МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий
(название института полностью)

Кафедра/департамент «Информационные системы»
(наименование кафедры/департамента полностью)

(код и наименование направления подготовки/специальности)

(наименование профиля/специальности)

№2510
07.12.2023

КУРСОВАЯ РАБОТА / КУРСОВОЙ ПРОЕКТ

по дисциплине
Объектно-ориентированное программирование
(наименование дисциплины)

на тему **Обработка табличной информации (вариант № 4)**

Выполнил: обучающийся
группы: ИТ/б-22-5-о

Базарный А.Р
(инициалы, фамилия)

« 25 » 12 20 23 г.

Научный руководитель:
В. А. Петросов
(инициалы, фамилия)

« 25 » 12 20 23 г.

Оценка 85

« 25 » 12 20 23 г.

Севастополь
20 23

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий
(название института полностью)

Кафедра/ департамент «Информационные системы»
(наименование кафедры/департамента полностью)

(код и наименование направления подготовки/специальности)

(наименование профиля/специальности)

Курс 2 Группа ИТ/б-22-5-о Семестр третий

З А Д А Н И Е
НА КУРСОВОЙ ПРОЕКТ (РАБОТУ)

обучающегося Базарного Александра Руслановича
(фамилия, имя, отчество)

1. Тема работы (проекта) Обработка табличной информации
вариант 4

2. Срок сдачи обучающимся законченного проекта (работы)

3. Исходные/входные данные к проекту (работе) 1. Язык программирования — C/ C++
2. Операционная система — Windows
3. Среда программирования — Visual Studio Community Edition
4. Класс вычислительной машины — персональная настольная ЭВМ
5. Меню-ориентированный интерфейс
6. Исходные данные — файл с записями о воздушных рейсах
7. Выходные данные — файл с записями о проданных товарах ,сумма выручки

4. Содержание пояснительной записки (перечень вопросов, подлежащих разработке) техническое задание, аннотация, содержание, введение, назначение и область применения программы, технические характеристики программы, выполнение программы, выводы, перечень ссылок, приложение (текст программы).

5. Перечень графического материала (с точным указанием обязательных чертежей) презентация

6. Дата выдачи задания 01.09.2023

КАЛЕНДАРНЫЙ ПЛАН

№ п/п	Название этапов проекта (работы)	Срок выполнения этапов проекта (работы)	Примечание
1	Постановка задачи	01.09.23 — 08.09.23	2-ая неделя
2	Функции организации и просмотра списка	01.09.23 — 08.09.23	2-ая неделя
3	Функции добавления и удаления	08.09.23 — 22.09.23	2-4 неделя
4	Функции корректировки и сортировки	22.09.23 — 6.10.23	4-6 неделя
5	Функции сохранения списка (в текстовый и типизированный файл)	06.10.23 — 20.10.23	6-8 неделя
6	Функции чтения списка (из текстового и типизированного файлов)	20.10.23 — 3.11.23	8-10 неделя
7	Функции поиска и обработки	03.11.23 — 17.11.23	10-12 неделя
8	Разработка интерфейса	17.11.23 — 1.12.23	12-14 неделя
9	Тестирование и отладка	03.11.23 — 1.12.23	10-14 неделя
10	Разработка программных документов (ПЗ и презентации)	1.12.23 — 15.12.23	14-16 неделя
11	Защита	15.12.23 — 25.12.23	16-17 неделя

Обучающийся Базарный А.Г. _____
(фамилия и инициалы) *(подпись)*

Руководитель курсового проекта (работы)
Петраков В.А. ассистент _____
(фамилия, инициалы, должность) *(подпись)*

« 01 » 09 20 23 г.

АННОТАЦИЯ

Документ представляет анализ обработки табличной информации, сфокусированной на воздушных рейсах. Рассматриваются методы анализа и визуализации данных, связанных со статистикой авиарейсов. В работе освещаются технологии обработки больших объемов данных для оптимизации рейсов и улучшения эффективности авиакомпаний. Рассматриваются современные подходы к анализу данных о стоимости рейсов, количеству пассажиров, рассчитывается прибыль после проведённого авиарейса. Документ представляет актуальные методы и инструменты обработки табличной информации, способствующие оптимизации и улучшению управления воздушными перевозками. Исследование подчеркивает значимость визуального предоставления информации в авиационной отрасли для повышения надежности и упрощения выявления статистики авиаперелетов.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

VS – Visual Studio

ВВЕДЕНИЕ

Этот проект является ответом на возросшую потребность в эффективной обработке данных в авиационной отрасли. В современном мире организации, такие как авиакомпании, сталкиваются с огромными объемами данных, требующими систематизации, анализа и преобразования для принятия обоснованных решений. Целью данной программы является создание инструмента, способного обрабатывать табличную информацию, используемую в авиационной сфере, с высокой степенью точности, эффективности и удобства.

Данный проект направлен на создание инновационного инструмента, способного значительно улучшить процессы обработки данных в авиационной сфере, повысить оперативность принятия решений и обеспечить более эффективное функционирование авиакомпаний.

Целью курсовой работы является разработка программы, согласно документу – техническому заданию, выданному организацией “Севастопольский государственный университет” 01.09.2023. Техническое задание заключается в подсчёте средней стоимости перевозки одного пассажира на рейсе, а также итоговые данные по затратам и количеству пассажиров и среднюю стоимость провозки одного пассажира по аэропорту. Кроме того, для удобной работы пользователя требуется разработать меню-ориентированный интерфейс и в нём реализовать следующие функции:

Начальное создание таблицы

Организация таблицы

Просмотр таблицы

Добавление записи в начало/конец таблицы

Корректировка записи в таблице

Сортировка таблицы

Поиск записи в таблице

Сохранение таблицы в файле

Чтение данных из файла

Обработка таблицы и просмотр результатов обработки

Выход

В последующих разделах данной пояснительной записки предоставлено подробное описание разрабатываемой программы, процессов ее реализации, анализа требований её особенностей. Особое внимание уделено модульности программы. Каждая её функция, требуемая в меню, состоит из отдельного модуля.

НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ ПРОГРАММЫ

Программа для обработки табличных данных в авиакомпании разработана с целью эффективного управления и анализа разнообразной информации, связанной с деятельностью авиакомпании. Основной задачей программы является обеспечение анализа и вывода систематизированных табличных данных для получения статистики и работы с ней.

Характеристика области применения:

Программа предназначена для работы с разнообразными табличными данными, такими как номер рейса, марка самолёта, название самолёта, данные о количестве пассажиров, расходы на рейс, и другие аспекты, связанные с оперативной и административной деятельностью авиакомпании.

Управление рейсами и билетами: Программа позволяет отслеживать продажу билетов, а также анализировать данные о пассажирах, что способствует оптимизации заполненности рейсов и максимизации доходов.

Финансовый анализ: Обработка данных в области финансов позволяет авиакомпании проводить анализ доходов и расходов, эффективно управлять бюджетом и прогнозировать финансовые результаты.

Программа для обработки табличных данных в авиакомпании обладает высокой гибкостью и адаптируемостью, что позволяет ей быть эффективным инструментом в различных аспектах бизнеса авиакомпании, от ведения отчётности до стратегического управления.

1 ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ ПРОГРАММЫ

1.1 Постановка задачи

Даны сведения о перевозках авиапассажиров на рейсах одного аэропорта. Структура записи: номер рейса, название маршрута (например, «Севастополь — Лондон», 20 символов), марка самолета (6 символов), общие стоимостные затраты на рейс, количество пассажиров. Подсчитать среднюю стоимость перевозки одного пассажира на рейсе, а также итоговые данные по затратам и количеству пассажиров и среднюю стоимость провозки одного пассажира по аэропорту.

Реализовать меню-ориентированный интерфейс для списка/бинарного дерева. и написать следующие функции:

1) Начальное создание таблицы. При необходимости создания новой таблицы исходные данные считываются из текстового файла. Имя файла должен задавать пользователь.

2) Просмотр таблицы. При этом необходимо предусмотреть возможность скроллинга.

3) Добавление новой записи в таблицу. В начало/конец списка или в левое/правое поддерево бинарного дерева.

4) Удаление записи. Удаляемый элемент выбирается по одному из полей таблицы (ключевому). Ключевое поле выбирает студент. При удалении нужно переспросить пользователя программы уверен ли он в этом.

5) Корректировка записи в таблице. Корректируемую запись выбирают по одному из полей таблицы (ключевому).

6) Сортировка таблицы. Сортировка производится по одному из полей таблицы (ключевому). Метод сортировки выбирает студент.

7) Поиск записи в таблице (по не ключевому полю).

8) Сохранение таблицы в файле. Имя файла должен вводить пользователь. Сохранять таблицу следует в текстовый файл и в типизированный.

9) Чтение данных из файла. Имя файла должен вводить пользователь. При чтении данных необходимо создавать новый список.

10) Обработка таблицы и просмотр результатов обработки. Результат обработки необходимо вывести на экран и в текстовый файл. Имя файла вводит пользователь.

11) Выход — завершение работы программы.

1.2 Применяемые математические методы

Для расчёта средней стоимости перевозки одного пассажира на рейсе была использована следующая формула:

$$x = \frac{y}{z} \quad (1.1)$$

где x - средняя стоимость перевозки одного пассажира на рейсе;

y - общие стоимостные затраты на рейс;

z – количество пассажиров на рейсе.

Для расчёта итоговых данных по затратам была использована следующая формула:

$$So = S_1 + S_2 ... + S_n \quad (1.2)$$

где S_o – итоговые данные по затратам;

S_1, S_2 – первые элементы таблицы по затратам;

S_n – последний элемент таблицы по затратам.

Для расчёта итоговых данных по количеству пассажиров была использована следующая формула:

$$Ko = K_1 + K_2 ... + K_n \quad (1.3)$$

где K_o – итоговые данные по затратам;

K_1, K_2 – первые элементы таблицы по количеству пассажиров;

K_n – последний элемент таблицы по количеству пассажиров.

Для расчёта итоговой средней стоимости провозки для провозки одного пассажира по аэропорту была использована следующая формула:

$$\frac{X1 + X2 \ldots + Xn}{n} \quad (1.4)$$

где X1, X2 – первые элементы таблицы по средней стоимости перевозки пассажиров на рейсе, полученные по формуле (1.1)

Xn – последний элемент таблицы по средней стоимости перевозки пассажиров на рейсе, полученный по формуле (1.1)

n – количество элементов в таблице

1.3 Описание и обоснование выбора метода организации входных, выходных и промежуточных данных

Для организации данных был выбран двунаправленный линейный список. В отличие от однонаправленного линейного списка, содержащего только указатель на следующий элемент двунаправленный облегчает работу программиста и ускоряет работу программы в некоторых функция, но взамен требует больше памяти для хранения указателя на предыдущий элемент.

Альтернативой для организации рассматривалось бинарное дерево. Дерево содержит также 2 указателя. На левый и правый элемент, но не содержит указатель на предыдущий, что в некоторых случаях усложняет написание функций. По сравнению с двунаправленным линейным списком большинство функций использует рекурсию, из-за чего программа выполняется дольше. Можно создать и функции, не использующие рекурсию, но это усложнит работу программисту

На основании выводов о памяти/скорости работы/удобства разработки можно составить таблицу преимуществ и недостатков методов организации данных. Для оценки методов можно использовать следующую шкалу оценки от 1 до 3

Таблица 1

Метод	Память	Скорость работы	Удобство разработки

Однонаправленный линейный список	3	2	2
Двунаправленный линейный список	2	3	3
Бинарное дерево	2	1	1

Подводя итоги, можно сказать, что из-за ряда преимуществ был выбран двунаправленный линейный список. Несмотря на то, что программа жертвует частью памяти на хранение указателя на следующий элемент она работает быстрее в некоторых функциях и их удобней разрабатывать, имея в памяти указатель на предыдущий элемент. Примером данной функции может послужить вывод элементов предыдущей страницы на экран. Вместо того, чтобы идти к выбранному элементу с начала функции можно обратиться к предыдущим элементам, ускорив работу программы.

1.4 Обоснование выбора языка и среды программирования

Для разработки программы был использован язык C++ и среда программирования VS Community Edition.

У языка C++ есть свои плюсы, благодаря чему он и был выбран:

Высокая производительность, потому что C++ не накладывает никакой избыточной нагрузки на программу, не использующую какие-либо возможности.

C++ — это мощный язык программирования общего назначения, который часто используется как системный. Он поддерживает не только объектно-ориентированное программирование, но и другие парадигмы. Этот язык может обращаться к низкоуровневым функциям и работать напрямую с системой — это важно, например, для оптимизации под определенную платформу или для расчетов графики

Также у языка C++ есть свои преимущества по сравнению с другими языками программирования. Например, в сравнении с языком C у C++ есть родная поддержка ООП.

Стабильность. У языка большое сообщество разработчиков и он сохраняет свою актуальность, находясь на третьем месте по популярности на момент 2023г.

Среда разработки VS Community Edition была выбрана по следующим причинам:

Основной причиной выбора среды VS Community Edition для создания данного программного продукта была исключительно высокая простота и скорость разработки прикладных решений. Продукты Microsoft для разработчиков давно входят в список наиболее востребованного программного обеспечения для программистов разного уровня.

VS Community Edition поддерживает работу с множеством языков, что помогает разработчикам переходить с 1 языка на другой.

Быстрая работа обеспечивается автодополнением при написании кода.

Есть возможность сменить компилятор на любой поддерживаемый, вплоть до пользовательского, что позволяет улучшить работу с программой.

1.5 Разработка модульной структуры программы

Модульность в любой программе облегчает восприятие кода и работу с ним. В случае необходимости изменить код для его оптимизации достаточно изменить какой-либо модуль программы.

В данной программе все методы были разбиты на модули:
AddElement.h и AddElement.cpp – отвечают за добавление элемента в начало, конец списка и добавление элемента в начало списка, в котором уже имеются данные для добавления. В cpp файле используются: `#include "List.h"`

```
#include "FillDatas.h"
```

CheckIndex.h и CheckIndex.cpp – отвечают за проверку входного индекса элемента. Возвращает true/false.

Move.h и Move.cpp – отвечают за возврат структуры под индексом. В cpp файле используются: `#include "List.h" #include "CheckIndex.h"`

Correct.h и Correct.cpp – отвечают за изменение существующего элемента. В cpp файле используются: `#include "List.h" #include "Move.h" #include "FillDatas.h"`

Delete.h и Delete.cpp – отвечают за удаление структуры под индексом. В cpp файле используются: `#include "List.h" #include "CheckIndex.h" #include "Move.h"`

FillDatas.h и FillDatas.cpp – отвечают за заполнение структуры данными. В cpp файле используется: `#include "List.h"`

Found.h и Found.cpp – отвечают за поиск данных по ключу. В cpp файле используются: `#include "List.h" #include "AddElement.h" #include "Show.h"`

List.h и List.cpp – отвечают за 3 используемые структуры и метод-меню программы. В cpp файле используются: `#include "List.h" #include "CheckIndex.h" #include "AddElement.h" #include "Show.h" #include "Correct.h" #include "Delete.h" #include "Sort.h" #include "Found.h" #include "Organize.h" #include "Save.h" #include "Load.h"`

Load.h и Load.cpp – отвечают за загрузку списка из файла. В cpp файле используются: `#include "List.h" #include "Delete.h" #include "AddElement.h" #include "Sort.h"`

Organize.h и Organize.cpp – отвечают за организацию списка. В cpp файле используется: `#include "AddElement.h"`

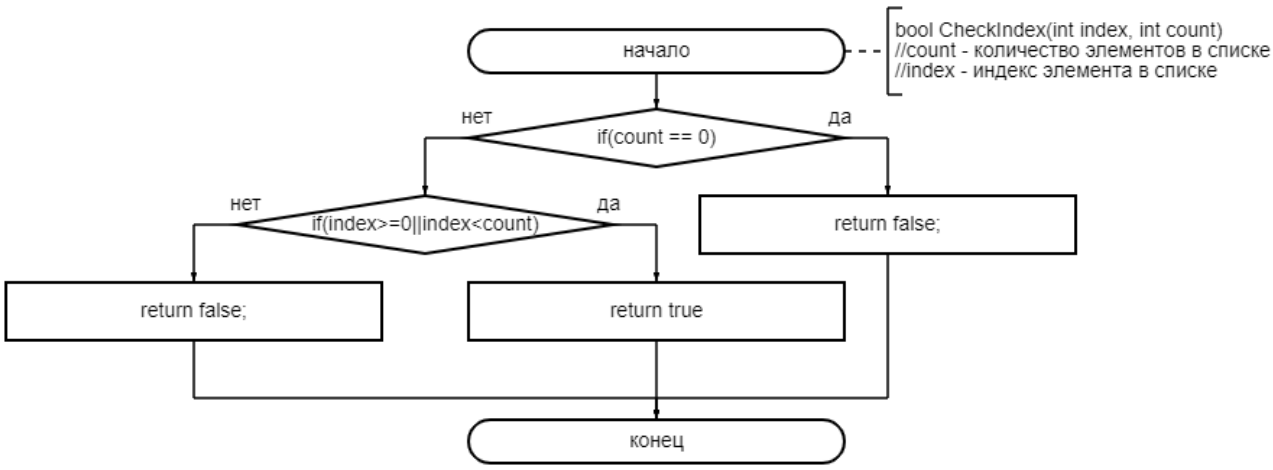
Reverse.h и Reverse.cpp – отвечают за переворот списка. В cpp файле используются: `#include "List.h" #include "AddElement.h"`

Save.h и Save.cpp – отвечают за сохранение списка в файл. В cpp файле используется: `#include "List.h"`

Show.h и Show.cpp – отвечают за вывод списка по страницам, а также за вывод требуемых итоговых данных. В cpp файле используется: `#include "List.h"`

Sort.h и Sort.cpp – отвечают за сортировку по любому из ключей. В cpp файле используются: `#include "List.h"` `#include "Reverse.h"`

1.6 Описание алгоритмов функционирования программы



Проверка на корректность индекса. Функция выдаёт true или false в зависимости от того существует ли такое количество count, к которому обращается индекс

Начало:

Проверка, если количество элементов в списке `count` равно 0. Если да, возвращение `false`.

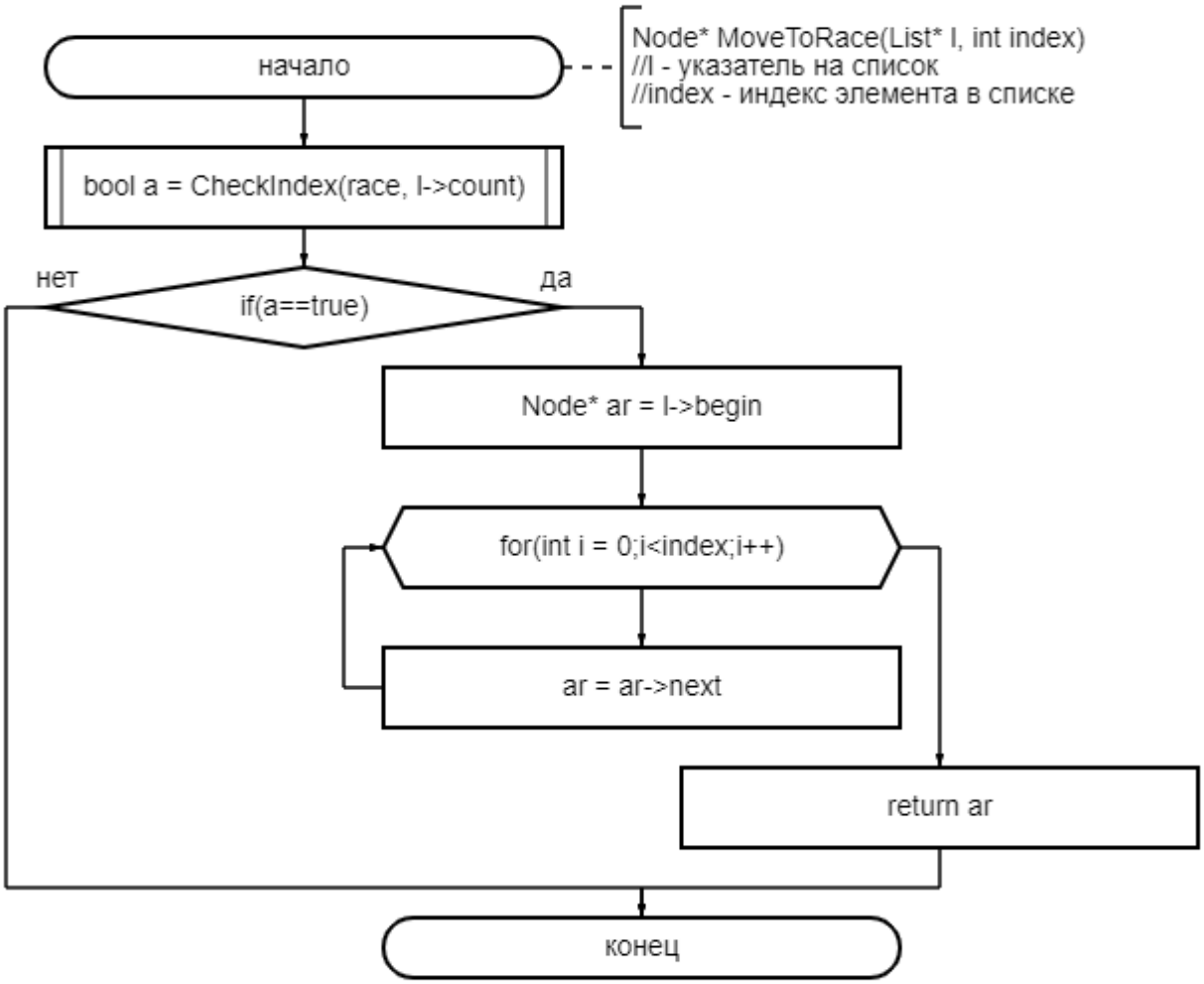
Проверка валидности индекса:

Проверка условия: если `index` больше либо равно 0 и `index` меньше `count`, возврат `true`.

