|  |  |
| --- | --- |
|  | |
| МИНОБРНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА - Российский технологический университет»**  **РТУ МИРЭА** | |
| Институт комплексной безопасности и специального приборостроения | |
| Кафедра *«*Аппаратного, программного и математического обеспечения вычислительных систем» | |

|  |  |
| --- | --- |
| **КУРСОВОЙ ПРОЕКТ (РАБОТА)**  **по дисциплине** | |
| **«** Прикладное программирование программного обеспечения **»**  **Тема курсового проекта (работы)** «Разработка программы для хранения связанной информации с использованием динамических структур данных» | |
| Студент группы \_\_\_БСБО-13-18\_\_\_  *(учебная группа)* |  |
| Руководитель курсового проекта (работы)  *должность, звание, ученая степень* |  |
| Рецензент |  |

|  |  |  |
| --- | --- | --- |
| Работа представлена к защите | «\_\_»\_\_\_\_\_\_\_\_\_\_201\_\_ г. |  |
|  |  |  |
| «Допущен к защите» | «\_\_»\_\_\_\_\_\_\_\_\_\_201\_\_ г. | *(* |

Москва 2019

|  |  |
| --- | --- |
| для прик эмбл | |
| МИНОБРНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"Московский технологический университет"**  **МИРЭА** | |
| Институт информационных технологий | |
| Кафедра название кафедры (полностью) | |
|  | **Утверждаю** |
|  | Заведующий кафедрой \_\_Михайлов Б.М\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_  *Подпись ФИО* |
|  | «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_201\_ г. |
| **ЗАДАНИЕ** | |
| **на выполнение курсового проекта (работы)** по дисциплине | |
| « Прикладное программирование программного обеспечения » | |

**Студент:** Шагаалан Бадый Чочагаевич Группа БСБО-13-18

**Тема:** «Разработка программы для хранения связанной информации с использованием

динамических структур данных»

**Исходные данные: \_**пункт отправления, пункт назначения,**\_**номер рейса, Ф.И.О, дата вылета**,** время вылета**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Перечень вопросов, подлежащих разработке, и обязательного графического материала:** Разобрать и сравнить методологии программирования, выбрать среду разработки, обосновать выбор среды разработки, описать способ работы с файлами и описать объекты предметной области, спроектировать структурную часть программы, диаграммы классов, разработать структурные схемы алгоритмов, разработать программу

|  |
| --- |
| **Срок представления к защите курсового проекта (работы):** **до** «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 201\_\_ г. |
|  |
| **Задание на выполнение курсовой проект (работу) выдал \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_**Крехов Е.В **)**  *Подпись руководителя Ф.И.О. руководителя* |
| «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_г.  **Задание на курсовой проект (работу) получил** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Шагаалан Б.Ч. \_)  *Подпись обучающегося Ф.И.О. исполнителя* |

Оглавление.

Введение4

**1.ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ.5**

**1.1 Постановка задачи** 5

**1.2 Краткие сведения из теории** ……………………………………………………………………………………………………..6

**1.2.1 Классификация методологий программирования** …………………………………………………...…7

**1.2.2 Сравнительный анализ методологий программирования** …………………………………………8

**1.3 Анализ инструментов разработки** …………………………………………………………………………………………10

**1.3.1 Анализ инструментальной среды** ……………………………………………………………………………….10

**1.3.2 Обоснование способа работы с файлами** …………………………………………………………………..11

**1.3.3 Описание объектов предметной области** …………………………………………………………………..12

**1.3.4 Описание способа обработки объектов предметной области** ………………………………….13

**2. ПРАКТИЧЕСКАЯ ЧАСТЬ………………………………………………………………………………………………………………..16**

**2.1 Проектирование структурной схемы программы** …………………………………………………………………16

**2.2 Проектирование диаграммы классов** …………………………………………………………………………………….17

**2.3 Разработка структурных схем алгоритмов** …………………………………………………………………………….18

**2.4 Разработка программы. Листинг** …………………………………………………………………………………………….26

**3. Результаты работы программы** ………………………………………………………………………………………………….. 41

**4. Заключение** ……………………………………………………………………………………………………………………………………44

**Источники**……………………………………………………………………………………………………………………………………………45

# ВВЕДЕНИЕ

На сегодняшний день прикладная программа должна иметь хорошо продуманную структуру хранения данных, подобранную специально под задачи программного средства. Ее неправильный выбор может привести к потере в производительности т.е. программа будет функционировать не эффективно по отношению к времени работы и использованию памяти.

Объектом исследования является прикладная программа моделирующая объекты предметной области.

