(total, totalCount); // Анализ данных о вызовах лифта

free(total); // Освобождение памяти

}

}

else {

printf("Режим мониторинга выключен. Включите его для анализа данных.\n");

}

break;

case '8':

printf("Выход из программы.\n");

break;

default:

printf("Неверный выбор. Пожалуйста, попробуйте снова.\n");

}

} while (choice != '8');

return 0;

}

int readUserInput(ElevatorCall\* calls) {

int count = INITIAL\_COUNT; // Счетчик вызовов

int inputFloor; // Введённый этаж

printf("Введите этаж вызова (например, -2 для вызова вниз со второго этажа), или 0 для завершения ввода:\n");

while (count < MAX\_REQUESTS) { // Ограничение на количество вводимых запросов

printf("Этаж вызова: ");

scanf("%d", &inputFloor);

if (inputFloor == -1) { // Проверка на спуск с первого этажа

printf("С этажа %d можно вызвать лифт только вверх.\n", MIN\_FLOOR);

continue;

}

else if (inputFloor == MAX\_FLOOR) { // Проверка на подъём с десятого этажа

printf("С этажа %d можно вызвать лифт только вниз.\n", MAX\_FLOOR);

continue;

}

if (inputFloor == 0) { // Завершение ввода, если введено 0

printf("Ручной ввод данных завершён.\n");

break;

}

if (abs(inputFloor) < MIN\_FLOOR || abs(inputFloor) > MAX\_FLOOR) { // Проверка диапазона

printf("Ошибка: Этаж должен быть в диапазоне [%d, %d].\n", MIN\_FLOOR, MAX\_FLOOR);

continue;

}

calls[count].floor = abs(inputFloor); // Модуль числа определяет этаж

calls[count].direction = (inputFloor > 0) ? 1 : -1; // Определение направления по знаку числа

count++; // Увеличение счетчика вызовов

}

return count;

}

void displayInput(ElevatorCall\* calls, int count) {

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

if (calls[i].direction > 0) {

printf("Вызов с этажа %d. Направление: Вверх.\n", calls[i].floor);

}

else {

printf("Вызов с этажа %d. Направление: Вниз.\n", calls[i].floor);

}

}

return 0;

}

void writeToFile(ElevatorCall\* calls, int count) {

FILE\* file = fopen("test\_data.txt", "a");

if (file == NULL) {

return -1;

}

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

fprintf(file, "%d\n", (calls[i].direction > 0 ? calls[i].floor : -calls[i].floor));

}

fclose(file);

return 0;

}

void generateRandomCalls(ElevatorCall\* calls, int count) {

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

int floor = rand() % (MAX\_FLOOR - MIN\_FLOOR + 1) + MIN\_FLOOR; // Генерация этажа от MIN\_FLOOR до MAX\_FLOOR

int direction = (rand() % 2) == 0 ? 1 : -1; // Генерация направления случайным образом

calls[i].floor = floor; // Сохранение сгенерированныго этажа в массив вызовов

calls[i].direction = direction; // Сохранение сгенерированного направления в массив вызовов

}

return 0;

}

int\* readFromFile(int\* totalCount) {

FILE\* file = fopen("test\_data.txt", "r");

if (file == NULL) {

return -1;

}

int\* total = malloc(MAX\_REQUESTS \* sizeof(int)); // Динамическое выделение памяти под массив total

if (total == NULL) {

fclose(file);

return -1;

}

\*totalCount = INITIAL\_COUNT\_DATA; // Инициализация счетчика прочитанных данных

while (\*totalCount < MAX\_REQUESTS && fscanf(file, "%d", &total[\*totalCount]) == 1) { // Увеличение счетчика на единицу при успешном чтении числа из файла

(\*totalCount)++;

}

fclose(file);

return total;

}

void selectRequests(int\* total, int totalCount, RequestData\* requests) {

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

if (total[i] > 0) { // Проверка, является ли вызов вверх

requests->request\_up[requests->upCount++] = total[i]; // Добавляем вызов вверх в массив request\_up и увеличиваем счетчик