В противном случае, возврат `false`.

Конец:

Завершение функции.



Перемещение к элементу по индексу. Функция проверяет существует ли данный индекс и возвращает элемент под этим индексом
Начало:

Проверка валидности индекса `index` с использованием функции `CheckIndex`. Если индекс невалиден, завершение функции.

Инициализация указателя:

Инициализация указателя `ar` значением `l->begin`.

Цикл перемещения к указанному индексу:

Начало цикла `for` с условием `i < index`.

Внутри цикла, обновление указателя `ar` на следующий узел `ar->next`.

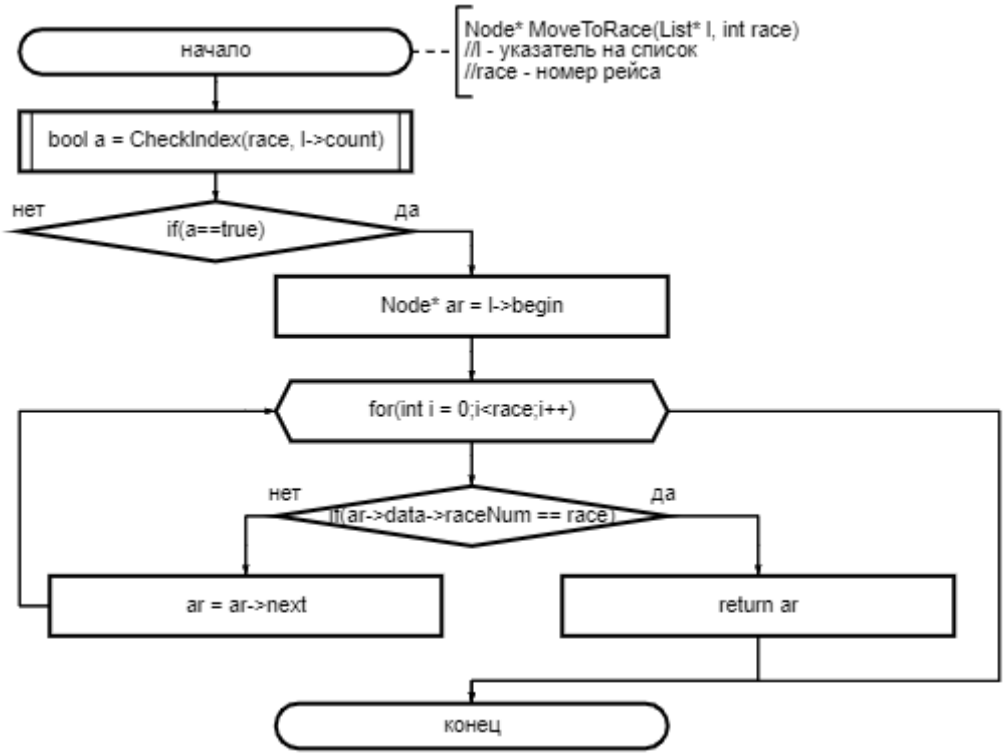
Увеличение значения переменной `i` на 1.

Возвращение указателя:

Возвращение указателя `ar`.

Конец:

Завершение функции.



Перемещение к элементу по номеру рейса. Функция проверяет существует ли данный номер рейса и если да, то возвращает элемент под этим номером. Номер рейса – уникальный ключ

Начало:

Проверка валидности номера гонки `race` с использованием функции `CheckIndex`. Если номер гонки невалиден, завершение функции.

Инициализация указателя:

Инициализация указателя `ar` значением `l->begin`.

Цикл поиска узла с заданным номером гонки:

Начало цикла `for` с условием `i < l->count`.

Внутри цикла, проверка, если номер гонки текущего узла `ar->data->raceNum` равен заданному `race`.

Если условие выполняется, возврат указателя `ar`.

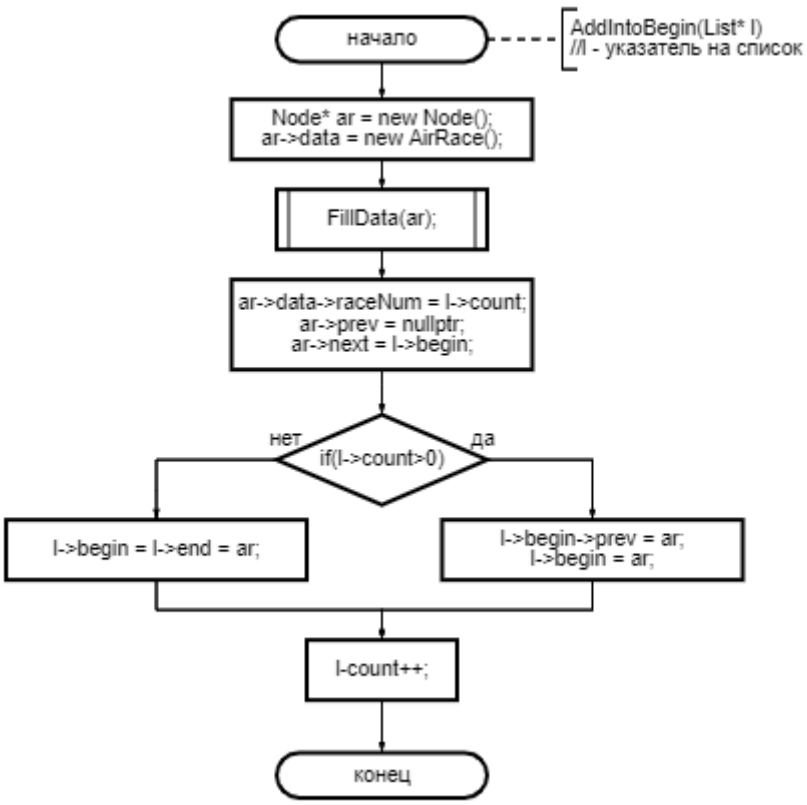
Увеличение значения переменной `i` на 1 и обновление указателя `ar` на следующий узел `ar->next`.

Возврат `NULL` в случае неудачи:

Если не найден узел с заданным номером гонки, возврат `NULL`.

Конец:

Завершение функции.



Начало:

Создание указателя `ar` на новый узел `Node`.

Выделение памяти для объекта `AirRace` и присвоение его указателю `ar->data`.

Заполнение данных объекта `AirRace` с помощью функции `FillData`.

Установка номера воздушного рейса:

Присвоение полю `raceNum` объекта `AirRace` значение `l->count`.

Установка указателей `prev` и `next`:

Установка `ar->prev` в `nullptr`.

Установка `ar->next` в текущий `l->begin`.

Обновление начала списка:

Проверка, если `l->count > 0`.

Если условие выполняется:

Установка `ar` в предыдущий узел для текущего `l->begin` (`l->begin->prev = ar`).

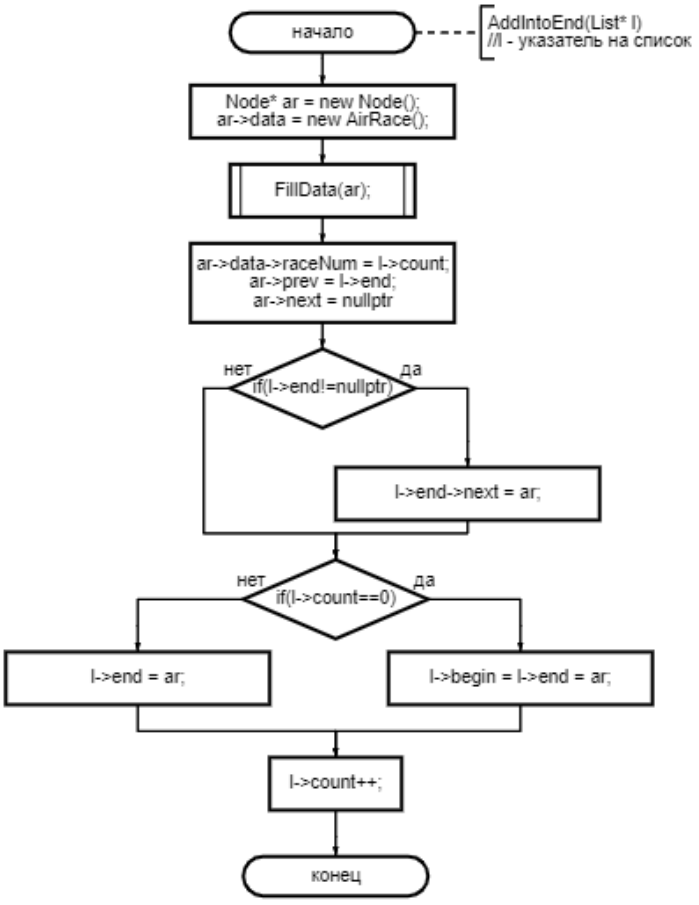
Обновление `l->begin` на новый узел `ar`.

В противном случае:

Установка `l->begin` и `l->end` в `ar`, так как это первый элемент списка.

Увеличение счетчика:

Увеличение `l->count` на единицу.



Начало:

Создание указателя `ar` на новый узел `Node`.

Выделение памяти для объекта "воздушный рейс" и присвоение его указателю `ar->data`.

Установка `ar->next` в `nullptr`.

Установка `ar->prev` в текущий `l->end`.

Заполнение данных объекта "воздушный рейс" с помощью функции `FillData`.

Установка номера воздушного рейса:

Присвоение полю `flightNum` объекта "воздушный рейс" значение `l->count`.

Обновление указателей:

Если `l->end` не является `nullptr`, то установка `l->end->next` в `ar`.

Обновление начала и конца списка:

Проверка, если `l->count == 0`.

Если условие выполняется:

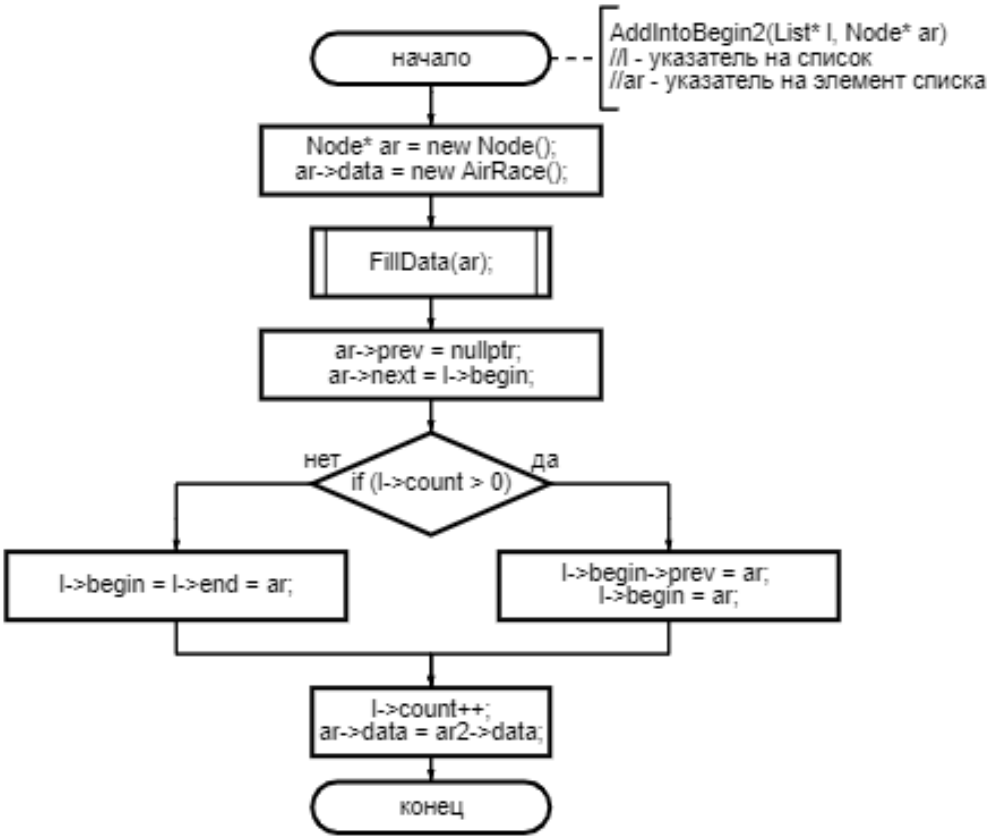
Установка `l->begin` и `l->end` в `ar`, так как это первый элемент списка.

В противном случае:

Обновление `l->end` на новый узел `ar`.

Увеличение счетчика:

Увеличение `l->count` на единицу.



Создание указателя `ar` на новый узел `Node`.

Выделение памяти для объекта "воздушный рейс" и присвоение его указателю `ar->data`.

Установка `ar->prev` в `nullptr`.

Установка `ar->next` в текущий `l->begin`.

Обновление начала списка:

Проверка, если `l->count > 0`.

Если условие выполняется:

Установка `ar` в предыдущий узел для текущего `l->begin` (`l->begin->prev = ar`).

Обновление `l->begin` на новый узел `ar`.

В противном случае:

Установка `l->begin` и `l->end` в `ar`, так как это первый элемент списка.

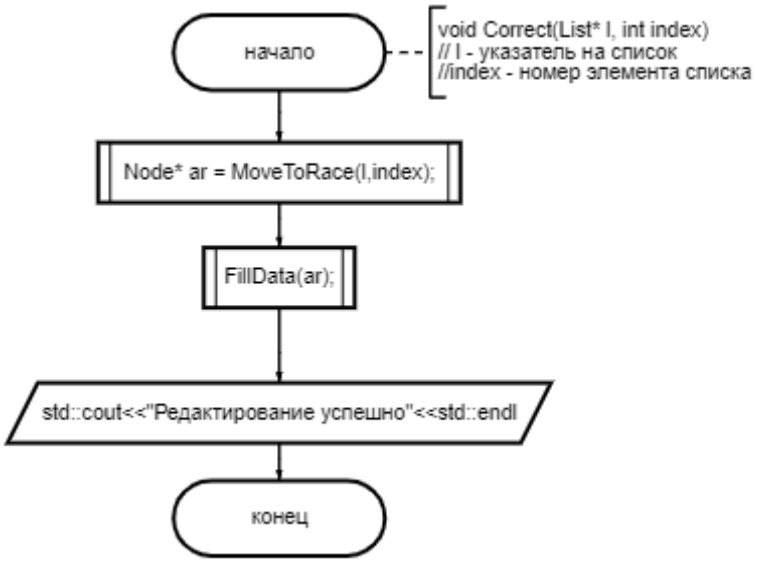
Увеличение счетчика:

Увеличение `l->count` на единицу.

Копирование данных из другого узла:

Присвоение `ar->data` данным из узла `ar2->data`.

В этом коде происходит добавление нового узла в начало списка и копирование данных из переданного узла `ar2`.



Корректировка элемента списка под индексом

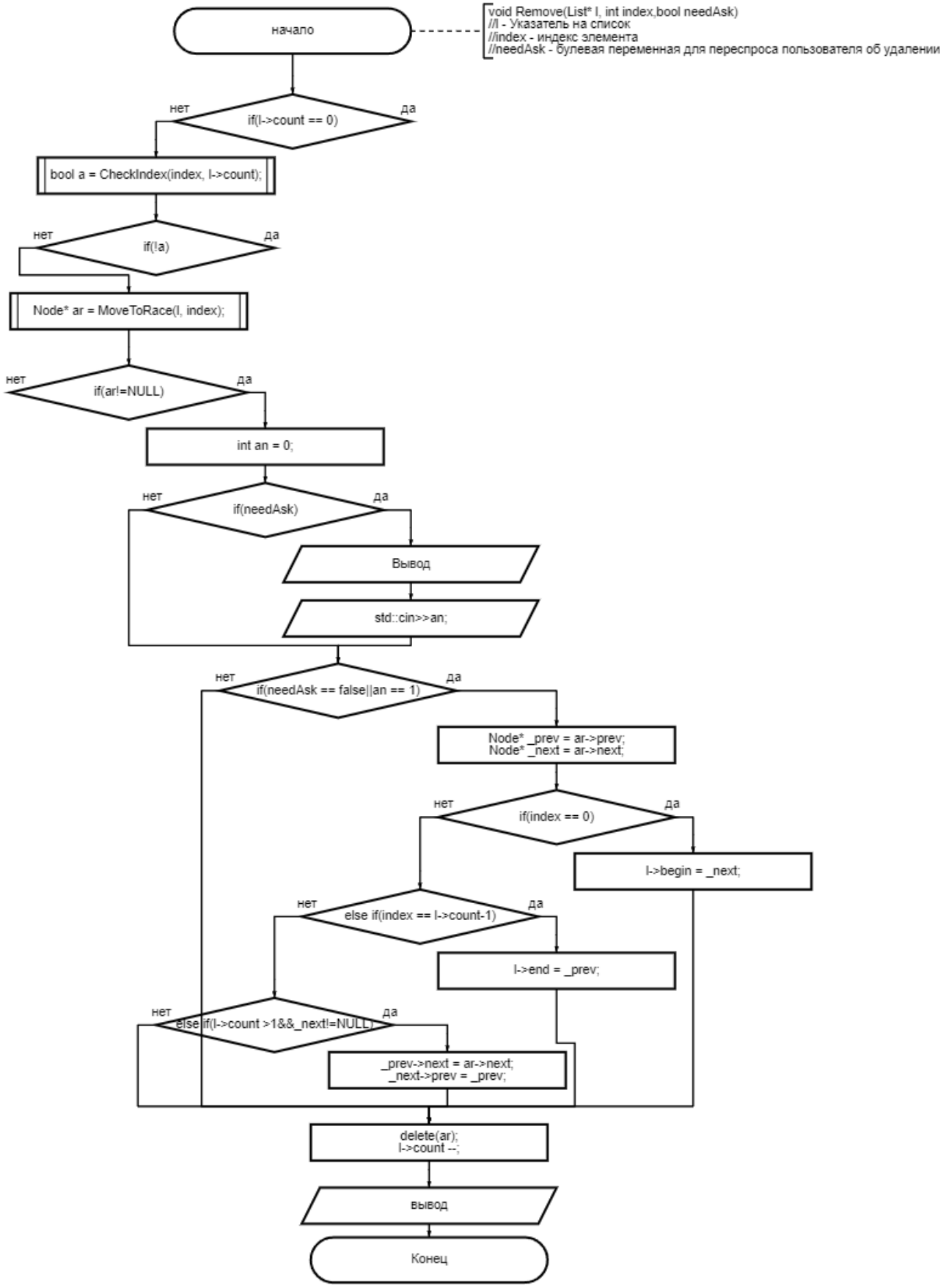
Перемещение к указанному индексу с использованием функции `MoveToRace` для получения указателя на узел `ar`.

Заполнение данных:

Вызов функции `FillData` для заполнения данных узла `ar`.

Вывод сообщения:

Вывод сообщения об успешной корректировке.



Удаление элемента под индексом

Начало:

Проверка, если количество элементов в списке `l->count` равно 0. Если да, завершение функции.

Проверка валидности индекса `index` с использованием функции `CheckIndex`. Если индекс невалиден, завершение функции.

Перемещение к указанному индексу с использованием функции `MoveToRace` для получения указателя на узел `ar`.

Подтверждение удаления (если нужно):

Если `needAsk` равно `true`, вывод сообщения с вопросом о подтверждении удаления записи.

Ввод ответа пользователя в переменную `an`.

Удаление записи:

Если ответ пользователя равен 1 (или если `needAsk` равно `false`), то продолжение удаления.

Получение указателей `_prev` и `_next` на предыдущий и следующий узлы от ar.