Предметом исследования является методология объектно-ориентированного программирования.

Цель курсовой работы: разработка прикладной программы учета заявок на авиабилеты, а также получение практических навыков в создании приложений с использованием высокоуровневых методов программирования в среде Microsoft Visual Studio на языке C++.

Задачи:

* рассмотреть и сравнить императивную и декларативную методологии программирования.
* Проанализировать различные инструментальные среды разработки (IDE)
* Разработать прикладную программу на языке С++ используя контейнерный класс.
* Разработать контейнерный класс для прикладной программы, используя объектно-ориентированный подход.

# 1. ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ.

1.1 Постановка задачи

Разработать программу в среде разработки Microsoft Visual Studio на языке высокого уровня C++.

Программа должна реализовывать средство для учета заявок на авиабилеты. Каждая заявка содержит: пункт назначения, номер рейса , фамилию, имя и отчество пассажира, а также желаемую дату вылета. Программа должна обеспечивать выбор с помощью меню и выполнение следующих функций:

* Добавление заявок в список;
* Удаление заявок;
* Вывод заявок по заданному номеру рейса и дате вылета;
* Вывод всех заявок;

Для хранения данных необходимо реализовать и использовать контейнерный класс list.

1.2 Краткие сведения из теории

1.2.1 Классификация методологий программирования

Методологии программирования можно разделить на два больших блока:

- императивная методология;

- декларативная методология.

Эти два блока в свою очередь делятся еще на более мелкие блоки, но подробно мы рассмотрим только эти два.

Императивная методология программирования описывает процесс вычислений посредством описания управляющей логики программы, т.е. в виде последовательности отдельных команд, которые должен выполнить компьютер. Каждая команда является инструкцией, изменяющей состояние программы. Программа, написанная в императивном стиле, похожа на набор приказов, выражаемых повелительным наклонением в естественных языках, то есть последовательность команд, которые должен выполнить компьютер.

Для императивной парадигмы характерны:

* в исходном коде программы записываются инструкции (команды);
* инструкции должны выполняться последовательно;
* при выполнении инструкции данные, полученные при выполнении предыдущих инструкций, могут читаться из памяти;
* данные, полученные при выполнении инструкции, могут записываться в память.

При императивном подходе к составлению кода (в отличие от функционального подхода, относящегося к декларативной парадигме) широко используется присваивание. Наличие операторов присваивания увеличивает сложность модели вычислений и делает императивные программы подверженными специфическим ошибкам, не встречающимся при функциональном подходе.

Основные черты императивныx языков:

* использование именованных переменных;
* использование оператора присваивания;
* использование составных выражений;
* использование подпрограмм;
* и др.

Императивная методология делится на:

* процедурную
* структурную
* ООП
* И др.

Императивные языки программирования это:

* С#
* C++
* C
* FORTRAN
* Pascal
* Python
* Java
* Язык ассемблера

Декларативное методология программирование — это методология программирования, в которой задаётся спецификация решения задачи, то есть описывается, что представляет собой проблема и ожидаемый результат. Противоположностью декларативного является императивное программирование, описывающее на том или ином уровне детализации, как решить задачу и представить результат. В общем и целом, декларативное программирование идёт от человека к машине, тогда как императивное — от машины к человеку. Как следствие, декларативные программы не используют понятия состояния, то есть не содержат переменных и операторов присваивания.

Декларативный подход к разработке компьютерных программ появился в начале 70-х годов. Он не получил столь широкого применения как процедурный, поскольку был направлен на относительно узкий круг задач искусственного интеллекта. При его применении программист описывает свойства исходных данных, их взаимосвязи, свойства, которыми должен обладать результат, а не алгоритм получения результата. Разумеется, для получения результата этот алгоритм все равно нужен, но он должен порождаться автоматически той системой, которая поддерживает декларативно-ориентированный язык программирования. При логическом варианте такого подхода (прежде всего это относится к языку Пролог, PROLOG) задача описывается совокупностью фактов и правил в некотором формальном логическом языке, при функциональном варианте – в виде функциональных соотношений между фактами (язык Лисп, LISP). К подвидам декларативного программирования также зачастую относят функциональное и логическое программирование — несмотря на то, что программы на таких языках нередко содержат алгоритмические составляющие, архитектура в императивном понимании (как нечто отдельное от кодирования) в них также отсутствует: схема программы является непосредственно частью исполняемого кода. Примеры декларативных языков программирования:

* Prolog
* SQL
* Agda
* FP
* Mercury
* Picat
* И др.