}

else { // В противном случае это вызов вниз

requests->request\_down[requests->downCount++] = -total[i]; // Добавляем модуль вызова вниз в массив request\_down и увеличиваем счетчик

}

}

}

void displayRequests(const RequestData\* requests) {

printf("\nЗапросы вверх:\n");

for (int i = 0; i < requests->upCount; i++) {

printf("%d ", requests->request\_up[i]);

}

printf("\nЗапросы вниз:\n");

for (int i = 0; i < requests->downCount; i++) {

printf("%d ", requests->request\_down[i]);

}

printf("\n");

return 0;

}

int compareAscending(int a, int b) {

return a - b; // Возвращает разницу между a и b

}

int compareDescending(int a, int b) {

return b - a; // Возвращает разницу между b и a

}

void sortRequests(int\* requests, int count, int (\*compare)(int a, int b)) {

for (int i = 0; i < count - 1; i++) {

for (int j = i + 1; j < count; j++) {

if (compare(requests[i], requests[j]) > 0) { // Использование функции сравнения для определения порядка

int temp = requests[i]; // Сохранение текущего элемента во временной переменной

requests[i] = requests[j]; // Замена местами элементов, если они не в правильном порядке

requests[j] = temp;

}

}

}

return 0;

}

void processRequests(int\* request\_up, int upCount, int\* request\_down, int downCount) {

int current\_floor = MIN\_FLOOR;

// Обработка запросов вверх

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

while (current\_floor < request\_up[i]) { // Подъём до запрашиваемого этажа

current\_floor++; // Увеличение текущего этажа

printf("Лифт поднимается на этаж %d ...\n", current\_floor);

}

if (current\_floor == request\_up[i]) { // Проверка с запрашиваемым этажом

printf("Лифт остановился на этаже %d\n", current\_floor);

}

}

// Проверка необходимости подняться до вызова "вниз"

int highest\_floor; // Этаж до которого нужно подяться

if (downCount > 0) { // Проверка наличия запросов вниз

highest\_floor = request\_down[0]; // Взятие самый верхнего этажа из запросов вниз

}

else {

highest\_floor = request\_up[upCount - 1]; // Взятие самого высокого этажа из запросов вверх

}

while (current\_floor < highest\_floor) { // Подъём до этажа выше текущего

current\_floor++;

printf("Лифт поднимается на этаж %d ...\n", current\_floor);

}

// Обработка запросов вниз

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

while (current\_floor > request\_down[i]) { // Спукс до запрашиваемого этажа

current\_floor--;

printf("Лифт спускается на этаж %d ...\n", current\_floor);

}

if (current\_floor == request\_down[i]) { // Проверка с запрашиваемым этажом

printf("Лифт остановился на этаже %d\n", current\_floor);

}

}

return 0;

}

void analyzeData(int\* total, int totalCount) {

if (total == NULL || totalCount <= 0) {

printf("Ошибка: Данные отсутствуют или введы неверно");

}

int upCalls = INITIAL\_COUNT\_UP; // Счетчик вызовов вверх

int downCalls = INITIAL\_COUNT\_DOWN; // Счетчик вызовов вниз

int callCounts[MAX\_FLOOR + 1] = { 0 }; // Массив для подсчета вызовов на каждом этаже

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

if (total[i] > 0) {

upCalls++;

}

else {

downCalls++;

}

callCounts[abs(total[i])]++; // Увеличение счетчика для соответствующего этажа

}

int mostFrequentFloor = MIN\_FLOOR; // Самый часто вызываемый этаж

for (int floor = MIN\_FLOOR; floor <= MAX\_FLOOR; floor++) {

if (callCounts[floor] > callCounts[mostFrequentFloor]) {

mostFrequentFloor = floor;

}

}

printf("\nАнализ данных:\n");

printf("Всего вызовов: %d\n", totalCount);

printf("Вызовы вверх: %d\n", upCalls);

printf("Вызовы вниз: %d\n", downCalls);

printf("Самый часто вызываемый этаж: %d\n", mostFrequentFloor);

return 0;

}