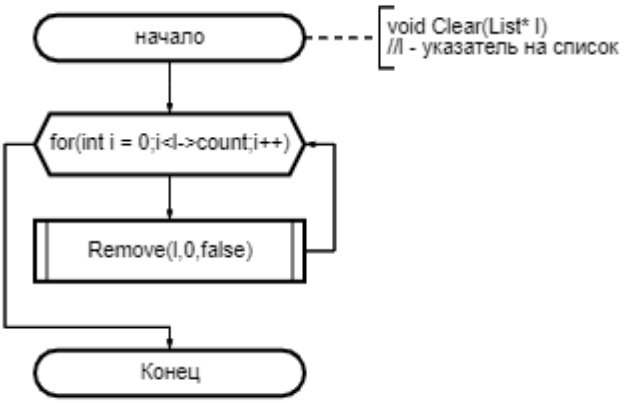
Обновление указателей `l->begin` и `l->end` в случае, если удаляемый узел является первым или последним.

Обновление указателей предыдущего и следующего узлов для обхода удаляемого узла.

Удаление узла `ar`.

Уменьшение счетчика `l->count`.

Вывод сообщения об успешном удалении.



Начало:

Инициализация переменной цикла `i` равной 0.

Цикл удаления:

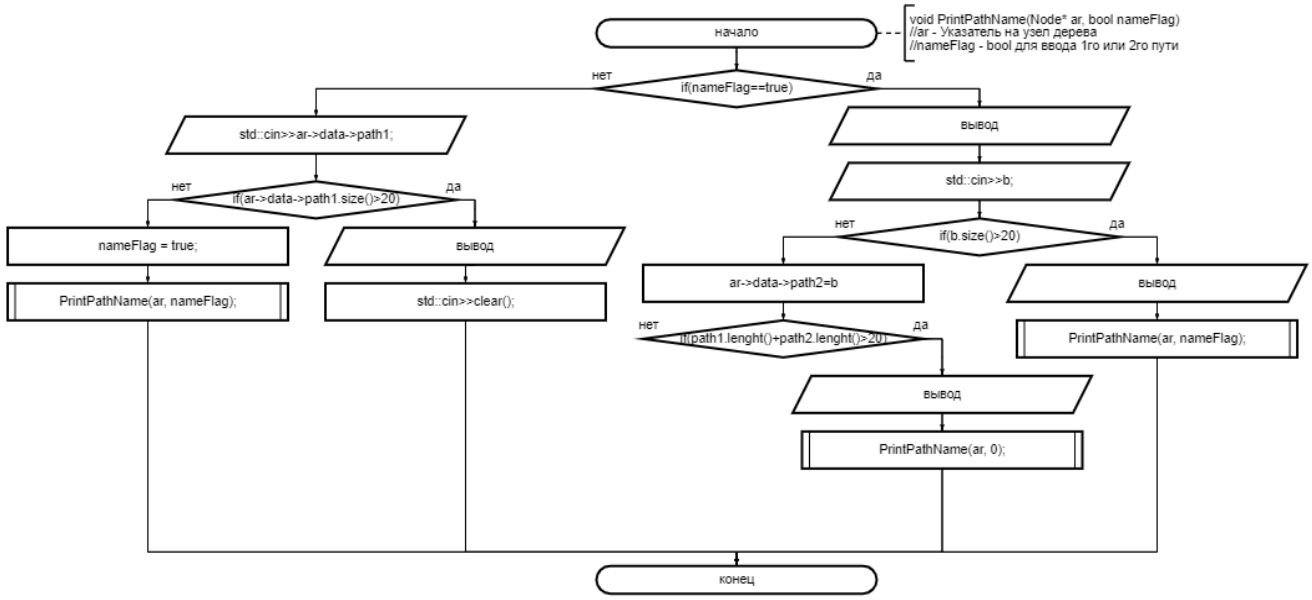
Начало цикла `for` с условием `i < l->count`.

Вызов функции `Remove` для удаления элемента с индексом 0, без запроса подтверждения (`false`).

Увеличение значения переменной `i` на 1.

Конец:

Завершение функции.



Начало:

Проверка значения флага `nameFlag`. Если `true`, вывод сообщения о вводе второго города (не более 20 символов).

Ввод и обработка данных:

Использование блока `try-catch` для обработки возможных исключений при вводе данных.

Ввод строки `b`.

Проверка длины строки `b`. Если длина больше 20 символов, вывод сообщения и повторный ввод.

Присвоение введенной строки `b` второму городу `ar->data->path2`.

Проверка общей длины строк `ar->data->path1` и `ar->data->path2`. Если сумма длин больше 20 символов, вывод сообщения и повторный ввод.

Обработка ошибок:

В блоке `catch`, вывод сообщения об ошибке, очистка входного буфера и повторный вызов функции `PrintPathName` с флагом `nameFlag`.

Ветвление в зависимости от nameFlag:

Если `nameFlag` равно `true`, возвращение к вводу первого города.

Если `nameFlag` равно `false`, вывод сообщения о вводе первого города (не более 20 символов).

Ввод и обработка данных для первого города:

Ввод строки `ar->data->path1`.

Проверка длины строки. Если длина больше 20 символов, вывод сообщения и очистка входного буфера.

Установка `nameFlag` в `true`.

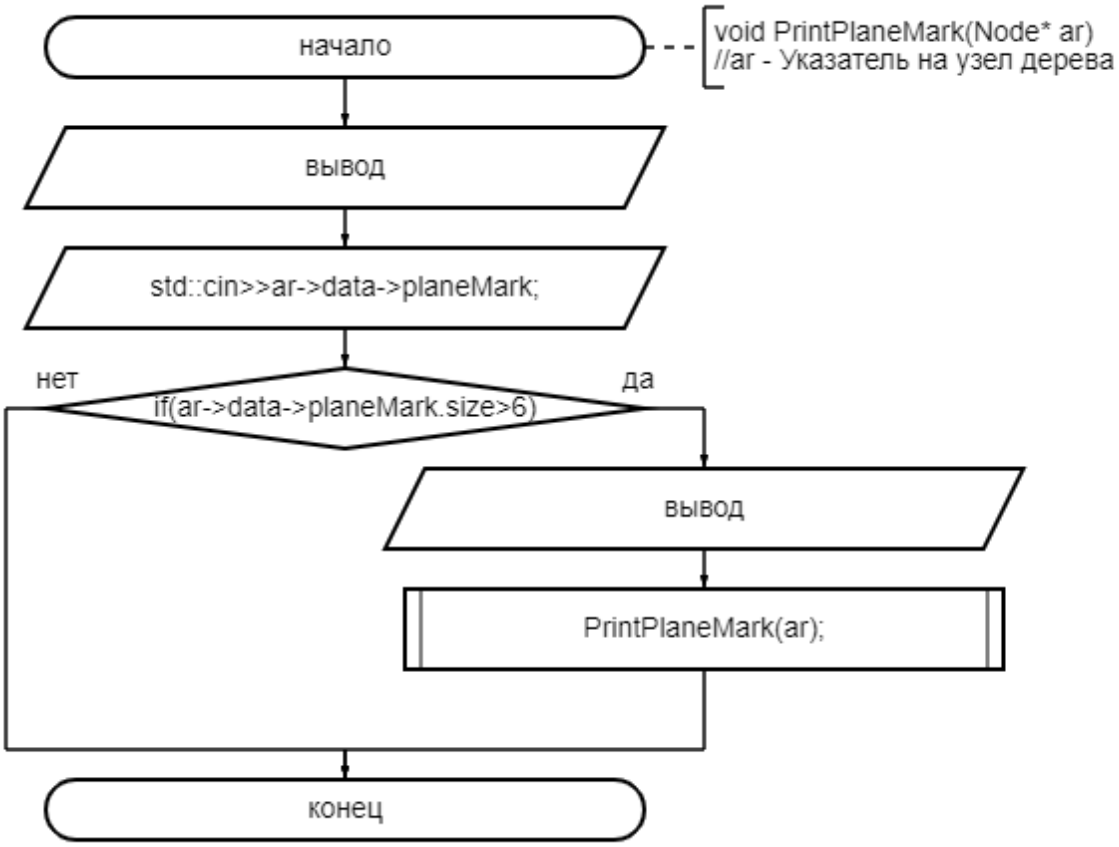
Рекурсивный вызов функции `PrintPathName` с флагом `nameFlag`.

Обработка ошибок для первого города:

В блоке `catch`, вывод сообщения об ошибке, очистка входного буфера и рекурсивный вызов функции `PrintPathName` с флагом `nameFlag`.

Конец:

Завершение функции.



Начало:

Вывод сообщения о вводе марки самолета (не более 6 символов).

Ввод и обработка данных:

Использование блока `try-catch` для обработки возможных исключений при вводе данных.

Ввод строки `ar->data->planeMark`.

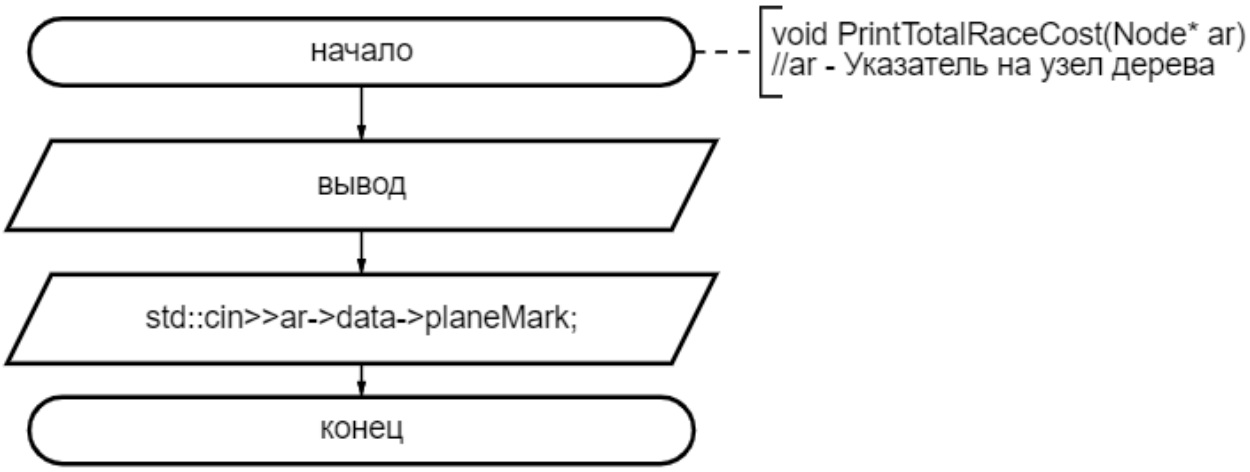
Проверка длины строки. Если длина больше 6 символов, вывод сообщения и повторный ввод.

Обработка ошибок:

В блоке `catch`, вывод сообщения об ошибке, очистка входного буфера и рекурсивный вызов функции `PrintPlaneMark`.

Конец:

Завершение функции.



Начало:

Вывод сообщения о вводе общей стоимости воздушного рейса.

Ввод и обработка данных:

Использование блока `try-catch` для обработки возможных исключений при вводе данных.

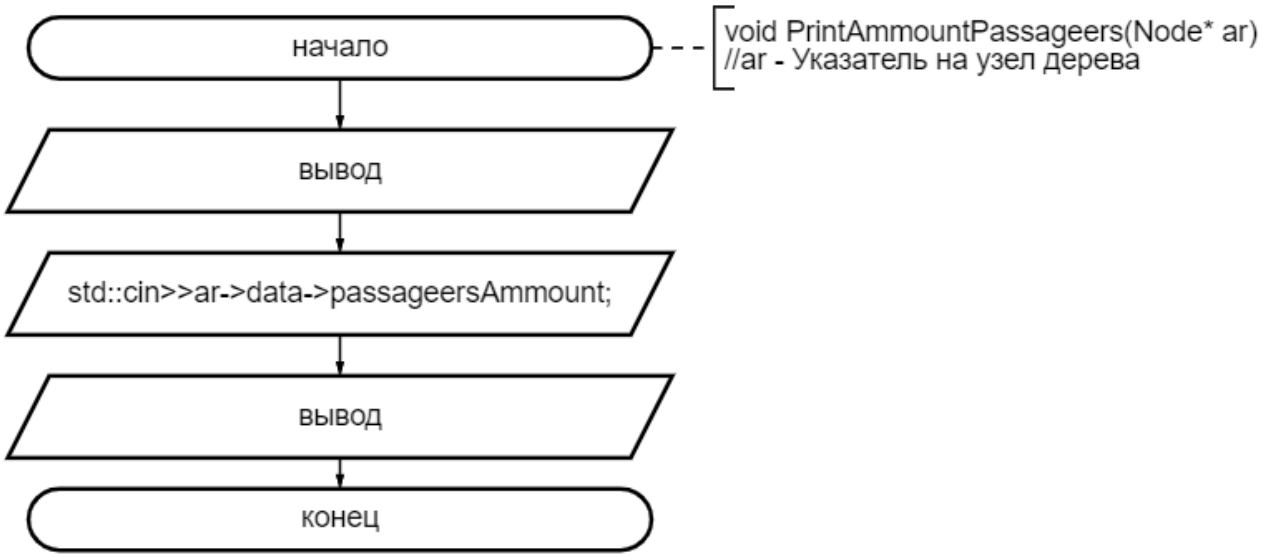
Ввод значения `ar->data->totalRaceCost`.

Обработка ошибок:

В блоке `catch`, вывод сообщения об ошибке, очистка входного буфера и рекурсивный вызов функции `PrintTotalRaceCost`.

Конец:

Завершение функции.



Начало:

Вывод сообщения о вводе количества пассажиров.

Ввод и обработка данных:

Использование блока `try-catch` для обработки возможных исключений при вводе данных.

Ввод значения `ar->data->passageersAmmount`.

Обработка ошибок:

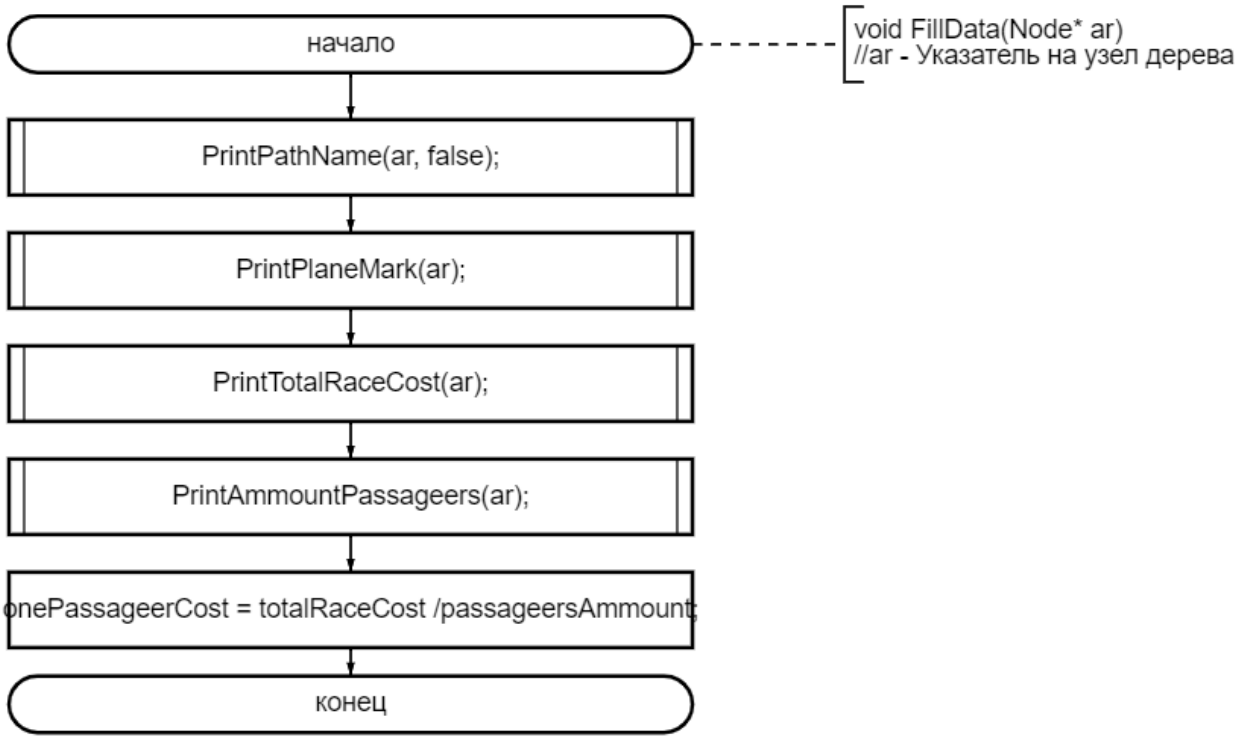
В блоке `catch`, вывод сообщения об ошибке, очистка входного буфера и рекурсивный вызов функции `PrintAmmountPassageers`.

Вывод сообщения об успешном создании:

Вывод сообщения "Created successful".

Конец:

Завершение функции.



Начало:

Вызов функции `PrintPathName` с флагом `false`.

Вызов функции `PrintPlaneMark`.

Вызов функции `PrintTotalRaceCost`.

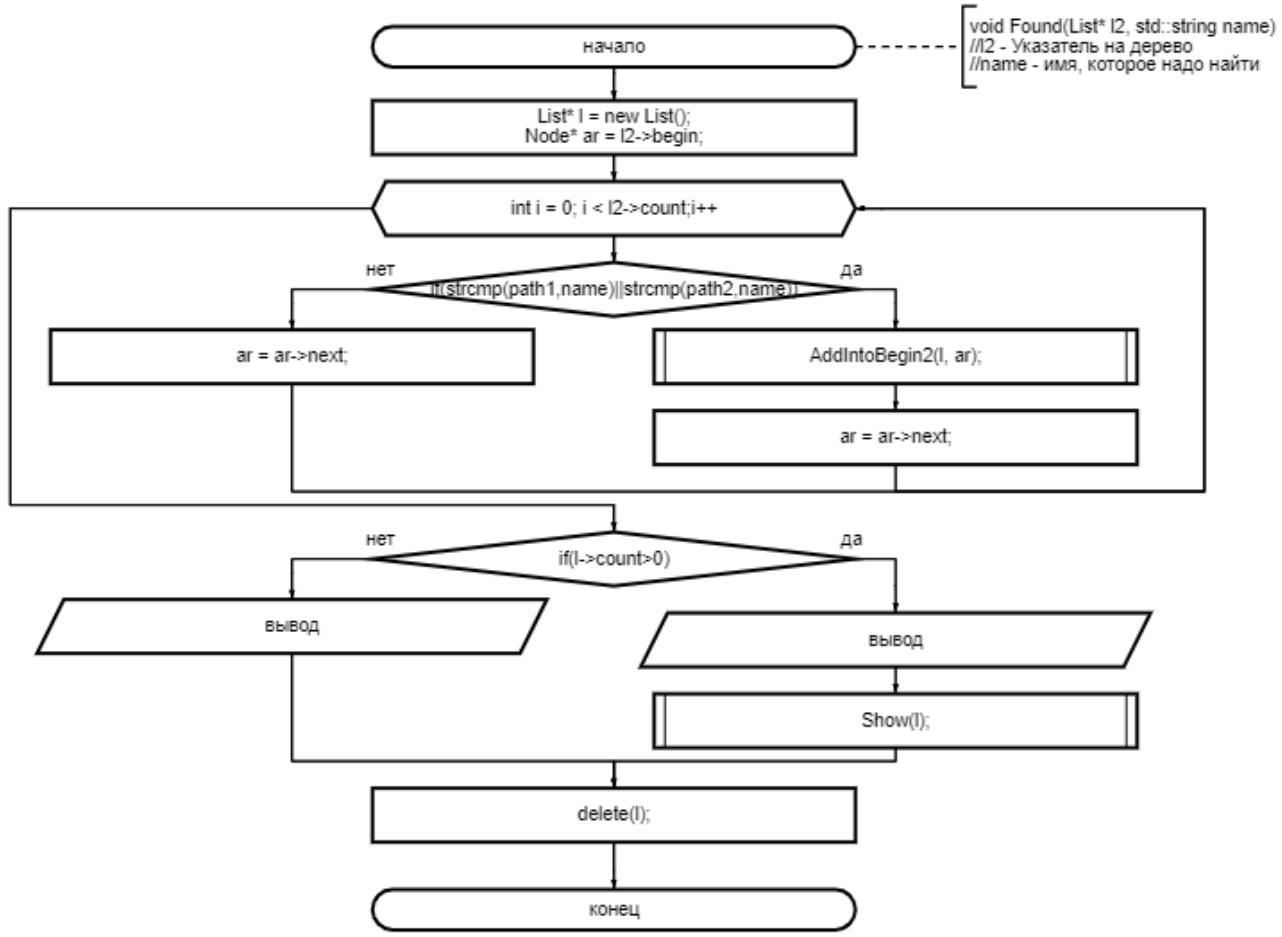
Вызов функции `PrintAmmountPassageers`.

Расчет стоимости для одного пассажира:

Расчет значения `ar->data->onePassageerCost` как отношение общей стоимости к количеству пассажиров.

Конец:

Завершение функции.



Создание временного списка `l`:

Создание нового списка `l`.

Инициализация указателя `ar`:

Инициализация указателя `ar` значением `l2->begin`.