1.2.2 Сравнительный анализ методологий программирования

При написании программы используются разные методологии программирования. Каждая из этих методологий максимально оптимизирует процесс написания кода при постановке той или иной задачи.

На мой взгляд, разница в том что, декларативное программирование задаётся спецификациями решения задач, то есть описывает, что представляет собой проблема и ожидаемый результат. А императивное программирование, описывающее на том или ином уровне детализации, как решить задачу и представить результат. Как следствие, декларативные программы не используют понятия состояния, то есть не содержат переменных и операторов присваивания. А так же к отличиям можно добавить то, что императивные программы, как правило, длиннее декларативных, а также декларативный стиль требует рекурсии, а императивный может без нее обойтись.

Таким образом, декларативное программирование - это когда в коде описано что должно получиться, то есть программиста не интересует каким образом машина сделает работу, какие инструкции в каком порядке выполняются команды, то есть мы просто хотим увидеть результат. Программируя же императивно, мы описываем конкретные шаги, действия и точный порядок.

Свои преимущества и недостатки имеет каждая из методологий. Плюсы декларативного программирования:

1. Более простой в сравнении с императивным подход к получению нужного результата;
2. Читаемость кода и ясность намерений программиста намного выше, чем в императивном;
3. Программирование осуществляется в терминах предметной области.

Минусы декларативного программирования:

1. Декларативный подход предполагает, что машина знает все необходимые шаги, чтобы добраться до нужного результата, т.е. если она вдруг их не знает, то программист сталкивается с проблемой;
2. Накладные расходы при реализации куда более существеннее, нежели у программ на императивных языках.

Плюсы императивного программирования:

1. При использовании императивного подхода можно решать более широкий спектр задач;
2. В большинстве случаев оптимизация и быстрота императивных решений намного выше;
3. Накладные расходы минимальны.

Минусы императивного программирования:

1. Читаемость кода осложнена, в сравнении с программами написанными декларативными языками;

Я считаю, что для создания прикладной программы лучше всего подходит императивная методология программирования. А именно объектно-ориентированный подход, потому что он позволяет нам работать с абстрактными классами моделирующими различные объекты из реального мира.

1.3 Анализ инструментов разработки

1.3.1 Анализ инструментальной среды разработки (IDE)

Для разработки программы на языке С++ нужно иметь на своем ПК установ-ленную среду разработки. Таких IDE существует много, но если рассматри-вать IDE под операционную систему Windows, то можно выделить следующие наиболее популярные IDE: Microsoft Visual Studio, Eclipse, NetBeans. Рассмот-рим их более подробно.

Microsoft Visual Studio

Microsoft Visual Studio — линейка продуктов компании Microsoft, включаю-щих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данная обладает огромным функциона-лом, например, в ней можно разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, а также и веб-сайты, веб-приложе-ния и прочее. Visual Studio приходится по душе многим разработчикам своим удобством и богатым функционалом. IDE поддерживает следующие языки программирования: Ajax, ASP.NET, DHTML, JavaScript, Visual Basic, Visual C#, Visual C++, Visual F#, XAML, Python и другие. Распространяется условно-бесплатно.

CLion

Это отличная среда разработки на C++, созданная известной компанией JetBrains. Она поставляется с некоторыми замечательными функциями, та-кими как “Smart Editor”, “Code Analysis”, “Embedded Terminal”. CLion имеет возможности работы с визуализированными интерфейсами, установки различ-ных плагинов, а также наличия удобных инструментов для написания кода и его отладки. Данная IDE платная, не имеющая бесплатной версии, однако вы можете получить пробную версию в течение 30 дней, чтобы протестировать ее.

NetBeans

NetBeans является одной из лучших IDE для разработки программ на C++. В ней есть много готовых шаблонных проектов для C и C++, которые можно ис-пользовать в качестве основы для ваших приложений. Из плюсов можно вы-делить возможность создавать приложения с графическим интерфейсом, мульти платформенную поддержку, поддержку Qt Toolkit. Помимо С++ под-держиваются такие языки: Java, Python, PHP, JavaScript, C, C++ и другие. Net-Beans распространяется бесплатно.