Цикл поиска записей с соответствующим именем:

Начало цикла `for` с условием `i < l2->count`.

Проверка, если имя первого города или второго города текущего узла `ar` совпадает с заданным именем.

Если условие выполняется, вызов функции `AddIntoBegin2` для добавления узла в начало временного списка `l`.

Обновление указателя `ar` на следующий узел `ar->next`.

Увеличение значения переменной `i` на 1.

Вывод результатов:

Проверка, если количество записей во временном списке `l->count` больше 0.

Если условие выполняется, вывод сообщения "Записи с соответствующим именем:" и вызов функции `Show` для отображения записей из временного списка `l`.

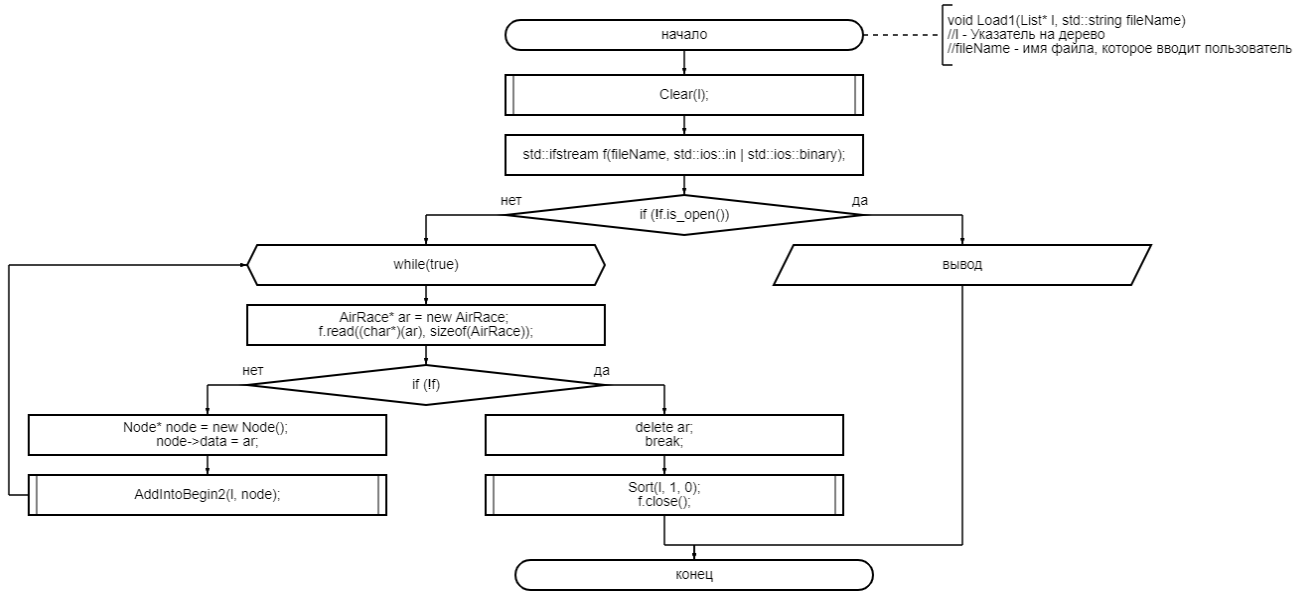
В противном случае, вывод сообщения "Записей с таким названием не найдено".

Освобождение памяти:

Удаление временного списка `l`.

Конец:

Завершение функции.



Очистка списка:

Вызов функции `Clear` для очистки списка l.

Открытие файла:

Попытка открытия файла с использованием `std::ifstream` и режимом `std::ios::in | std::ios::binary`.

Проверка успешности открытия файла:

Проверка, если файл успешно открыт. Если нет, вывод сообщения об ошибке и завершение функции.

Чтение данных из файла:

Начало бесконечного цикла.

Создание указателя `ar` на объект `AirRace`.

Чтение данных из файла в объект `ar` с использованием `f.read`.

Проверка успешности чтения. Если чтение не удалось, удаление объекта `ar` и завершение цикла.

Создание узла и добавление в список:

Создание нового узла `node`.

Присвоение указателю `node->data` объект `ar`.

Вызов функции `AddIntoBegin2` для добавления узла в начало списка `l`.

Сортировка списка:

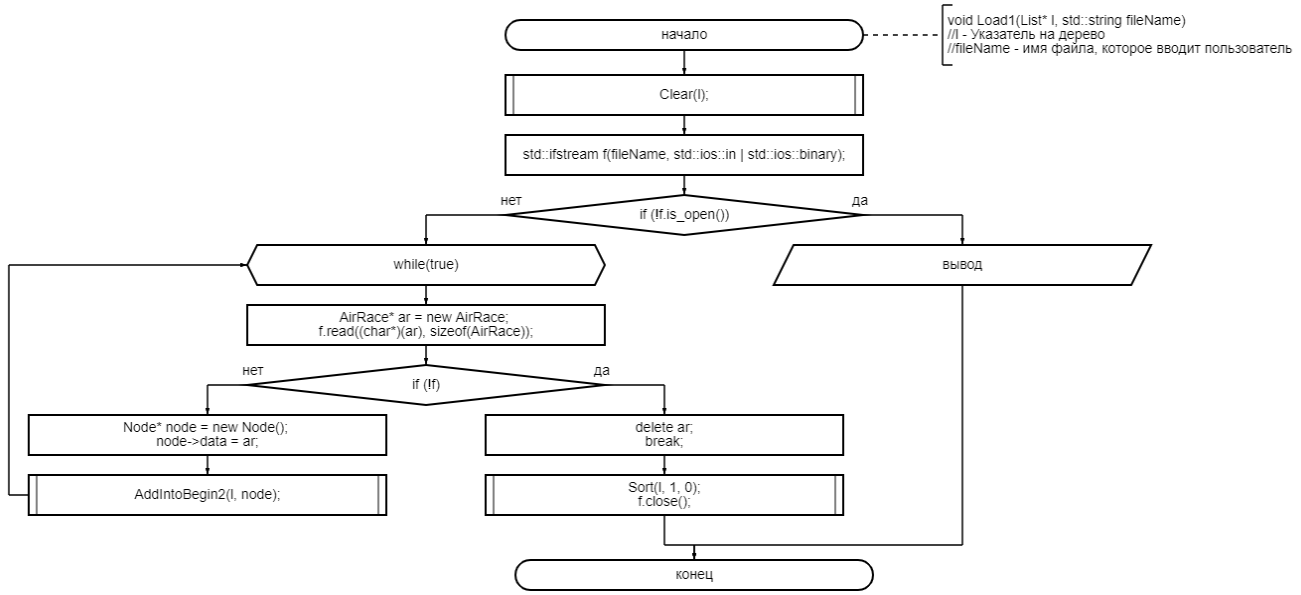
Вызов функции `Sort` для сортировки списка l по заданным параметрам.

Закрытие файла:

Закрытие файла с использованием `f.close()`.

Конец:

Завершение функции.



Открытие файла:

Попытка открытия файла с использованием `std::ifstream`.

Проверка успешности открытия файла:

Проверка, если файл успешно открыт. Если нет, вывод сообщения об ошибке и завершение функции.

Очистка списка:

Вызов функции `Clear` для очистки списка `l`.

Чтение данных из файла:

Начало цикла до достижения конца файла `(!f.eof())`.

Чтение данных из файла в соответствующие переменные: `path1`, `path2`, `planeMark`, `onePassageerCost`, `raceNum`, `totalRaceCost`, `passageersAmmount`.

Проверка успешности чтения. Если чтение не удалось (по достижении конца файла), завершение цикла.

Создание узла и добавление в список:

Создание нового узла `ar`.

Создание объекта `AirRace` и заполнение его данными из файловых переменных.

Присвоение указателю `ar->data` созданный объект `AirRace`.

Вызов функции `AddIntoBegin2` для добавления узла в начало списка l.

Сортировка списка:

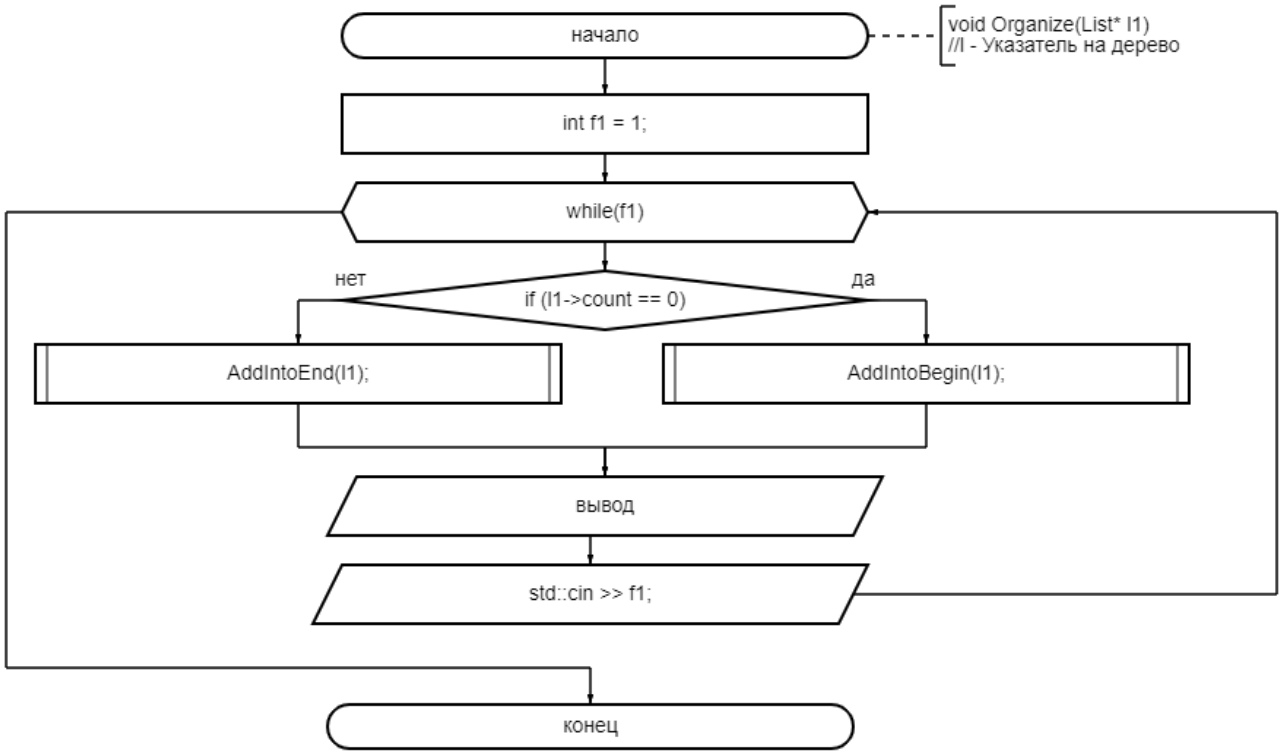
Вызов функции `Sort` для сортировки списка `l` по заданным параметрам.

Закрытие файла:

Закрытие файла с использованием `f.close()`.

Конец:

Завершение функции.



Инициализация переменной `f1`:

Инициализация переменной `f1` значением 0.

Начало цикла `do-while`:

Вход в цикл `do-while`.

Проверка количества элементов в списке:

Проверка, если количество элементов в списке l1->`count` равно 0.

Если условие выполняется, вызов функции `AddIntoBegin` для добавления элемента в начало списка.

В противном случае, вызов функции `AddIntoEnd` для добавления элемента в конец списка.

Вывод сообщения и ввод значения `f1`:

Вывод сообщения "Добавить ещё? 1- да 0 - нет".

Ввод значения `f1` с использованием `std::cin`.

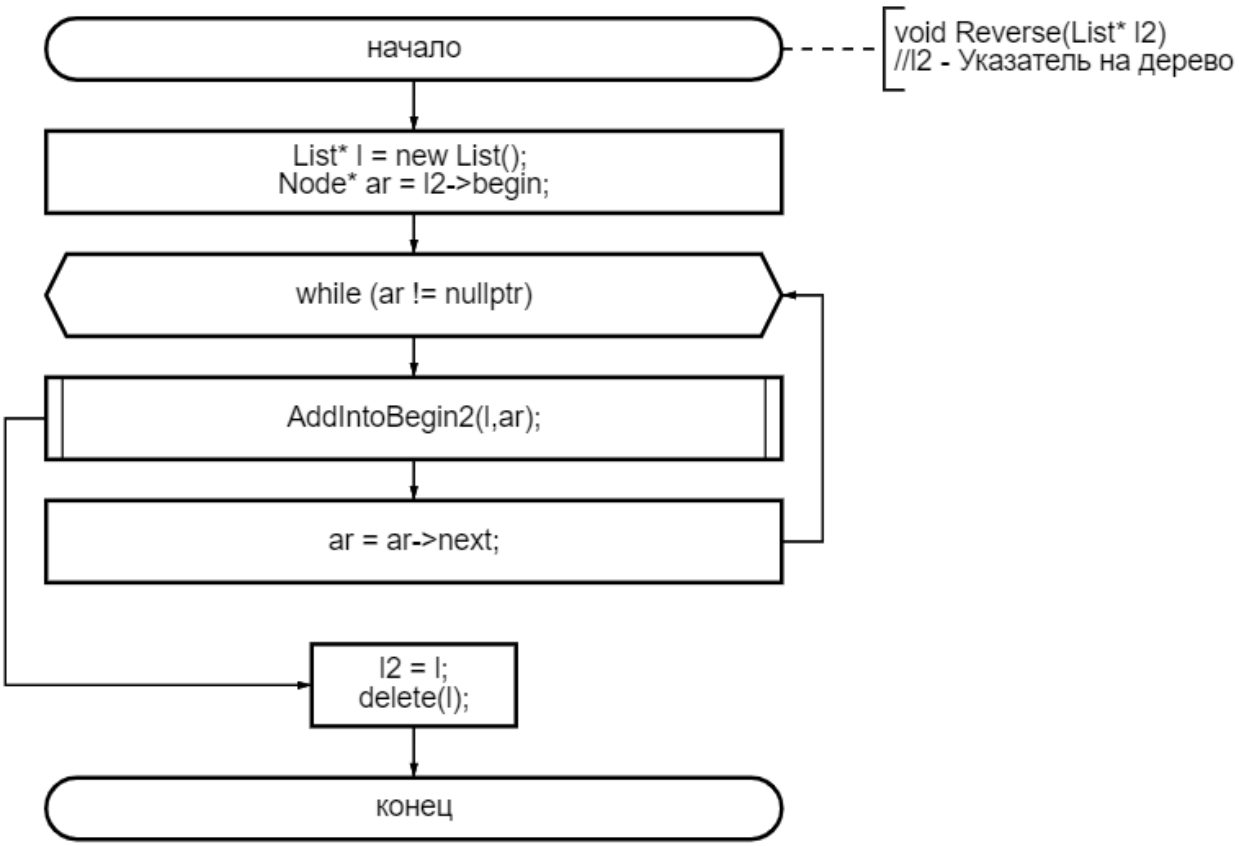
Проверка значения f1 для продолжения цикла:

Проверка значения переменной `f1`. Если `f1` равно 1, продолжение цикла.

В противном случае, завершение цикла.

Конец цикла `do-while`:

Выход из цикла `do-while`



Создание временного списка `l`:

Создание нового списка `l`.

Инициализация указателя `ar`:

Инициализация указателя `ar` значением `l2->begin`.

Цикл обращения списка:

Начало цикла `while` с условием `ar != nullptr`.

Вызов функции `AddIntoBegin2` для добавления текущего узла `ar` в начало временного списка `l`.

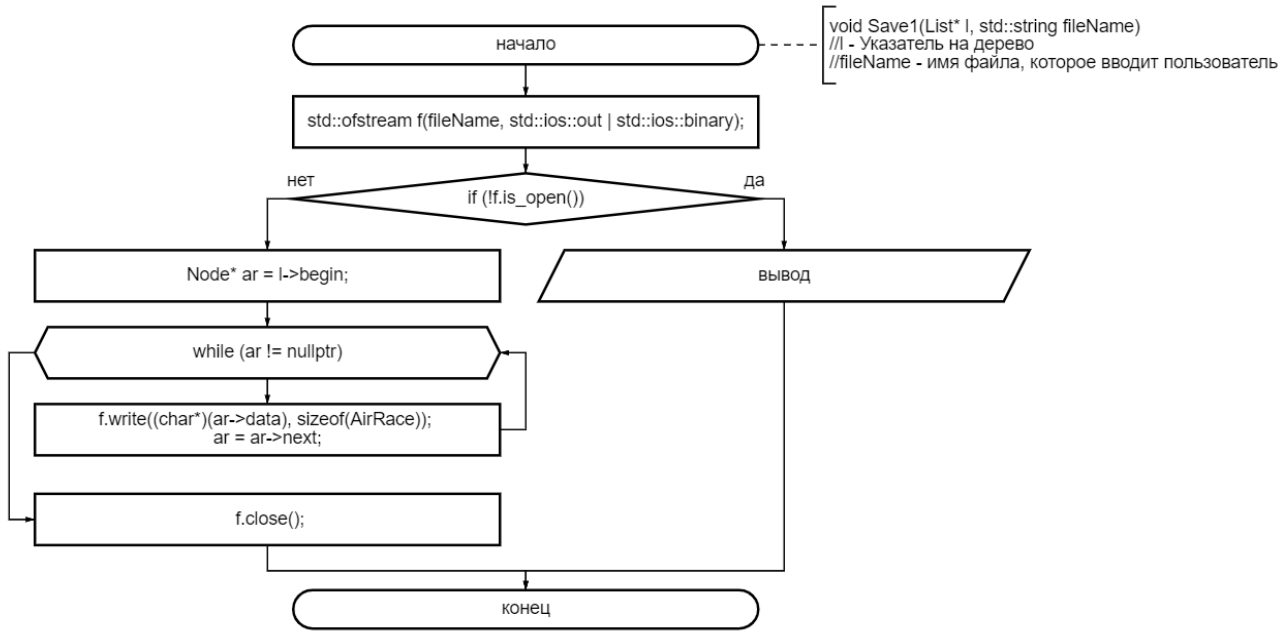
Обновление указателя `ar` на следующий узел `ar->next`.

Присвоение временного списка основному списку:

Присвоение значению `l` основного списка `l2`.

Конец:

Завершение функции.



Открытие файла для записи:

Попытка открытия файла с использованием `std::ofstream` и режимом `std::ios::out | std::ios::binary`.

Проверка успешности открытия файла:

Проверка, если файл успешно открыт. Если нет, вывод сообщения об ошибке и завершение функции.

Инициализация указателя `ar`:

Инициализация указателя `ar` значением `l->begin`.

Запись данных в файл:

Начало цикла `while` с условием `ar != nullptr`.

Запись данных из объекта `ar->data` в файл с использованием `f.write`.

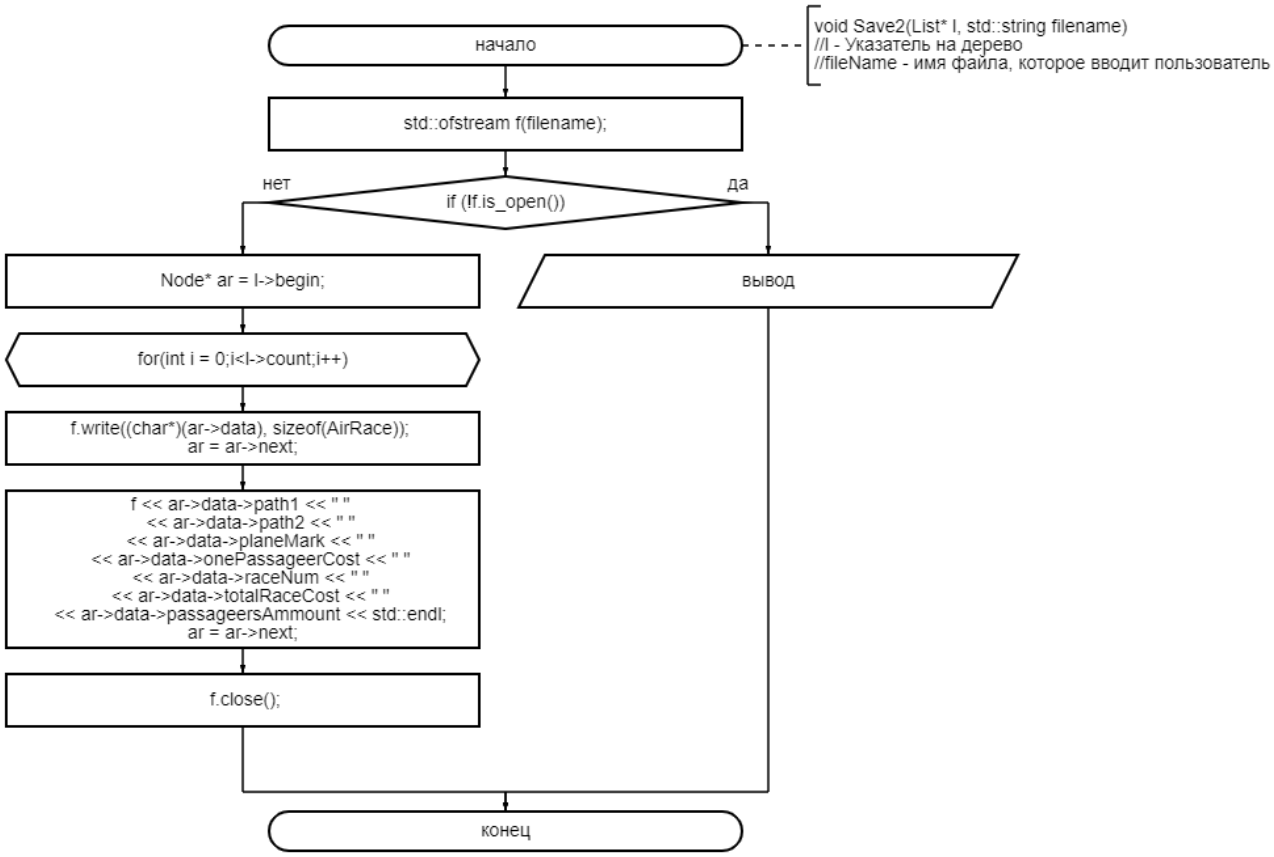
Обновление указателя `ar` на следующий узел `ar->next`.

Закрытие файла:

Закрытие файла с использованием `f.close()`.

Конец:

Завершение функции.



Открытие файла для записи:

Попытка открытия файла с использованием `std::ofstream`.

Проверка успешности открытия файла:

Проверка, если файл успешно открыт. Если нет, вывод сообщения об ошибке и завершение функции.

Инициализация указателя `ar`:

Инициализация указателя `ar` значением `l->begin`.

Цикл записи данных в файл:

Начало цикла `for` с условием `i < l->count`.

Запись данных из объекта `ar->data` в файл в текстовом формате с использованием `f <<`.

Обновление указателя `ar` на следующий узел `ar->next`.

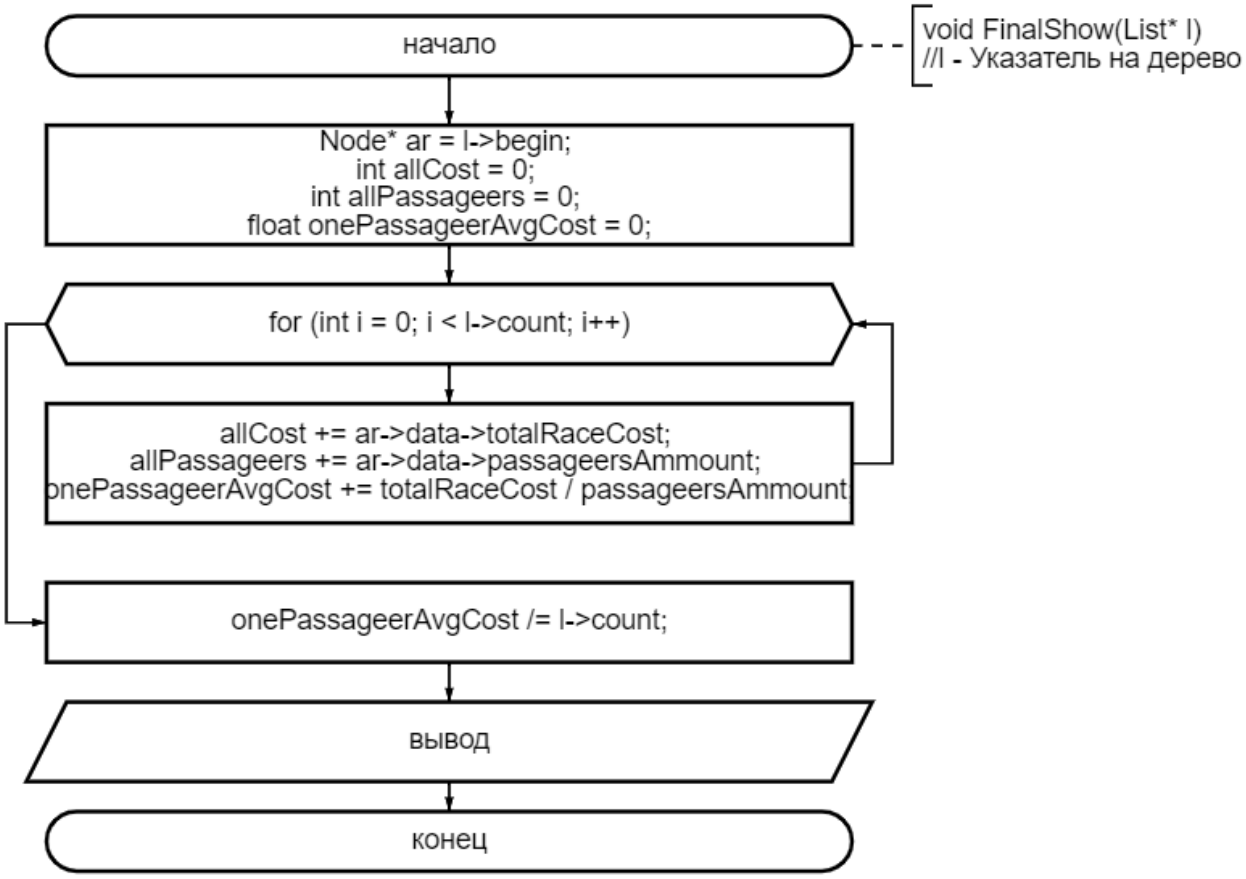
Увеличение значения переменной `i` на 1.

Закрытие файла:

Закрытие файла с использованием `f.close()`.

Конец:

Завершение функции.



Инициализация переменных:

Инициализация переменных `allCost`, `allPassageers` и `onePassageerAvgCost` значением 0.

Инициализация указателя `ar`:

Инициализация указателя ar значением `l->begin`.

Цикл подсчета статистики:

Начало цикла `for` с условием `i < l->count`.

Увеличение значения переменной `allCost` на `ar->data->totalRaceCost`.

Увеличение значения переменной `allPassageers` на `ar->data->passageersAmmount`.

Увеличение значения переменной `onePassageerAvgCost` на `ar->data->totalRaceCost / ar->data->passageersAmmount`.

Обновление указателя `ar` на следующий узел `ar->next`.

Увеличение значения переменной `i` на 1.

Расчет средней стоимости для одного пассажира:

Расчет значения `onePassageerAvgCost` как отношение суммы стоимостей к сумме количества пассажиров.

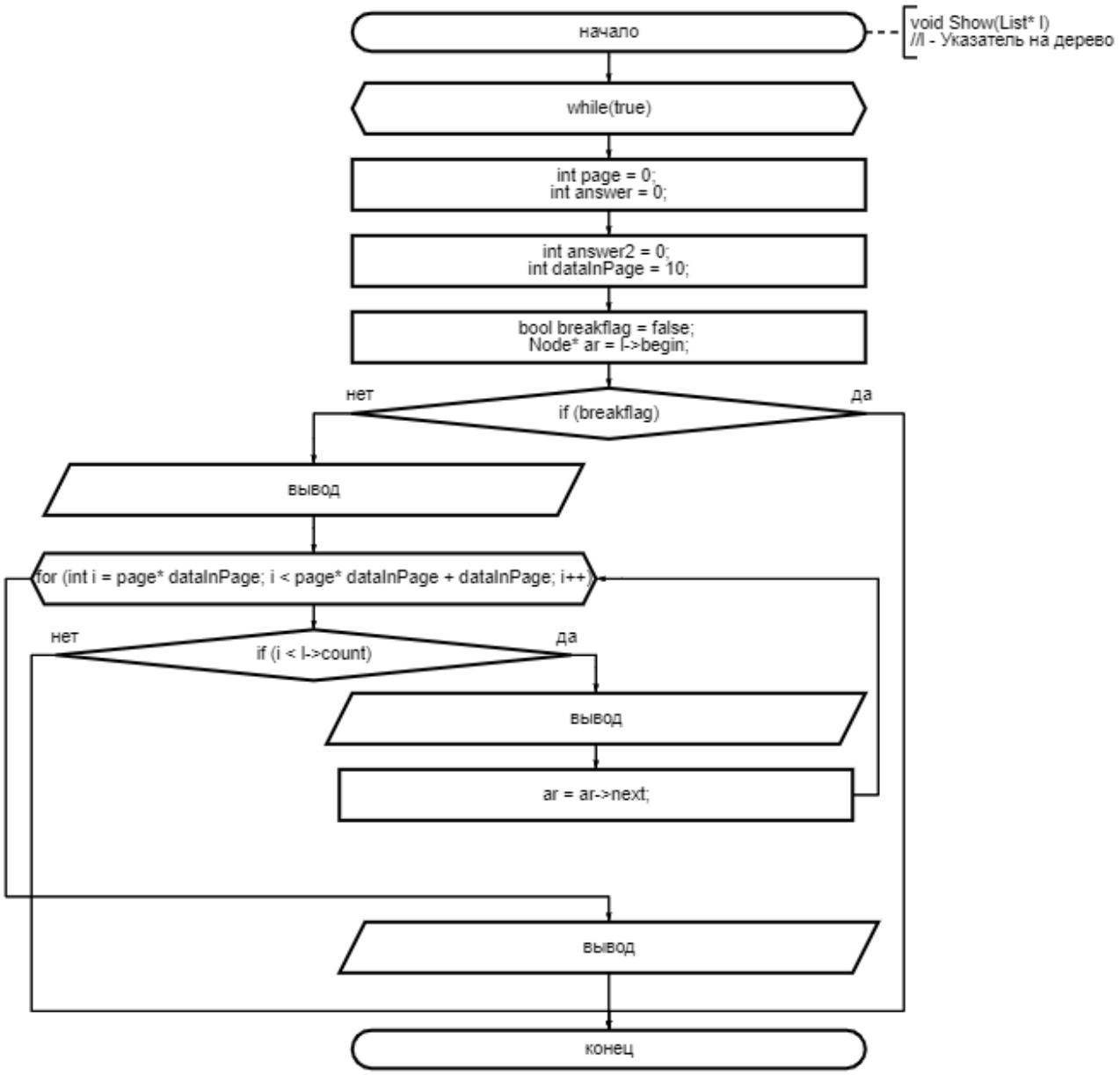
Разделение значения `onePassageerAvgCost` на количество элементов в списке `l->count`.

Вывод результатов:

Вывод стоимости всех рейсов, количества пассажиров за всё время и средней стоимости провозки 1 пассажира за всё время.

Конец:

Завершение функции.



Инициализация переменных:

Инициализация переменных `page`, `answer`, `answer2`, `dataInPage` значением 0.

Инициализация переменной `breakflag` значением `false`.

Инициализация указателя `ar` значением `l->begin`.

Начало цикла `do-while`:

Вход в цикл `do-while`.

Вывод заголовка таблицы:

Вывод строки с заголовком таблицы.

Цикл вывода данных:

Начало цикла `for` с условием `i < page * dataInPage + dataInPage`.

Проверка, если `i` меньше общего количества элементов в списке `l->count`.

Вывод данных из объекта `ar->data`.

Обновление указателя `ar` на следующий узел `ar->next`.

Вывод меню и ввод ответа:

Вывод сообщения с меню выбора: "Введите: 1 - предыдущая страница 2 - следующая страница 3 - выбранная страница 4 - Итоговые данные 0 - выход".

Ввод значения `answer` с использованием `std::cin`.

Обработка ответа:

Ветвление `switch` в зависимости от значения переменной `answer`.

По умолчанию (`default`): установка `breakflag` в `true`.

В случае 1: уменьшение `page` на 1 и обновление указателя `ar` для вывода предыдущей страницы.

В случае 2: увеличение `page` на 1 и обновление указателя `ar` для вывода следующей страницы.

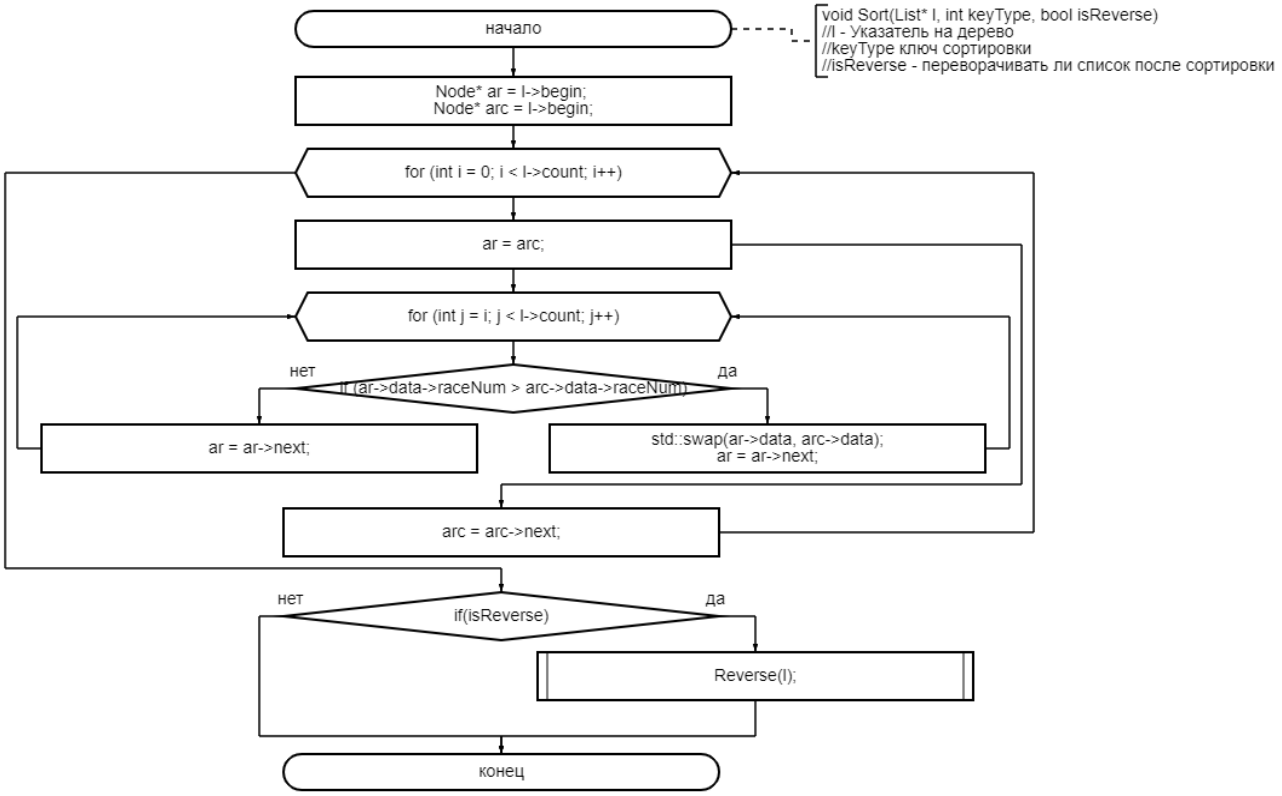
В случае 3: ввод значения `answer2` и изменение `page` и `ar` в соответствии с введенным значением.

В случае 4: вызов функции `FinalShow(l)` для вывода итоговых данных.

В случае 0: установка `breakflag` в `true`.

Конец цикла `do-while`:

Выход из цикла `do-while`.



Инициализация указателей:

Инициализация указателей `ar` и `arc` значением `l->begin`.

Выбор типа сортировки:

Ветвление `switch` в зависимости от значения переменной `keyType`.

В случае 1: Сортировка по номеру рейса `(raceNum)` .

В случае 2: Сортировка по марке самолёта (`planeMark`).

В случае 3: Сортировка по пути (`path1`).

В случае 4: Сортировка по общим затратам (`totalRaceCost`).

В случае 5: Сортировка по количеству пассажиров (`passageersAmmount`).

В случае 6: Сортировка по стоимости 1 пассажира (`onePassageerCost`).

В случае по умолчанию: Вывод сообщения об ошибке ("`Wrong input data`").

Вложенные циклы сортировки:

Начало внешнего цикла `for` с индексом `i` от 0 до `l->count`.

Инициализация указателя `ar` значением `arc`.

Начало внутреннего цикла `for` с индексом `j` от `i` до `l->count`.

Сравнение значений в зависимости от выбранного типа сортировки, и при необходимости выполнение обмена элементов.

Обновление указателя `ar` на следующий узел `ar->next`.

Обновление указателя `arc` на следующий узел `arc->next`.

Конец внутреннего цикла.

Сортировка в обратном порядке:

Проверка, если флаг `isReverse` не установлен (`!isReverse`).

Конец внешнего цикла:

Вызов функции `Reverse(l)` для инвертирования порядка элементов.

Конец внешнего цикла.

1.7 Обоснование состава технических и программных средств, необходимых для работы программы

При выполнении программа потребляет 933байта-1 мб оперативной памяти (Приложение А), код и 100 элементов списка занимают 56кб памяти, а exe файл 2,3 мб поэтому для работы программы потребуются как минимум:

1 Процессор (CPU): Производительный процессор, поддерживающий архитектуру программы и способный обрабатывать её вычислительные задачи

2 Оперативная память (RAM): Достаточное количество оперативной памяти для загрузки программы и обеспечения её нормальной работы. В данном случае, программа потребляет 1МБ, следовательно, должно быть достаточно оперативной памяти

3 Хранилище данных (Жесткий диск/SSD): Пространство для хранения исполняемых файлов программы, данных, и временных файлов. В данном случае достаточно 2,36мб памяти, однако при увеличении количества элементов в

программе потребуется дополнительная память. В среднем 100 элементов занимают 1,4 Кб или 4,8Кб памяти в зависимости от того типизированное или не типизированное сохранение.

4 Устройства ввода/вывода

5 Операционная система: Windows XP – Windows 11

6 Среда выполнения: VS Community Edition 2019/2022

7 Компилятор: встроенный компилятор VS Community Edition

2 ВЫПОЛНЕНИЕ ПРОГРАММЫ

2.1 Условия выполнения программы

Аппаратные средства:

Процессор: Двухъядерный процессор с тактовой частотой 2.0 КГц или выше.

Оперативная память (RAM): Минимум 1 мб оперативной памяти.

Хранилище данных: Свободное место на жестком диске не менее 2,5 МБ для установки программы и хранения данных.

Монитор: Рекомендуется разрешение экрана не менее 1280x800 пикселей.

Программные средства:

Операционная система: Windows XP-Windows 11

Компилятор C++: Для сборки и выполнения программы требуется компилятор C++ VS (Community Edition)

Стандартная библиотека C++: Обеспечивает базовую функциональность программы и должна быть доступной для компилятора.

Файловая система: Доступ к файловой системе для чтения и записи данных.

Консоль или терминал: Для взаимодействия с текстовым интерфейсом пользователя.

Дополнительные требования:

Права администратора: Не требуются, если программа не выполняет операции, требующие повышенных привилегий.

Дополнительные рекомендации:

Резервное копирование данных: Регулярное создание резервных копий файлов с данными программы для предотвращения потери информации.

Обновленные драйверы: Рекомендуется наличие актуальных драйверов для обеспечения стабильной работы программы.

2.2 Загрузка и запуск программы

Для успешной инсталляции программы следует перейти на страницу проекта в GitHub после 01.01.2024 года и загрузить соответствующий файл (рис. 1). После завершения загрузки пользователь может перейти в папку AirRaces\x64\Debug и запустить исполняемый файл Labrab.exe для запуска программы (рис. 2). При запуске пользователь встречается с меню-ориентированным интерфейсом, предоставляющим 10 функций и опцию выхода из программы (рис. 3).

Ввод данных

Для ввода данных пользователь может воспользоваться функцией организации списка (рис. 4). Эта функция добавляет элемент в начало списка, если их количество меньше 1, и в конец, если элементов больше 1. После добавления элемента пользователь может ввести дополнительные данные или вернуться в главное меню. Программа обрабатывает исключения, предлагая повторить ввод при некорректных данных или превышении лимита символов (рис. 5).

Отображение данных

Функция Show выводит задаваемое количество записей в списке на экран. Пользователь получает информацию о следующих данных: номер рейса, название пути, марка самолёта, общая стоимость рейса, количество пассажиров на рейсе, средняя стоимость провозки 1 пассажира (рис 6). Также пользователь может перейти на предыдущую/следующую/выбранную страницу, состоящую из определённого количества записей. Если страница не существует, то программа выведет соответствующее сообщение (рис 7). Помимо просмотра страниц, пользователь имеет возможность посмотреть итоговые данные за все рейсы, занесённые в список. Подсчитываются следующие данные: итоговые затраты на

все рейсы, итоговое количество пассажиров, итоговая средняя стоимость провозки 1 пассажира на рейсе.

Добавление в список

Помимо организации списка пользователь имеет доступ к функции добавления в начало/конец списка (рис 8). Помимо этого, программа может обрабатывать исключения, если пользователь ввёл некорректные данные или превысил допустимое количество символов, при вводе данных. На (рис 9) продемонстрирован вывод данных на экран после добавления пользователем данных в начало и конец списка.