1.3.2 Обоснование способа работы с файлами

Как и в большинстве языков программирования в C++ есть возможность работы с файлами, для этого нужно подключить соответствующую библиотеку fstream. Для записи данных в файл нужно использовать класс ofstream, а для чтения данных из файла – класс ifstream. В ходе работы с файлами следует соблюдать следующий порядок действий:

1. Создать экземпляр соответствующего класса на чтение или запись
2. Открыть файл. Сделать это можно еще при создании экземпляра класса следующим образом (на примере класса следующим образом (на примере класса ifstream) :

ifstream fin(“название файла”);

Или же с помощью метода open:

ifstream fin;

fin.open(“название файла”);

1. Записать или считать данные из файла с помощью методов write(), read() соответственно или с помощью операторов потокового ввода/вывода <<,>> соответственно.
2. Закрыть файл с помощью метода close().

fin.close();

Хранение данных в файле, можно организовать несколькими способами: в текстовом или бинарном виде. Данные в текстовом виде записаны в файле на языке, понятному для человека, а данных в бинарном виде записываются в виде последовательностей нулей и единиц.

В бинарных файлах для записи/чтения можно использовать только методы write() и read():

fin.read(char\*)&current, sizeof(Person))

fout.write((char\*)&p, sizeof(p));

Для работы с файлами в своей программы я выбрал хранение данных в текстовом виде для того, чтобы пользователю, в случае нужды, было удобно работать с данными непосредственно в самом файле формата txt без использования самой программы.

1.3.3 Описание объектов предметной области

Для работы программы используется пять классов: List, List:: Node, Stack, Stack:: Node, request, my\_string. Рассмотрим их более подробно, какие поля и методы они содержат.

Класс request содержит информацию о заявке пользователя на перелет. В нем объявлены следующие поля: departure – информация о том из какого города вылет рейса; destination – информация о том в какой город прибывает рейс; flight\_number – информация о том какой номер у рейса, surname – фамилия пассажира, name – имя пассажира, patronymic – отчество пассажира; departure\_date – желаемая дата вылета, departure\_time – желаемое время вылета. Все перечисленные выше поля отмечены модификатором доступа private. К полям с модификатором доступа private мы не можем получить данные через экземпляр класса. Класс request содержит в себе два конструктора: один по умолчанию, другой принимает информацию, которая записывается в вышеперечисленные поля класса. Также этот класс содержит деструктор удаляющий все поля класса при его вызове. Функции get\_date() и get\_flight\_n() возвращают информацию о дате вылета и номере рейса.

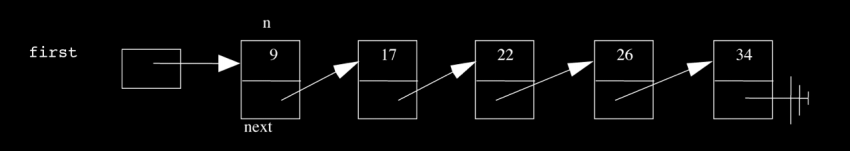
Класс List используется для реализации контейнерного класса – связный список. В моем случае, для списка необходимо создать вложенный класс node для того, чтобы хранить в нем данные и строить связь с остальными узлами. Класс node содержит экземпляр класса request и два указателя: на предыдущий и следующий узел. Класс List содержит указатель на первый и последний узел. Все перечисленные выше поля отмечены модификатором доступа private. В классе List реализованы функции: Privat функция find\_by\_index для получения доступа к узлу node внутри класса List; открытые конструктор и деструктор; открытая функция добавления add() принимающая указатель на экземпляр класса request и добавляющая его в список; функция empty проверяющая пуст ли список; функция size возвращающая размер списка; функция delete\_by\_index(int i) удаляющая узел списка по индексу; insert\_by\_index(int i, request\* key) – вставка нового узла в список; swap(int i, int j) – функция перестановки двух узлов списка по индексам. Также для класса List переопределен оператор [], который принимает индекс узла и возвращает значение request\*.

Класс my\_string используется в этой программе для более удобной работы с текстовой информацией. Поля класса: char\* \_ch – указатель на массив символов; int \_length – количество символов в \_ch. Все перечисленные выше поля отмечены модификатором доступа private. Класс my\_string содержит в себе три конструктора: первый по умолчанию, второй принимает информацию, которая записывается поле \_ch, третий аналогичен второму с отличием, что он принимает константный указатель на строку. Для этого класса также определен деструктор удаляющий поля класса при вызове. Функция Length возвращает значение поля \_length. Функция change\_ch принимает указатель на массив символов и меняет старый \_ch на новый. get\_ch возвращает константный указатель на поле \_ch. В классе my\_string я перегрузил оператор ==, >>, <<, =.

Класс Stack я использую для реализации контейнерного класса – стек. Стек мне нужен для не рекурсивной быстрой сортировки половинным делением. В моем случае, для списка необходимо создать вложенный класс node для того, чтобы хранить в нем данные и строить связь с остальными узлами. Класс node содержит поле key хранящее целое число и указатель next на следующий узел. Класс Stack содержит поле указатель на первый узел head. В нем есть конструктор и деструктор. Функция push(int key) создает новый узел, добавляет значение в узел и добавляет его конец стека.

1.3.4 Описание способа обработки объектов предметной области

**Связный список** (linked list) — это структура данных, в которой элементы линейно упорядочены, но порядок определяется не номерами элементов (как в массивах), а указателями, входящих в состав элементов списка и указывают на следующий элемент. У списка должна быть «голова» (первый элемент) и «хвост» (последний элемент).



## Преимущества и недостатки по сравнению с массивами:

В отличие от массивов, вставка и удаление элементов в Связный список производится за постоянное время (**О (1)**). Также значительным преимуществом связных списков является возможность легкого расширения: чтобы увеличить размер списка, нужно только добавить еще один элемент.

Недостатком связных списков необходимость проходить весь список, чтобы найти элемент (то есть время доступа к элементу списка = **О (n)**).

### Элементы связанного списка называют узлами. Каждый узел содержит данные, которые мы хотим сохранить, а также указатель на следующий элемент в списке или NULL, если этот элемент является последним.

Инициализация ОЛС:

* Инициализация списка предназначена для создания корневого узла списка, у которого поле указателя на следующий элемент содержит нулевое значение.
* Добавление узла в ОЛС
* Функция добавления узла в список принимает два аргумента:
* Указатель на узел, после которого происходит добавление
* Данные для добавляемого узла.

Добавление узла в ОЛС включает в себя следующие этапы:

* создание добавляемого узла и заполнение его поля данных;
* переустановка указателя узла, предшествующего добавляемому, на добавляемый узел;
* установка указателя добавляемого узла на следующий узел (тот, на который указывал предшествующий узел).
* Удаление узла ОЛС
* В качестве аргументов функции удаления элемента ОЛС передаются указатель на удаляемый узел, а также указатель на корень списка.
* Функция возвращает указатель на узел, следующий за удаляемым.

Удаление узла ОЛС включает в себя следующие этапы:

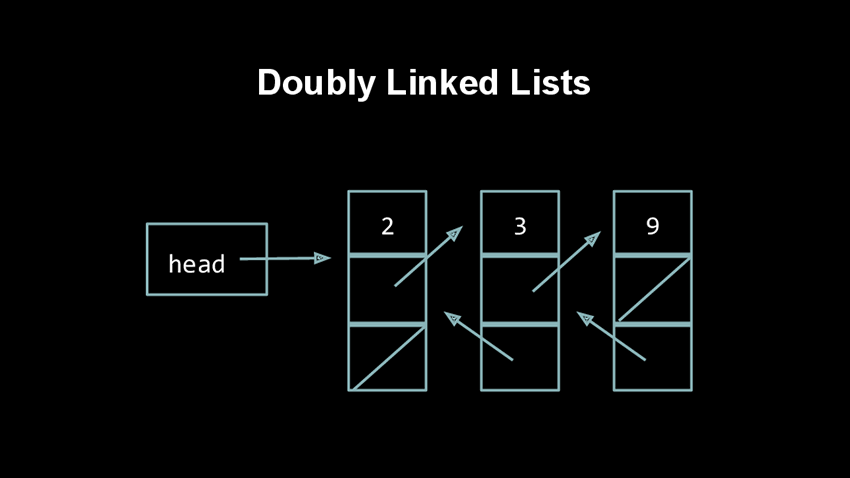
* установка указателя предыдущего узла на узел, следующий за удаляемым;
* освобождение памяти удаляемого узла.

Вывод элементов списка:

* В качестве аргумента в функцию вывода элементов передается указатель на корень списка.
* Функция осуществляет последовательный обход всех узлов с выводом их значений.

**Двусвязный список**

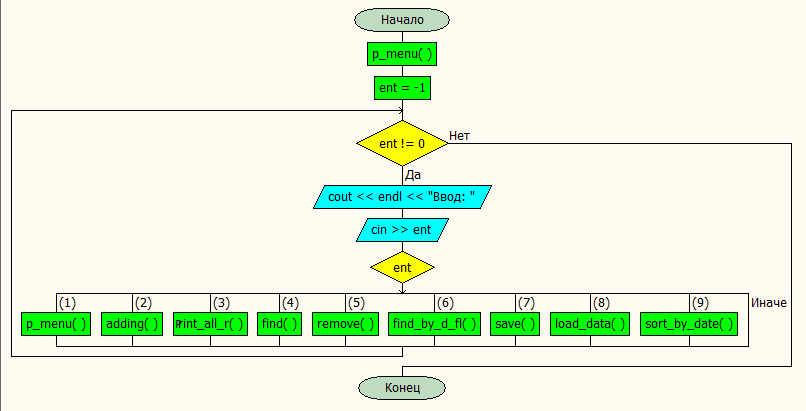
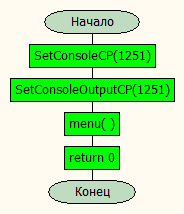
Двусвязный список похож на обычный связный список, только элементы в нем хранят ссылки не только на следующий, но и на предыдущий элемент. Благодаря этому свойству, можно перемещаться по списку вперед и назад.