Сохранение и загрузка

С функциями сохранения и загрузки пользователь может хранить и восстанавливать данные из файла, находящегося в той же папке, что и программа. Пользователь вводит имя файла самостоятельно. Опции сохранения и загрузки могут быть типизированными и нетипизированными, с предварительным удалением предыдущих данных при загрузке (рис. 10-рис. 14).

Удаление элемента

Пользователь может удалить элемент из списка по ключу "номер рейса". Программа запрашивает подтверждение перед удалением, выводя номер рейса и маршрут. После подтверждения элемент удаляется (рис. 15).

Корректировка данных

Функция корректировки позволяет пользователю изменить данные элемента с указанным номером рейса. При корректировке пользователь вводит новые данные для изменения элемента (рис. 16).

Сортировка списка

При помощи функции сортировки пользователь может сортировать список по любому ключу на выбор: по номеру рейса, марке самолёта, названию маршрута, итоговой стоимости рейса, количеству пассажиров, средней стоимости одного пассажира. При сортировке пользователь также может указать в порядке убывания или возрастания он хочет провести сортировку по выбранному им ключу. (рис 17)

Поиск

Функция поиска позволяет пользователю найти маршрут с заданным именем. Причём будет проводится поиск как названия города до знака “-”, так и после него. (рис 18)

Выход из программы

После завершения работы с программой пользователь может воспользоваться функцией выхода (рис. 19).

3 ЗАКЛЮЧЕНИЕ

В ходе разработки программы для управления данными о воздушных рейсах были решены несколько ключевых задач, направленных на обеспечение эффективного функционирования и удобного взаимодействия с пользователем.

В первую очередь, разработанная программа успешно решает задачу организации данных о воздушных рейсах, предоставляя пользователю удобные инструменты для ввода, отображения, хранения и обработки информации. Применение двунаправленного списка в основе алгоритма обеспечивает эффективный просмотр данных в обоих направлениях, что способствует удобству работы с программой.

Благодаря функциональности отображения данных в удобной табличной форме, предоставленной функцией Show, пользователь может легко получать информацию о рейсах, а также выполнять различные операции с данными, такие как сортировка и поиск.

Программа успешно решает задачу сохранения и загрузки данных, обеспечивая пользователя возможность хранить информацию о рейсах в файле и восстанавливать её при необходимости. Типизированные и нетипизированные опции сохранения и загрузки расширяют функционал программы, позволяя пользователю выбирать наиболее подходящий способ работы с данными.

В результате проделанной работы, цель проекта - разработать программу для эффективного управления данными о воздушных рейсах - была достигнута, необходимые функции реализованы. Получено программное обеспечение, способное обеспечивать организацию, отображение, редактирование и сохранение данных, связанных с воздушными рейсами.

Рекомендуется использовать программу для учета и анализа данных о воздушных рейсах в авиакомпаниях, аэропортах или других организациях, связанных с воздушным транспортом. Пользователи смогут воспользоваться удобным и функциональным интерфейсом для эффективной работы с данными, а также для проведения анализа и оптимизации операций, связанных с воздушными

4 СПИСОК ИСПОЛЬЗУЕМЫХ МАТЕРИАЛОВ

1. Подбельский, В. В. Язык СИ++ : учебное пособие / В. В. Подбельский. — 5- е изд. — Москва : Финансы и статистика, 2022. — 560 с. — ISBN 978-5- 00184-082-4. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book.

2. Керниган Б., Ритчи Д. Язык программирования СИ: пер. с англ./Под ред. и с предисл. В.С. Штаркмана. — 2-е изд., перераб. и доп. — М. ; СПб. ; К. : Вильямс, 2006. —272с.

3. Павловская, Т. А. Программирование на языке С++ : учебное пособие / Т. А. Павловская. — 2-е изд. — Москва : ИНТУИТ, 2016. — 154 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/100409.

4. Павловская Т. А. Паскаль. Программирование на языке высокого уровня : практикум / Т. А. Павловская. — СПб. : Питер, 2006. — 317 с.

5. Павловская Т. А. Паскаль. Программирование на языке высокого уровня : учеб. для вузов / Т. А. Павловская. — СПб. : Питер, 2008. — 393 с.

6. Методические указания к лабораторным работам по дисциплине «Алгоритмизация и программирование» для студентов дневной и заочной форм обучения направлений 09.03.02 – «Информационные системы и технологии» и 09.03.03 – «Прикладная информатика», часть 1 / Сост. В. Н. Бондарев, Т. И. Сметанина, А. К. Забаштанский, А.Ю. Абрамович – Севастополь: Изд-во СевГУ, 2021. – 88 с.

7. Фридман, А. Л. Язык программирования Си++ : учебное пособие / А. Л. Фридман. — 2-е изд. — Москва : ИНТУИТ, 2016. — 218 с. — ISBN 5-9556- 0017-5. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/100541.

8. Быков, А. Ю. Решение задач на языках программирования Си и Си++ : методические указания / А. Ю. Быков. — Москва : МГТУ им. Баумана, 2017. — 248

с. — ISBN 978-5-7038-4577-6. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/103505.

9. Рацеев, С. М. Программирование на языке Си / С. М. Рацеев. — 2-е изд., стер. — Санкт-Петербург : Лань, 2023. — 332 с. — ISBN 978-5-507-47236- 9. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/351863. 10.Разработка САПР : в

10 кн. Кн. 3. Проектирование программного обеспечения САПР : практ. пособие / Б. С. Федоров, Н. Б. Гуляев ; под ред. А.В. Петрова. — М. : Высш. шк., 1990. — 159с.

5 ПРИЛОЖЕНИЯ

5.1 **Приложение А требуемые ресурсы для работы программы**

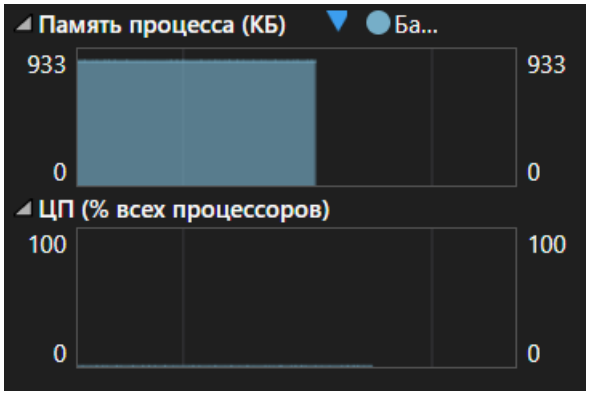


Рис 1 - Потребляемая память при выполнении программы

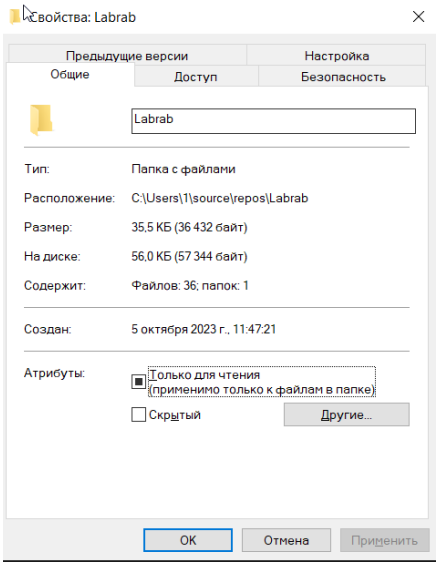


Рис 2 - Память, требуемая для хранения файлов программы

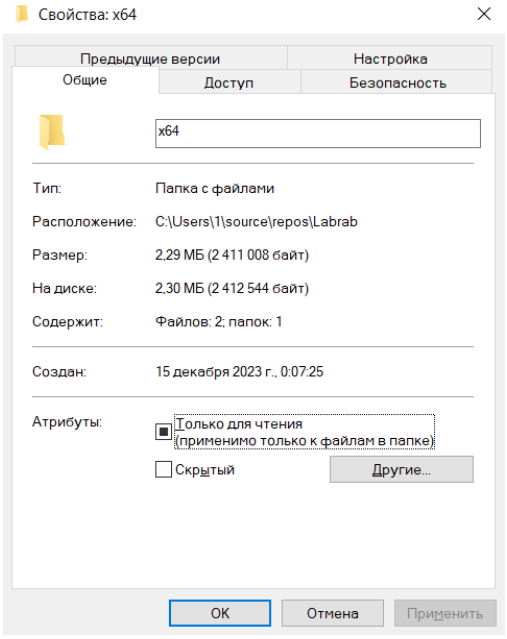


Рис 3 - Память требуемая для хранения exe файла программы

5.2 Приложение В загрузка и работа программы

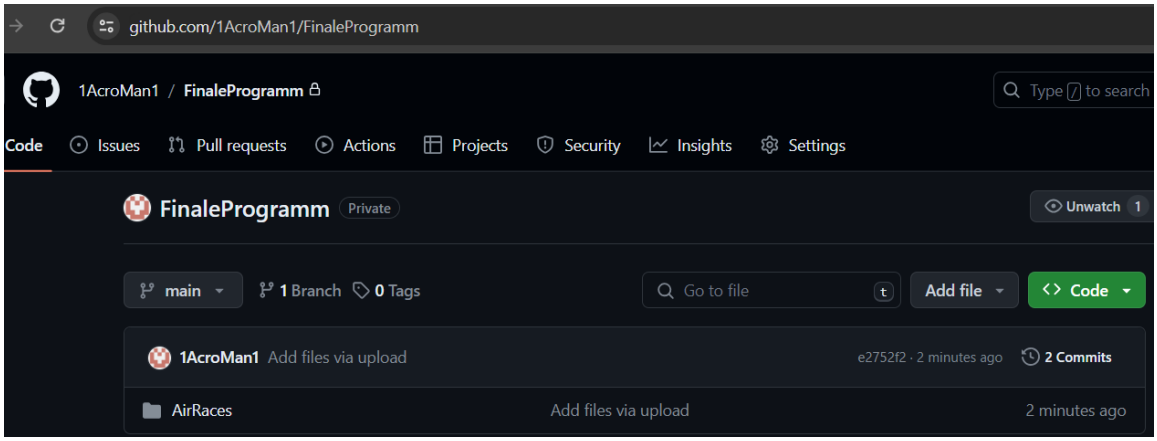


Рис 1 - Инсталляция программы по ссылке:

https://github.com/1AcroMan1/FinaleProgramm

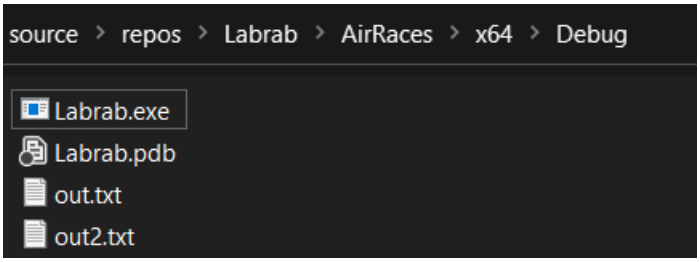


Рис 2 - Путь для достижения exe файла, который служит для запуска программы

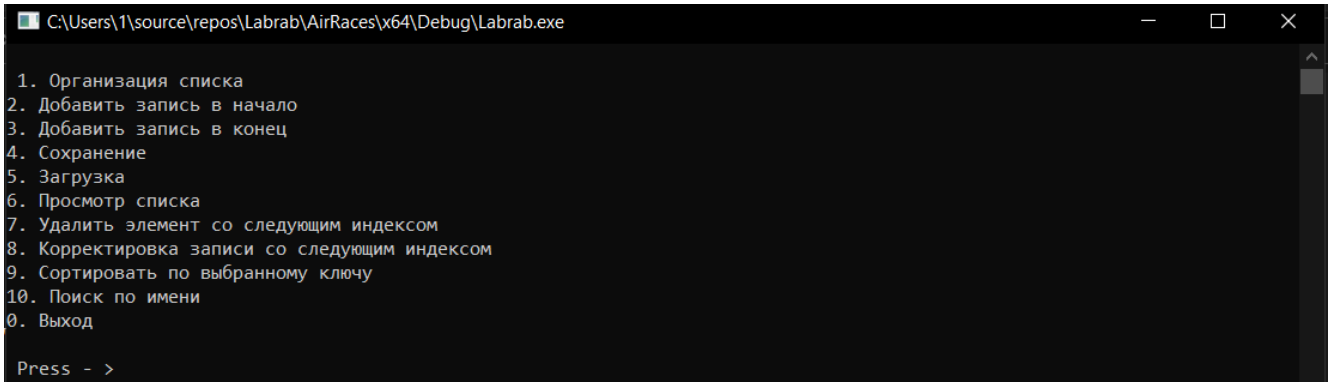


Рис 3 - Меню-ориентированный интерфейс программы

```
 1. Организация списка
2. Добавить запись в начало
3. Добавить запись в конец
4. Сохранение
5. Загрузка
6. Просмотр списка
7. Удалить элемент со следующим индексом
8. Корректировка записи со следующим индексом
9. Сортировать по выбранному ключу
10. Поиск по имени
0. Выход

 Press - >1
Напишите имя первого города (Лондон, Севастополь и т.п). Название маршрута не должно превышать 20 символов
Севастополь
Напишите имя второго города (Лондон, Севастополь и т.п). Название маршрута не должно превышать 20 символов
Москва
Введите марку самолёта. Значение не должно превышать 6 символов
марка
Введите затраты на рейс
1000000
Введите количество пассажиров
60
Создание успешно
Добавить ещё? 1- да 0 - нет
```

Рис 4 - Функция организации списка

```
Добавить ещё? 1- да 0 - нет
1
Напишите имя первого города (Лондон, Севастополь и т.п). Название маршрута не должно превышать 20 символов
Москва
Напишите имя второго города (Лондон, Севастополь и т.п). Название маршрута не должно превышать 20 символов
Симферополь
Введите марку самолёта. Значение не должно превышать 6 символов
щщфлщлащ
Значение маршрута больше 6 символов. Попробуйте ещё раз
Введите марку самолёта. Значение не должно превышать 6 символов
марка
Введите затраты на рейс
ui
Вы не можете импользовать это значение. Попробуйте снова
Введите затраты на рейс
9000000
Введите количество пассажиров
yt
Вы не можете импользовать это значение. Попробуйте снова
Введите количество пассажиров
40
Создание успешно
Добавить ещё? 1- да 0 - нет
0

 1. Организация списка
2. Добавить запись в начало
3. Добавить запись в конец
```

Рис 5 - Пример добавления новых записей в список с помощью функции организации списка и обработка исключений.

```
 1. Организация списка
2. Добавить запись в начало
3. Добавить запись в конец
4. Сохранение
5. Загрузка
6. Просмотр списка
7. Удалить элемент со следующим индексом
8. Корректировка записи со следующим индексом
9. Сортировать по выбранному ключу
10. Поиск по имени
0. Выход

 Press - >6
|Номер рейса|      Название пути      |Марка самолёта|Общие затраты|Количество пассажиров|Стоимость 1 пассажира|
|         0 |       Sevastopol-Moskov |         mark |      900000 |                  60 |             15000,00|
|         1 |           London-Moskov |         mark |   100000000 |                 120 |            833333,00|
Страница: 0 Введите: 1 - предыдущая страница 2 - следующая страница 3 - выбранная страница 4 - Итоговые данные 0 - выход
```

Рис 6 – Функция просмотра списка постранично.

```
|Номер рейса|      Название пути      |Марка самолёта|Общие затраты|Количество пассажиров|Стоимость 1 пассажира|
|         0 |       Sevastopol-Moskov |        plane |     1200000 |                  90 |             13333,00|
|         1 |           London-Parise |         avia |     1700000 |                 106 |             16037,00|
|         2 |             Moscov-Kiev |         mark |      800000 |                  70 |             11428,00|
Страница: 0 Введите: 1 - предыдущая страница 2 - следующая страница 3 - выбранная страница 4 - Итоговые данные 0 - выход1
|Номер рейса|      Название пути      |Марка самолёта|Общие затраты|Количество пассажиров|Стоимость 1 пассажира|
|         0 |       Sevastopol-Moskov |        plane |     1200000 |                  90 |             13333,00|
|         1 |           London-Parise |         avia |     1700000 |                 106 |             16037,00|
|         2 |             Moscov-Kiev |         mark |      800000 |                  70 |             11428,00|
Страница: 0 Введите: 1 - предыдущая страница 2 - следующая страница 3 - выбранная страница 4 - Итоговые данные 0 - выход2
|Номер рейса|      Название пути      |Марка самолёта|Общие затраты|Количество пассажиров|Стоимость 1 пассажира|
|         0 |       Sevastopol-Moskov |        plane |     1200000 |                  90 |             13333,00|
|         1 |           London-Parise |         avia |     1700000 |                 106 |             16037,00|
|         2 |             Moscov-Kiev |         mark |      800000 |                  70 |             11428,00|
Страница: 0 Введите: 1 - предыдущая страница 2 - следующая страница 3 - выбранная страница 4 - Итоговые данные 0 - выход3
Введите страницу
1
Неверная страница
|Номер рейса|      Название пути      |Марка самолёта|Общие затраты|Количество пассажиров|Стоимость 1 пассажира|
|         0 |       Sevastopol-Moskov |        plane |     1200000 |                  90 |             13333,00|
|         1 |           London-Parise |         avia |     1700000 |                 106 |             16037,00|
|         2 |             Moscov-Kiev |         mark |      800000 |                  70 |             11428,00|
Страница: 0 Введите: 1 - предыдущая страница 2 - следующая страница 3 - выбранная страница 4 - Итоговые данные 0 - выход4
Стоимсоть всех рейсов: 3700000
Количество пассажиров за всё время: 266
Средняя стоимость провозки 1 пассажира за всё время: 13599.3
|Номер рейса|      Название пути      |Марка самолёта|Общие затраты|Количество пассажиров|Стоимость 1 пассажира|
|         0 |       Sevastopol-Moskov |        plane |     1200000 |                  90 |             13333,00|
|         1 |           London-Parise |         avia |     1700000 |                 106 |             16037,00|
|         2 |             Moscov-Kiev |         mark |      800000 |                  70 |             11428,00|
Страница: 0 Введите: 1 - предыдущая страница 2 - следующая страница 3 - выбранная страница 4 - Итоговые данные 0 - выход
```

Рис 7 – просмотр постранично, а также вывод итоговых данных

```
 Press - >2
Напишите имя первого города (Лондон, Севастополь и т.п). Название маршрута не должно превышать 20 символов
1
Напишите имя второго города (Лондон, Севастополь и т.п). Название маршрута не должно превышать 20 символов
1
Введите марку самолёта. Значение не должно превышать 6 символов
1
Введите затраты на рейс
1
Введите количество пассажиров
1
Создание успешно

 1. Организация списка
2. Добавить запись в начало
3. Добавить запись в конец
4. Сохранение
5. Загрузка
6. Просмотр списка
7. Удалить элемент со следующим индексом
8. Корректировка записи со следующим индексом
9. Сортировать по выбранному ключу
10. Поиск по имени
0. Выход

 Press - >3
Напишите имя первого города (Лондон, Севастополь и т.п). Название маршрута не должно превышать 20 символов
2
Напишите имя второго города (Лондон, Севастополь и т.п). Название маршрута не должно превышать 20 символов
2
Введите марку самолёта. Значение не должно превышать 6 символов
2
Введите затраты на рейс
2
Введите количество пассажиров
2
```

Рис 8 – добавление записей в начало и конец списка

```
 Press - >6
|Номер рейса|      Название пути      |Марка самолёта|Общие затраты|Количество пассажиров|Стоимость 1 пассажира|
|         3 |                    1-1 |            1 |           1 |                   1 |                1,00|
|         0 |       Sevastopol-Moskov |        plane |     1200000 |                  90 |            13333,00|
|         1 |           London-Parise |         avia |     1700000 |                 106 |            16037,00|
|         2 |             Moscov-Kiev |         mark |      800000 |                  70 |            11428,00|
|         4 |                    2-2 |            2 |           2 |                   2 |                1,00|
Страница: 0 Введите: 1 - предыдущая страница 2 - следующая страница 3 - выбранная страница 4 - Итоговые данные 0 - выход
```

Рис 9 – список после добавления в начало и конец

```
 1. Организация списка
2. Добавить запись в начало
3. Добавить запись в конец
4. Сохранение
5. Загрузка
6. Просмотр списка
7. Удалить элемент со следующим индексом
8. Корректировка записи со следующим индексом
9. Сортировать по выбранному ключу
10. Поиск по имени
0. Выход

 Press - >4
Введите имя txt файла

 Press - >out3.txt
Выберите тип сохранения. 0 - нетипизированное 1 - типизированное
```

Рис 10 – нетипизированное сохранение

```
1 1 1 1 3 1 1
Sevastopol Moskov plane 13333 0 1200000 90
London Parise avia 16037 1 1700000 106
Moscov Kiev mark 11428 2 800000 70
2 2 2 1 4 2 2
```