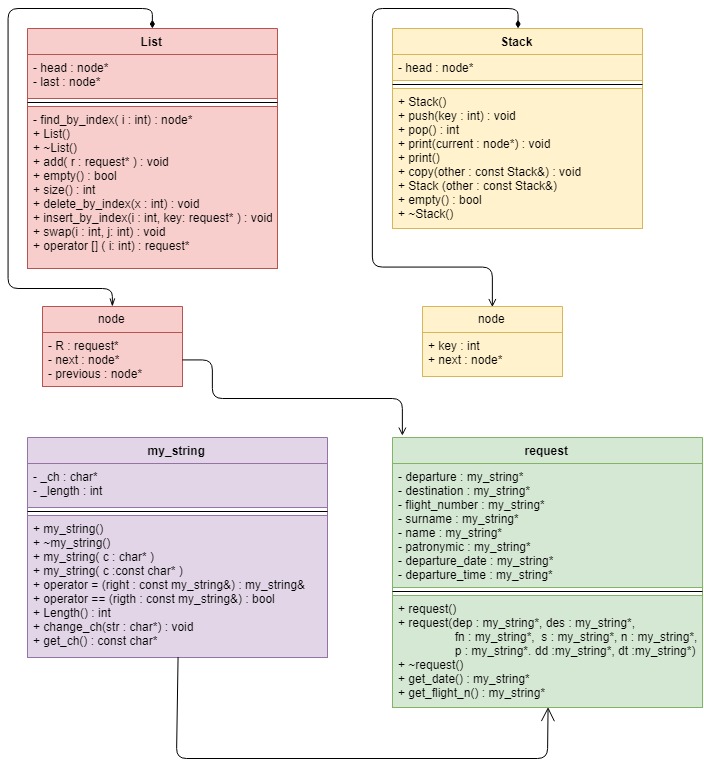
Элемент двусвязного списка содержит данные и два поля: prev и next, указатели на предыдущий и следующий элементы списка соответственно.