Рис 11 – файл с данными о нетипизированном сохранении

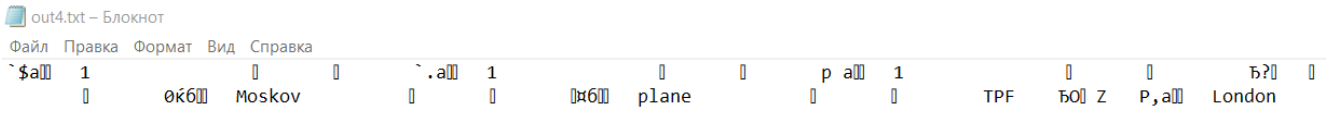


Рис 12 – файл с данными типизированного сохранения

```
 1. Организация списка
2. Добавить запись в начало
3. Добавить запись в конец
4. Сохранение
5. Загрузка
6. Просмотр списка
7. Удалить элемент со следующим индексом
8. Корректировка записи со следующим индексом
9. Сортировать по выбранному ключу
10. Поиск по имени
0. Выход

 Press - >5
Введите имя txt файла

 Press - >out3.txt
Выберите тип загрузки. 0 - нетипизированная 1 - типизированная

 Press - >0

 1. Организация списка
2. Добавить запись в начало
3. Добавить запись в конец
4. Сохранение
5. Загрузка
6. Просмотр списка
7. Удалить элемент со следующим индексом
8. Корректировка записи со следующим индексом
9. Сортировать по выбранному ключу
10. Поиск по имени
0. Выход

 Press - >6
|Номер рейса|      Название пути      |Марка самолёта|Общие затраты|Количество пассажиров|Стоимость 1 пассажира|
|         0 |       Sevastopol-Moskov |        plane |     1200000 |                  90 |             13333,00|
|         1 |           London-Parise |         avia |     1700000 |                 106 |             16037,00|
|         2 |             Moscov-Kiev |         mark |      800000 |                  70 |             11428,00|
|         3 |                     1-1 |            1 |           1 |                   1 |                 1,00|
|         4 |                     2-2 |            2 |           2 |                   2 |                 1,00|
Страница: 0 Введите: 1 - предыдущая страница 2 - следующая страница 3 - выбранная страница 4 - Итоговые данные 0 - выход
```

Рис 13 – нетипизированная загрузка

```
 Press - >5
Введите имя txt файла

 Press - >out4.txt
Выберите тип загрузки. 0 - нетипизированная 1 - типизированная

 Press - >1

 1. Организация списка
2. Добавить запись в начало
3. Добавить запись в конец
4. Сохранение
5. Загрузка
6. Просмотр списка
7. Удалить элемент со следующим индексом
8. Корректировка записи со следующим индексом
9. Сортировать по выбранному ключу
10. Поиск по имени
0. Выход

 Press - >6
|Номер рейса|      Название пути      |Марка самолёта|Общие затраты|Количество пассажиров|Стоимость 1 пассажира|
|         0 |       Sevastopol-Moskov |        plane |     1200000 |                  90 |             13333,00|
|         1 |           London-Parise |         avia |     1700000 |                 106 |             16037,00|
|         2 |             Moscov-Kiev |         mark |      800000 |                  70 |             11428,00|
|         3 |                     1-1 |            1 |           1 |                   1 |                 1,00|
|         4 |                     2-2 |            2 |           2 |                   2 |                 1,00|
Страница: 0 Введите: 1 - предыдущая страница 2 - следующая страница 3 - выбранная страница 4 - Итоговые данные 0 - выход
```

Рис 14 – типизированная загрузка

```
 Press - >6
|Номер рейса|      Назване пути      |Марка самолёта|Общие затраты|Количество пассажиров|Стоимость 1 пассажира|
|         0 |       Sevastopol-Moskov |        plane |     1200000 |                  90 |             13333,00|
|         1 |           London-Parise |         avia |     1700000 |                 106 |             16037,00|
|         2 |             Moscov-Kiev |         mark |      800000 |                  70 |             11428,00|
|         3 |                     1-1 |            1 |           1 |                   1 |                 1,00|
|         4 |                     2-2 |            2 |           2 |                   2 |                 1,00|
Страница: 0 Введите: 1 - предыдущая страница 2 - следующая страница 3 - выбранная страница 4 - Итоговые данные 0 - выход0

 1. Организация списка
2. Добавить запись в начало
3. Добавить запись в конец
4. Сохранение
5. Загрузка
6. Просмотр списка
7. Удалить элемент со следующим индексом
8. Корректировка записи со следующим индексом
9. Сортировать по выбранному ключу
10. Поиск по имени
0. Выход

 Press - >7
Введите индекс. Запись с этим индексом будет удалена

 Press - >3
Вы уверены, что хотите удалить запись под номером: 3 Путь: 1-1 1 - да 0 - нет
1
Удаление успешно

 1. Организация списка
2. Добавить запись в начало
3. Добавить запись в конец
4. Сохранение
5. Загрузка
6. Просмотр списка
7. Удалить элемент со следующим индексом
8. Корректировка записи со следующим индексом
9. Сортировать по выбранному ключу
10. Поиск по имени
0. Выход

 Press - >6
|Номер рейса|      Назване пути      |Марка самолёта|Общие затраты|Количество пассажиров|Стоимость 1 пассажира|
|         0 |       Sevastopol-Moskov |        plane |     1200000 |                  90 |             13333,00|
|         1 |           London-Parise |         avia |     1700000 |                 106 |             16037,00|
|         2 |             Moscov-Kiev |         mark |      800000 |                  70 |             11428,00|
|         4 |                     2-2 |            2 |           2 |                   2 |                 1,00|
Страница: 0 Введите: 1 - предыдущая страница 2 - следующая страница 3 - выбранная страница 4 - Итоговые данные 0 - выход
```

Рис 15 – удаление элемента со следующим номером рейса

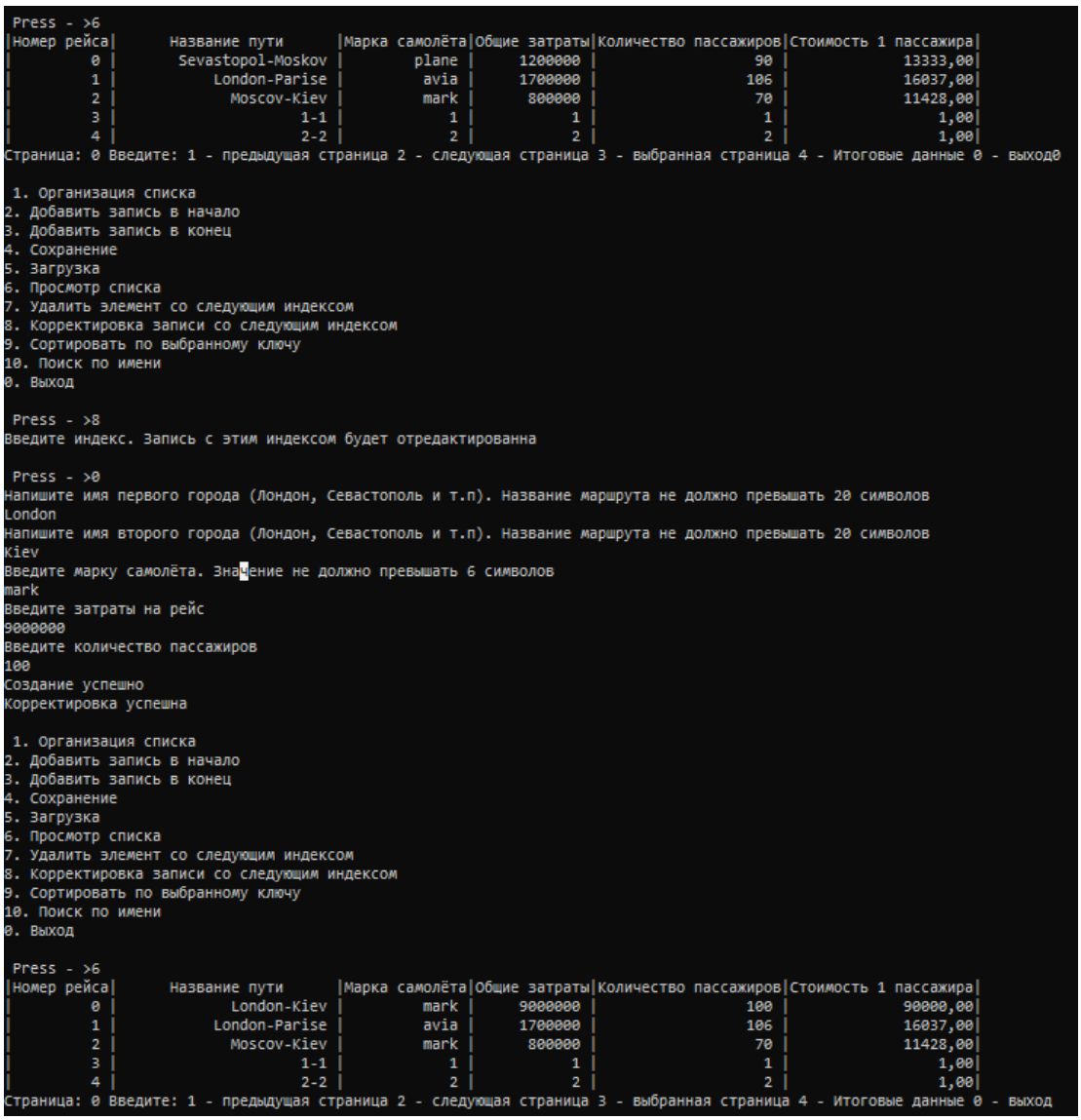


Рис 16 – корректировка записи под индексом

```
|Номер рейса|      Название пути      |Марка самолёта|Общие затраты|Количество пассажиров|Стоимость 1 пассажира|
|         0 |            London-Kiev |         mark |     9000000 |                 100 |             90000,00|
|         1 |          London-Parise |         avia |     1700000 |                 106 |             16037,00|
|         2 |           Moscov-Kiev |         mark |      800000 |                  70 |             11428,00|
|         3 |                   1-1 |            1 |           1 |                   1 |                 1,00|
|         4 |                   2-2 |            2 |           2 |                   2 |                 1,00|
Страница: 0 Введите: 1 - предыдущая страница 2 - следующая страница 3 - выбранная страница 4 - Итоговые данные 0 - выход
0

 1. Организация списка
2. Добавить запись в начало
3. Добавить запись в конец
4. Сохранение
5. Загрузка
6. Просмотр списка
7. Удалить элемент со следующим индексом
8. Корректировка записи со следующим индексом
9. Сортировать по выбранному ключу
10. Поиск по имени
0. Выход

 Press - >9

Введите ключ сортировки
1 - Номер рейса
2 - Марка самолёта
3 - Название маршрута
4 - Общие стоимостные затраты на рейс
5 - Количество пассажиров
6 - Средняя стоимость перевозки 1 пассажира на рейсе

 Press - >5
0 - По возрастанию 1 - По убыванию
1

 1. Организация списка
2. Добавить запись в начало
3. Добавить запись в конец
4. Сохранение
5. Загрузка
6. Просмотр списка
7. Удалить элемент со следующим индексом
8. Корректировка записи со следующим индексом
9. Сортировать по выбранному ключу
10. Поиск по имени
0. Выход

 Press - >6
|Номер рейса|      Название пути      |Марка самолёта|Общие затраты|Количество пассажиров|Стоимость 1 пассажира|
|         1 |          London-Parise |         avia |     1700000 |                 106 |             16037,00|
|         0 |            London-Kiev |         mark |     9000000 |                 100 |             90000,00|
|         2 |           Moscov-Kiev |         mark |      800000 |                  70 |             11428,00|
|         4 |                   2-2 |            2 |           2 |                   2 |                 1,00|
|         3 |                   1-1 |            1 |           1 |                   1 |                 1,00|
Страница: 0 Введите: 1 - предыдущая страница 2 - следующая страница 3 - выбранная страница 4 - Итоговые данные 0 - выход
```

Рис 17 – сортировка по ключу

```
 Press - >6
|Номер рейса|      Название пути      |Марка самолёта|Общие затраты|Количество пассажиров|Стоимость 1 пассажира|
|         1 |          London-Parise |         avia |     1700000 |                 106 |             16037,00|
|         0 |            London-Kiev |         mark |     9000000 |                 100 |             90000,00|
|         2 |           Moscov-Kiev |         mark |      800000 |                  70 |             11428,00|
|         4 |                   2-2 |            2 |           2 |                   2 |                 1,00|
|         3 |                   1-1 |            1 |           1 |                   1 |                 1,00|
|         5 |       Sevastopol-London |         mark |     9000000 |                  50 |            180000,00|
Страница: 0 Введите: 1 - предыдущая страница 2 - следующая страница 3 - выбранная страница 4 - Итоговые данные 0 - выход
0

 1. Организация списка
2. Добавить запись в начало
3. Добавить запись в конец
4. Сохранение
5. Загрузка
6. Просмотр списка
7. Удалить элемент со следующим индексом
8. Корректировка записи со следующим индексом
9. Сортировать по выбранному ключу
10. Поиск по имени
0. Выход

 Press - >10
Введите название города

 Press - >London
Записи с соответствующим именем:
|Номер рейса|      Название пути      |Марка самолёта|Общие затраты|Количество пассажиров|Стоимость 1 пассажира|
|         5 |       Sevastopol-London |         mark |     9000000 |                  50 |            180000,00|
|         0 |            London-Kiev |         mark |     9000000 |                 100 |             90000,00|
|         1 |          London-Parise |         avia |     1700000 |                 106 |             16037,00|
Страница: 0 Введите: 1 - предыдущая страница 2 - следующая страница 3 - выбранная страница 4 - Итоговые данные 0 - выход
```

Рис 18 – поиск рейса по названию города

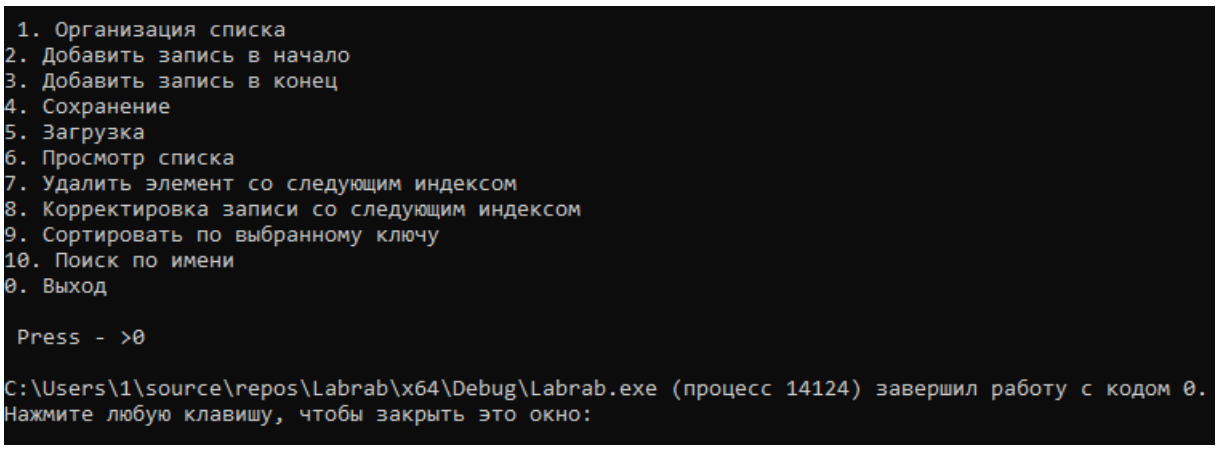

1. Организация списка
2. Добавить запись в начало
3. Добавить запись в конец
4. Сохранение
5. Загрузка
6. Просмотр списка
7. Удалить элемент со следующим индексом
8. Корректировка записи со следующим индексом
9. Сортировать по выбранному ключу
10. Поиск по имени
0. Выход
Press - >0
C:\Users\1\source\repos\Labrab\x64\Debug\Labrab.exe (процесс 14124) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно:

Рис 19 - выход

5.3 Приложение С код программы

PeredelKyrsa4.cpp – точка входа

```
#include "list.h"
int main()
{
	ClientCode();
	return 0;
	}
```

AddElement.h – заголовочный файл для добавления элемента

```
#ifndef ADD
#define ADD
#include "List.h"
void AddIntoBegin(List* l);
void AddIntoEnd(List* l);
void AddIntoBegin2(List* l, Node* ar2);
	#endif
```

AddElement.cpp – исходный файл для добавления элемента

```
#include "List.h"
#include "FillDatas.h"
void AddIntoBegin(List* l)
{
	Node* ar = new Node();
	ar->data = new AirRace();
	FillData(ar);
	ar->data->raceNum = l->count;
	ar->prev = nullptr;
	ar->next = l->begin;
	if (l->count > 0)
	{
		l->begin->prev = ar;
		l->begin = ar;
	}
	else
	{
		l->begin = l->end = ar;
	}
	l->count++;
};
void AddIntoEnd(List* l)
{
	Node* ar = new Node();
	ar->data = new AirRace();
```

```
    ar->next = nullptr;
    ar->prev = l->end;
    FillData(ar);
    ar->data->raceNum = l->count;
    if (l->end != nullptr)
        l->end->next = ar;
    if (l->count == 0)
    {
        l->begin = l->end = ar;
    }
    else
    {
        l->end = ar;
    }
    l->count++;
};
void AddIntoBegin2(List* l, Node* ar2)
{
    Node* ar = new Node();
    ar->data = new AirRace();
    ar->prev = nullptr;
    ar->next = l->begin;
    if (l->count > 0)
    {
        l->begin->prev = ar;
        l->begin = ar;
    }
    else
    {
        l->begin = l->end = ar;
    }
    l->count++;
    ar->data = ar2->data;
    };
```

CheckIndex.h – заголовочный файл для проверки на корректность индекса элемента

```
#ifndef CHECK_INDEX
#define CHECK_INDEX
bool CheckIndex(int index, int count);
#endif
```

CheckIndex.cpp – исходный файл для проверки на корректность индекса элемента

```
bool CheckIndex(int index, int count)
{
	if (count == 0)
	{
		return false;
	}
	if (index >= 0 || index < count)
		return true;
	else
		return false;
	}
```

Correct.h – исходный файл для корректировки элемента под индексом

```
#ifndef CORRECT
#define CORRECT
#include "List.h"
void Correct(List* l, int index);
	#endif
```

Correct.cpp – исходный файл для корректировки элемента под индексом

```
#include "List.h"
#include "Move.h"
#include "FillDatas.h"
#include <iostream>
void Correct(List* l, int index)
{
	Node* ar = MoveToRace(l, index);
	FillData(ar);
	std::cout << "Correct succes" << std::endl;
	};
```

Delete.h – заголовочный файл для удаления элемента под индексом

```
#ifndef DELETE
#define DELETE
#include "List.h"
void Remove(List* l, int index,bool needAsk);
void Clear(List* l);
	#endif
```

Delete.cpp – исходный файл для удаления элемента под индексом

```
#include "List.h"
#include "CheckIndex.h"
#include "Move.h"
#include <iostream>
void Remove(List* l, int index,bool needAsk)
{
    if (l->count == 0)
        return;
    if (!CheckIndex(index, l->count))
        return;
    Node* ar = MoveToRace(l,index);
    if (ar != NULL)
    {
        int an = 0;
        if (needAsk)
        {
            std::cout << "Вы уверены, что хотите удалить запись под номером: " << ar->data->raceNum << " Путь: " << ar->data->path1 << "-" << ar->data->path2 << " 1 - да 0 - нет" << std::endl;
            std::cin >> an;
        }
        if (an == 1||needAsk == false)
        {
            Node* _prev = ar->prev;
            Node* _next = ar->next;
            if (index == 0)
            {
                l->begin = _next;
            }

            else if (index == l->count - 1)
            {
                l->end = _prev;
            }
            else if (l->count > 1 && _next != NULL)
            {
                _prev->next = ar->next;
            }
            else if (l->count > 1 && _next != NULL)
            {
                _next->prev = _prev;
```

```
            }
            delete(ar);
            l->count--;
            std::cout << "Удаление успешно" << std::endl;
        }
    }
};
//----------- Clear all elements ----------------------
void Clear(List* l)
{
    int count = l->count;
    for (int i = 0; i < count-1; i++)
    {
        Remove(l, i, false);
    }
    };
```

FillDatas.h – заголовочный файл для заполнения узла дерева данными

```
#ifndef FILL
#define FILL
#include "List.h"
void FillData(Node* ar);
void PrintPathName(Node* ar, bool nameFlag);
void PrintPlaneMark(Node* ar);
void PrintTotalRaceCost(Node* ar);
void PrintAmmountPassageers(Node* ar);
    #endif
```