# 2. Практическая часть.

2.1. Проектирование структурной схемы программы

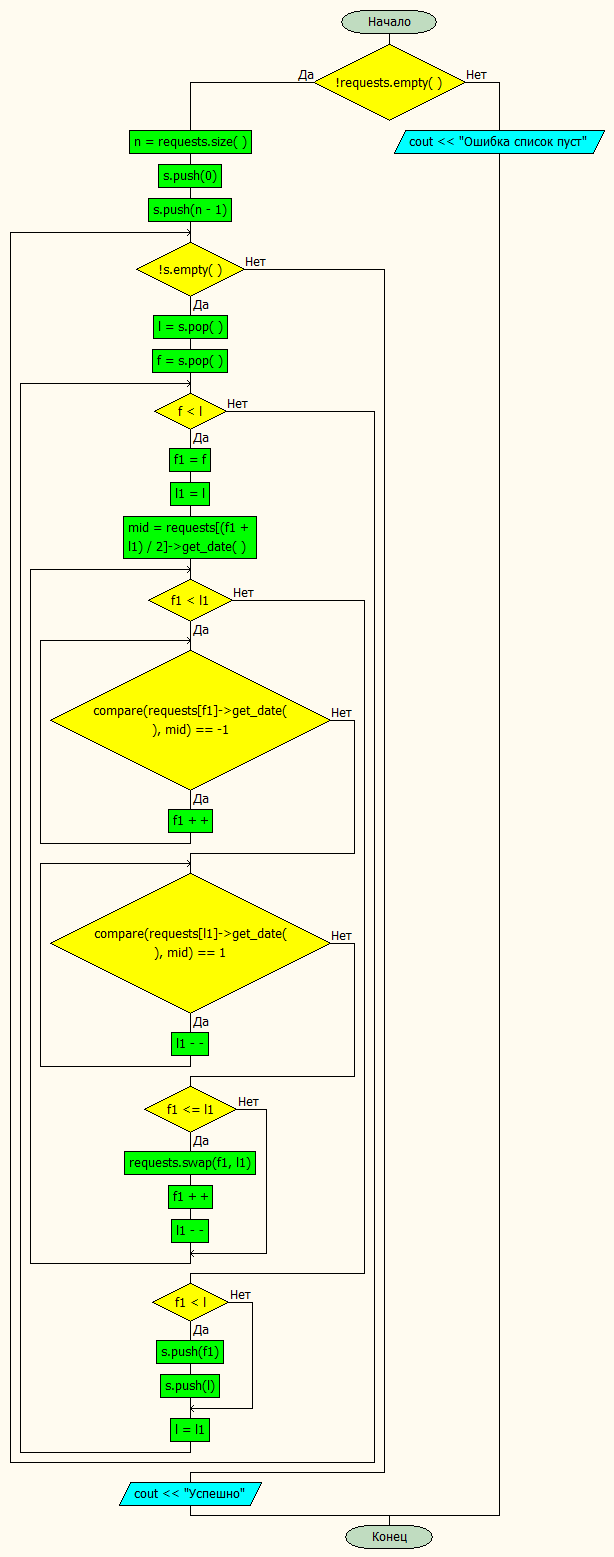


2.2 Проектирование диаграмм классов