FillDatas.cpp – исходный файл для заполнения узла дерева данными

```
#include "List.h"
#include <iostream>
#include <limits>
void PrintPathName(Node* ar, bool nameFlag)
{
    if (nameFlag == true)
    {
        puts("print 2 city name (Londom, Sevastopol etc.).
Not longer 20 symbols");
        try
        {
            std::string b;
            std::cin >> b;
            if (b.size() > 20)
            {
                std::cout << "Value is longer than 20
symbols. Print it again" << std::endl;
                PrintPathName(ar, nameFlag);
```

```
                }
                ar->data->path2 = b;
                if    (ar->data->path1.length()    +    ar->data-
>path2.length() > 20)
                {
                    std::cout << "Both values cant be longer 20
symb" << std::endl;
                    PrintPathName(ar, 0);
                }
            }
            catch (...)
            {
                std::cout << "You cant use that value. Print it
again" << std::endl;
                std::cin.clear();

    std::cin.ignore(std::numeric_limits<std::streamsize>::
max(), '\n');
                PrintPathName(ar, nameFlag);
            }
        }
        else
        {
            puts("print 1 city name (Londom, Sevastopol etc.).
Not longer 20 symbols");
            try
            {
                if (ar == NULL)
                {
                    std::cout << "NULL!!!";
                }
                std::cin >> ar->data->path1;
                if (ar->data->path1.size() > 20)
                {
                    std::cout  <<  "Value  is  longer  than  20
symbols. Print it again" << std::endl;
                    std::cin.clear();
                }
                nameFlag = true;
                PrintPathName(ar, nameFlag);
            }
            catch (...)
            {
                std::cout << "You cant use that value. Print it
again" << std::endl;
```

```
            std::cin.clear();

        std::cin.ignore(std::numeric_limits<std::streamsize>::
max(), '\n');
            PrintPathName(ar, nameFlag);
        }
    }
};
void PrintPlaneMark(Node* ar)
{
    puts("print plane mark. Not longer 6 symbols");
    try
    {
        std::cin >> ar->data->planeMark;
        if (ar->data->planeMark.size() > 6)
        {
            std::cout << "Value is longer than 20 symbols.
Print it again" << std::endl;
            PrintPlaneMark(ar);
        }
    }
    catch (...)
    {
        std::cout << "You cant use that value. Print it
again" << std::endl;
        std::cin.clear();

    std::cin.ignore(std::numeric_limits<std::streamsize>::
max(), '\n');
        PrintPlaneMark(ar);
    }
};
void PrintTotalRaceCost(Node* ar)
{
    puts("print total race cost");
    try
    {
        std::cin >> ar->data->totalRaceCost;
    }
    catch (...)
    {
        std::cout << "You cant use that value. Print it
again" << std::endl;
        std::cin.clear();
```

```
        std::cin.ignore(std::numeric_limits<std::streamsize>::
max(), '\n');
            PrintTotalRaceCost(ar);
        }
};
void PrintAmmountPassageers(Node* ar)
{
        puts("print ammount passageers");
        try
        {
            std::cin >> ar->data->passageersAmmount;
            puts("Created successful");
        }
        catch (...)
        {
            std::cout << "You cant use that value. Print it
again" << std::endl;
            std::cin.clear();

        std::cin.ignore(std::numeric_limits<std::streamsize>::
max(), '\n');
            PrintAmmountPassageers(ar);
        }
};
void FillData(Node* ar)
{
        PrintPathName(ar, false);
        PrintPlaneMark(ar);
        PrintTotalRaceCost(ar);
        PrintAmmountPassageers(ar);
        ar->data->onePassageerCost = ar->data->totalRaceCost /
ar->data->passageersAmmount;
        };
```

Found.h – заголовочный файл для поиска элемента по ключу

```
#ifndef FOUND
#define FOUND
#include "List.h"
void Found(List* l2, std::string name);
        #endif
```

Found.cpp – исходный файл для поиска элемента по ключу

```
#include <string>
#include <iostream>
#include "List.h"
#include "AddElement.h"
#include "Show.h"
void Found(List* l2, std::string name)
{
    List* l = new List();
    Node* ar = l2->begin;
    for (int i = 0; i < l2->count;i++)
    {
        if (strcmp(ar->data->path1.c_str(), name.c_str())
== 0 || strcmp(ar->data->path2.c_str(), name.c_str()) == 0)
        {
            AddIntoBegin2(l, ar);
        }
        ar = ar->next;
    }
    if (l->count > 0)
    {
        std::cout << "Записи с соответствующим именем: " <<
std::endl;
        Show(l);
    }
    else
    {
        std::cout << "Записей с таким названием не найдено"
<< std::endl;
    }
    delete(l);
    }
```

List.h – заголовочный файл, содержащий 3 основные структуры

```
#ifndef LIST
#define LIST
#include <string>
struct AirRace
{
    std::string path1;
    std::string path2;
    std::string planeMark;//6 symb
    float onePassageerCost;
    int raceNum;
    int totalRaceCost;
```

```
    int passageersAmmount;
};
struct Node
{
    Node* next;
    Node* prev;
    AirRace* data;
};
struct List
{
    Node* begin = nullptr;
    Node* end = nullptr;
    int count = 0;
};
int ClientCode();
    #endif
```

List.cpp – исходный файл, отвечающий за меню программы

```
#define _CRT_SECURE_NO_DEPRECATE
#include "List.h"
#include "CheckIndex.h"
#include "AddElement.h"
#include "Show.h"
#include "Correct.h"
#include "Delete.h"
#include "Sort.h"
#include "Found.h"
#include "Organize.h"
#include "Save.h"
#include "Load.h"
#include <locale.h>
#include <iostream>
int ClientCode()
{
    setlocale(LC_ALL, "Russian");
    List* l1 = new List();
    int answer;
    int answer2;
    std::string answer3;
    while (1) {
        printf("\n 1. Организация списка\n"
            "2. Добавить запись в начало\n"
            "3. Добавить запись в конец\n"
            "4. Сохранение\n"
            "5. Загрузка\n"
```

```
            "6. Просмотр списка\n"
            "7. Удалить элемент со следующим индексом\n"
            "8.   Корректировка   записи   со   следующим
индексом\n"
            "9. Сортировать по выбранному ключу\n"
            "10. Поиск по имени\n"
            "0. Выход\n");
        printf("\n Press - >"), scanf("%d", &answer);
        switch (answer) {
        case 0:
            return 0;
        case 1:
            Organize(l1);
            break;
        case 2:
            AddIntoBegin(l1);
            break;
        case 3:
            AddIntoEnd(l1);
            break;
        case 4:
            std::cout  <<  "Введите  имя  txt  файла"  <<
std::endl;
            printf("\n Press - >");
            std::cin >> answer3;
            std::cout  <<  "Выберите  тип  сохранения.  0  -
нетипизированное 1 - типизированное " << std::endl;
            printf("\n Press - >");
            std::cin >> answer;
            switch (answer)
            {
            default:
                std::cout << "Такого сохранения нету" <<
std::endl;
                break;
            case 0:
                Save2(l1, answer3);
                break;
            case 1:
                Save1(l1, answer3);
                break;
            }
            break;
        case 5:
```

```
            std::cout << "Введите имя txt файла" << std::endl;
            printf("\n Press - >");
            std::cin >> answer3;
            std::cout << "Выберите тип загрузки. 0 - нетипизированная 1 - типизированная " << std::endl;
            printf("\n Press - >");
            std::cin >> answer;
            switch (answer)
            {
            default:
                std::cout << "Такой загрузки нету" << std::endl;
                break;
            case 0:
                Load2(l1, answer3);
                break;
            case 1:
                Load1(l1, answer3);
                break;
            }
            break;
        case 6:
            Show(l1);
            break;
        case 7:
            std::cout << "Введите индекс. Запись с этим индексом будет удалена" << std::endl;
            printf("\n Press - >");
            scanf("%d", &answer);
            if (CheckIndex(answer,l1->count))
            {
                Remove(l1,answer,true);
            }
            break;
        case 8:
            std::cout << "Введите индекс. Запись с этим индексом будет отредактированна" << std::endl;
            printf("\n Press - >");
            scanf("%d", &answer);
            if (CheckIndex(answer,l1->count))
            {
                Correct(l1, answer);
            }
            break;
```

```
        case 9:
            printf("\nВведите ключ сортировки\n"
                "1 - Номер рейса\n"
                "2 - Марка самолёта\n"
                "3 - Название маршрута\n"
                "4 - Общие стоимостные затраты на рейс\n"
                "5 - Количество пассажиров\n"
                "6 - Средняя стоимость перевозки 1
пассажира на рейсе\n");
            printf("\n Press - >"), scanf("%d", &answer);
            std::cout << "0 - По возрастанию 1 - По
убыванию" << std::endl;
            scanf("%d", &answer2);
            Sort(l1, answer, answer2);
            break;
        case 10:
            std::cout << "Введите название города" <<
std::endl;
            printf("\n Press - >");
            std::string a = "";
            std::cin >> a;
            Found(l1,a);
            break;
        }
    }
    };
```

Load.h – заголовочный файл для загрузки из файла

```
#ifndef LOAD
#define LOAD
void Load1(List* l, std::string fileName);
void Load2(List* l, std::string fileName);
#endif
```

Load.cpp – исходный файл для загрузки из файла

```
#include <fstream>
#include <iostream>
#include "List.h"
#include "Delete.h"
#include "AddElement.h"
#include "Sort.h"
void Load1(List* l, std::string fileName) {
    Clear(l);
    std::ifstream f(fileName, std::ios::in |
std::ios::binary);
    if (!f.is_open())
```

```
    {
        std::cerr << "Не получается открыть файл: " <<
std::endl;
        return;
    }
    while (true) {
        AirRace* ar = new AirRace;
        f.read((char*)(ar), sizeof(AirRace));
        if (!f)
        {
            delete ar;
            break;
        }
        Node* node = new Node();
        node->data = ar;
        AddIntoBegin2(l, node);
    }
    Sort(l, 1, 0);
    f.close();
}
void Load2(List* l, std::string filename) {
    std::ifstream f(filename);
    if (!f.is_open()) {
        std::cerr << "Не получается отрыть файл " << filename
<< std::endl;
        return;
    }
    Clear(l);
    while (!f.eof()) {
        std::string path1, path2, planeMark;
        float onePassageerCost, totalRaceCost;
        int raceNum, passageersAmmount;
        f >> path1 >> path2 >> planeMark >> onePassageerCost
>> raceNum >> totalRaceCost >> passageersAmmount;
        if (!f.fail()) {
            Node* ar = new Node();
            ar->data = new AirRace();
                ar->data->path1 = path1;
                ar->data->path2 = path2;
                ar->data->planeMark = planeMark;
                ar->data->onePassageerCost                    =
onePassageerCost;
                ar->data->raceNum = raceNum;
                ar->data->totalRaceCost = totalRaceCost;
```

```
                ar->data->passageersAmmount =
passageersAmmount;
                AddIntoBegin2(l, ar);
            }
        }
        Sort(l, 1, 0);
        f.close();
}
```

Move.h – заголовочный файл для перемещения по списку

```
#ifndef MOVE
#define MOVE
#include "List.h"
Node* Move(List* l, int index);
Node* MoveToRace(List* l, int race);
    #endif MOVE
```

Move.cpp – исходный файл для перемещения по списку

```
#include "List.h"
#include "CheckIndex.h"
//----------- return object with index -------------------
----
Node* Move(List* l, int index)
{
    if (CheckIndex(index,l->count))
    {
        Node* ar = l->begin;
        for (int i = 0; i < index; i++)
        {
            ar = ar->next;
        }
        return ar;
    }
};

Node* MoveToRace(List* l, int race)
{
    if (CheckIndex(race,l->count))
    {
        Node* ar = l->begin;
        for (int i = 0; i < l->count; i++)
        {
            if (ar->data->raceNum == race)
            {
                return ar;
            }
```

```
            ar = ar->next;
        }
        return NULL;
    }
    };
```

Organize.h – заголовочный файл для организации списка

```
#ifndef ORGANIZE
#define ORGANIZE
#include "List.h"
void Organize(List* l1);
#endif
```

Organize.cpp – исходный файл для организации списка

```
#include "AddElement.h"
#include <iostream>
void Organize(List* l1)
{
    int f1 = 0;
    do
    {
        if (l1->count == 0)
        {
            AddIntoBegin(l1);
        }
        else
        {
            AddIntoEnd(l1);
        }
        printf("Добавить ещё? 1- да 0 - нет\n");
        std::cin >> f1;
    } while (f1);
    }
```

PeredelKyrsa4.cpp – исходный файл с точка входа

```
#include "list.h"
int main()
{
    ClientCode();
    return 0;
    }
```

Reverse.h

Reverse.h – заголовочный файл для переворота списка

```
#ifndef REVERSE
#define REVERSE
#include "List.h"
void Reverse(List* l2);
#endif
```

Reverse.cpp – исходный файл для переворота списка

```
#include "List.h"
#include "AddElement.h"
void Reverse(List* l2)
{
    List* l = new List();
    Node* ar = l2->begin;
    while (ar != nullptr)
    {
        AddIntoBegin2(l,ar);
        ar = ar->next;
    }
    l2 = l;
    };
```

Save.h – заголовочный файл для сохранения списка в файл

```
#ifndef SAVE
#define SAVE
void Save1(List* l, std::string fileName);
void Save2(List* l, std::string fileName);
    #endif
```

Save.cpp – исходный файл для сохранения списка в файл

```
#include <fstream>
#include <iostream>
#include "List.h"
void Save1(List* l, std::string fileName)
{
    std::ofstream f(fileName, std::ios::out |
std::ios::binary);
    if (!f.is_open()) {
        std::cout << "Не получается отрыть файл " << fileName
<< std::endl;
        return;
    }
    Node* ar = l->begin;
    while (ar != nullptr) {
        f.write((char*)(ar->data), sizeof(AirRace));
```

```
        ar = ar->next;
    }
    f.close();
}

void Save2(List* l, std::string filename) {
    std::ofstream f(filename);
    if (!f.is_open()) {
        std::cout << "Не получается отрыть файл " << filename
<< std::endl;
        return;
    }
    Node* ar = l->begin;
    for(int i = 0;i<l->count;i++)
    {
        f << ar->data->path1 << " "
            << ar->data->path2 << " "
            << ar->data->planeMark << " "
            << ar->data->onePassageerCost << " "
            << ar->data->raceNum << " "
            << ar->data->totalRaceCost << " "
            << ar->data->passageersAmmount << std::endl;
        ar = ar->next;
    }
    f.close();
    }
```

Show.h – заголовочный файл для отображения элементов списка и финальных данных после подсчёта

```
#ifndef SHOW
#define SHOW
#include "List.h"
void Show(List* l);
void FinalShow(List* l);
#endif
```

Show.cpp – исходный файл для отображения элементов списка и финальных данных после подсчёта

```
#include "List.h"
#include <iostream>
void FinalShow(List* l)
{
    Node* ar = l->begin;
    int allCost = 0;
    int allPassageers = 0;
```

```
    float onePassageerAvgCost = 0;
    for (int i = 0; i < l->count; i++)
    {
        allCost += ar->data->totalRaceCost;
        allPassageers += ar->data->passageersAmmount;
        onePassageerAvgCost  +=  ar->data->totalRaceCost  /
ar->data->passageersAmmount;
        ar = ar->next;
    }
    onePassageerAvgCost /= l->count;
    std::cout << "Стоимсоть всех рейсов: " << allCost <<
std::endl;
    std::cout << "Количество пассажиров за всё время: " <<
allPassageers << std::endl;
    std::cout << "Средняя стоимость провозки 1 пассажира за
всё время: " << onePassageerAvgCost << std::endl;
}
void Show(List* l)
{
    int page = 0;
    int answer = 0;
    int answer2 = 0;
    int dataInPage = 10;
    bool breakflag = false;
    Node* ar = l->begin;
    do
    {
        if (breakflag)
            break;
        puts("|Номер рейса|      Название пути      |Марка
самолёта|Общие  затраты|Количество  пассажиров|Стоимость  1
пассажира|");
        for (int i = page* dataInPage; i < page* dataInPage
+ dataInPage; i++)
        {
            if (i < l->count)
            {
                printf("| %9d | %23s | %12s | %11d | %19d
| %20.2f|", ar->data->raceNum, (ar->data->path1 + "-" + ar-
>data->path2).c_str(),   ar->data->planeMark.c_str(),   ar-
>data->totalRaceCost,   ar->data->passageersAmmount,   ar-
>data->onePassageerCost);
                std::cout << std::endl;
                ar = ar->next;
            }
```

```
            else
                break;
        }
        std::cout <<"Страница: " << page <<" Введите: 1 - предыдущая страница 2 - следующая страница 3 - выбранная страница 4 - Итоговые данные 0 - выход" << std::endl;
        std::cin >> answer;
        switch (answer)
        {
        default:
            breakflag = true;
            break;
        case 1:
            if (page != 0)
            {
                page--;
                ar = l->begin;
                for (int n = 0; n < page * dataInPage; n++)
                {
                    ar = ar->prev;
                }
            }
            else
            {
                ar = l->begin;
            }
            break;
        case 2:
            if ((page + 1) * dataInPage > l->count)
            {
                if (page != 0)
                {
                    for (int n = page * 10; n < page * dataInPage; n++)
                    {
                        ar = ar->next;
                    }
                }
                else
                {
                    ar = l->begin;
                }
                break;
            }
            page++;
```

```
            break;
        case 3:
            std::cout << "Введите страницу" << std::endl;
            std::cin >> answer2;
            int divide;
            if (page * dataInPage == 0)
                divide = 1;
            else
                divide = page * dataInPage;
            if (answer2 >= l->count / divide)
            {
                std::cout << "Неверная страница" << std::endl;
                if (page != 0)
                {
                    for (int n = 0; n < dataInPage; n++)
                    {
                        ar = ar->prev;
                    }
                }
                else
                {
                    ar = l->begin;
                }
            }
            else
            {
                page = answer2;
                ar = l->begin;
                if (page != 0)
                {
                    for (int n = page*10; n < page * dataInPage; n++)
                    {
                        ar = ar->next;
                    }
                }
            }
            break;
        case 4:
            FinalShow(l);
            page = 0;
            ar = l->begin;
            break;
        }
```

```
    } while (true);
    };
```

Sort.h – заголовочный файл для сортировки списка по любому ключу

```
#ifndef SORT
#define SORT
void Sort(List* l, int keyType, bool isReverse);
    #endif
```

Sort.cpp – исходный файл для сортировки списка по любому ключу

```
#include "List.h"
#include "Reverse.h"
#include <iostream>
void Sort(List* l, int keyType, bool isReverse)
{
    Node* ar = l->begin;
    Node* arc = l->begin;
    switch (keyType)
    {
    case 1:
        for (int i = 0; i < l->count; i++)
        {
            ar = arc;
            for (int j = i; j < l->count; j++)
            {
                if    (ar->data->raceNum    >    arc->data-
>raceNum)
                {
                    std::swap(ar->data, arc->data);
                }
                ar = ar->next;
            }
            arc = arc->next;
        }
        break;
    case 2:
        for (int i = 0; i < l->count; i++)
        {
            ar = arc;
            for (int j = i; j < l->count; j++)
            {
                if    (strcmp(ar->data->planeMark.c_str(),
arc->data->planeMark.c_str())>0)
                {
                    std::swap(ar->data, arc->data);
```

```
                }
                ar = ar->next;
            }
            arc = arc->next;
        }
        break;
    case 3:
        for (int i = 0; i < l->count; i++)
        {
            ar = arc;
            for (int j = i; j < l->count; j++)
            {
                if (strcmp(ar->data->path1.c_str(), arc->data->path1.c_str()) > 0)
                {
                    std::swap(ar->data, arc->data);
                }
                ar = ar->next;
            }
            arc = arc->next;
        }
        break;
        break;
    case 4:
        for (int i = 0; i < l->count; i++)
        {
            ar = arc;
            for (int j = i; j < l->count; j++)
            {
                if (ar->data->totalRaceCost > arc->data->totalRaceCost)
                {
                    std::swap(ar->data, arc->data);
                }
                ar = ar->next;
            }
            arc = arc->next;
        }
        break;
    case 5:
        for (int i = 0; i < l->count; i++)
        {
            ar = arc;
            for (int j = i; j < l->count; j++)
            {
```

```
                    if   (ar->data->passageersAmmount   >   arc-
>data->passageersAmmount)
                    {
                        std::swap(ar->data, arc->data);
                    }
                    ar = ar->next;
                }
                arc = arc->next;
            }
            break;
        case 6:
            for (int i = 0; i < l->count; i++)
            {
                ar = arc;
                for (int j = i; j < l->count; j++)
                {
                    if   (ar->data->onePassageerCost   >   arc-
>data->onePassageerCost)
                    {
                        std::swap(ar->data, arc->data);
                    }
                    ar = ar->next;
                }
                arc = arc->next;
            }
            break;
        default:
            std::cout << "Wrong input data" << std::endl;
            break;
        }
        if (!isReverse)
        {
            Reverse(l);
        }
        };
```