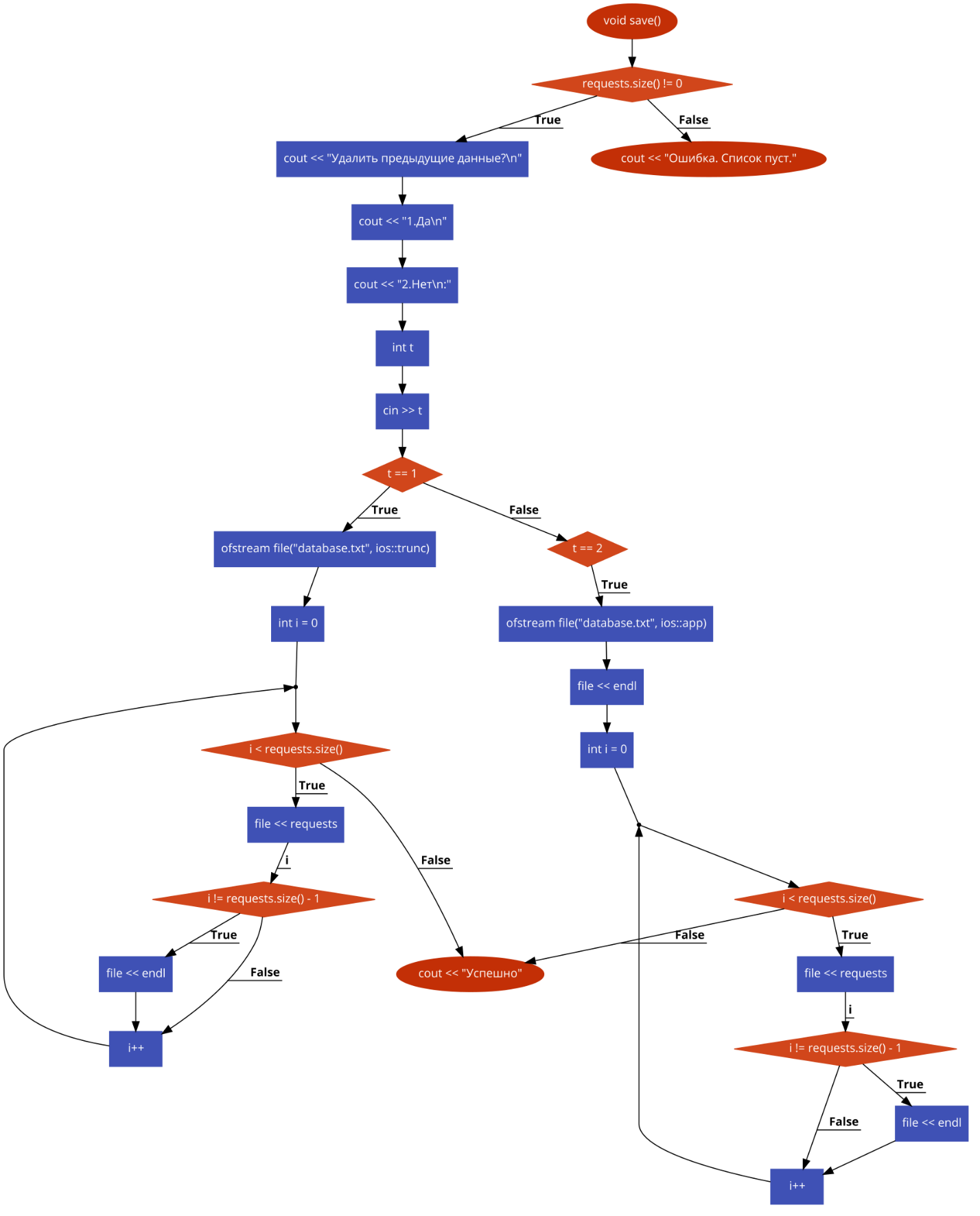


2.3 Разработка структурных схем

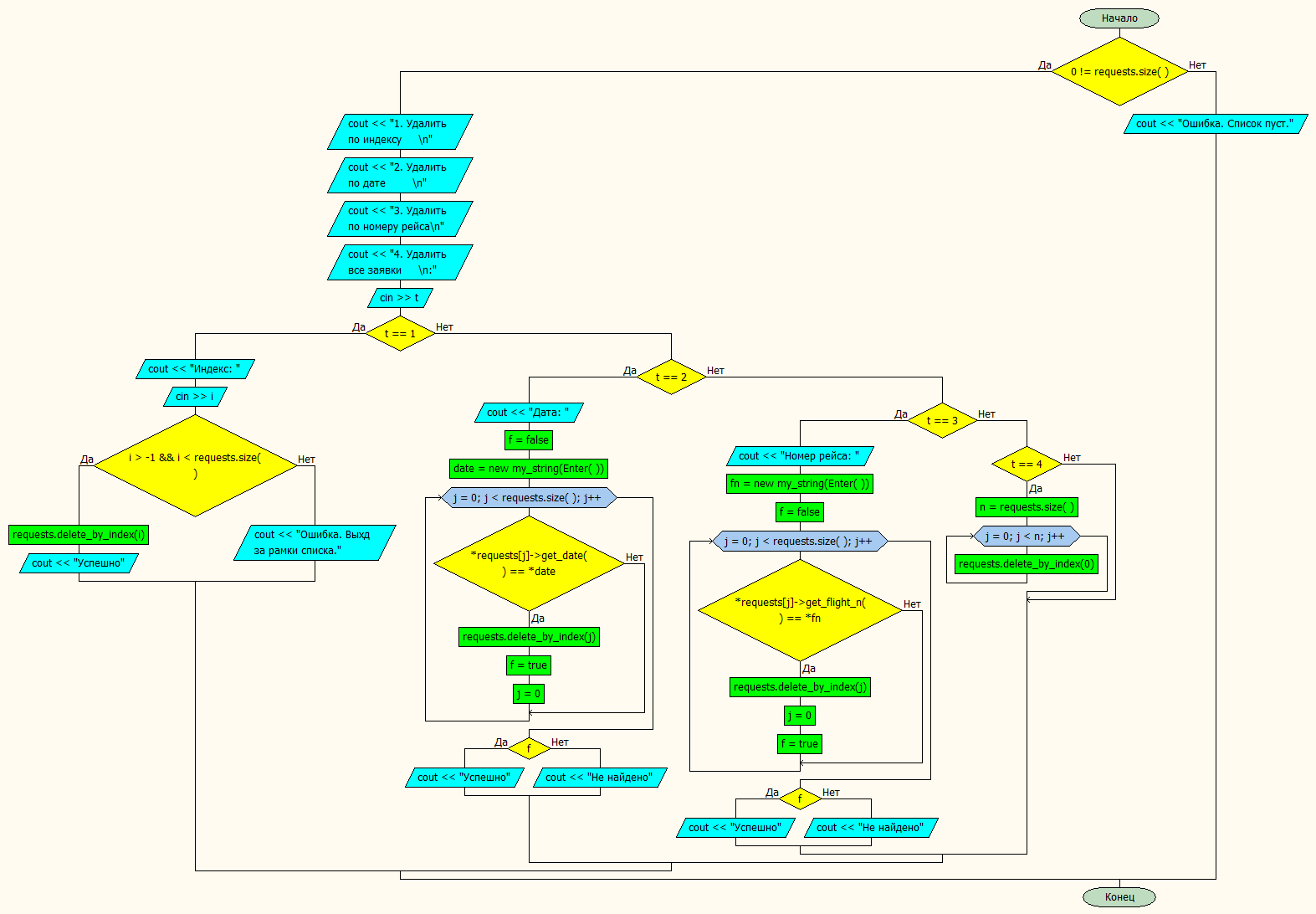
Алгоритм не рекурсивной быстрой сортировки, использованный в программе



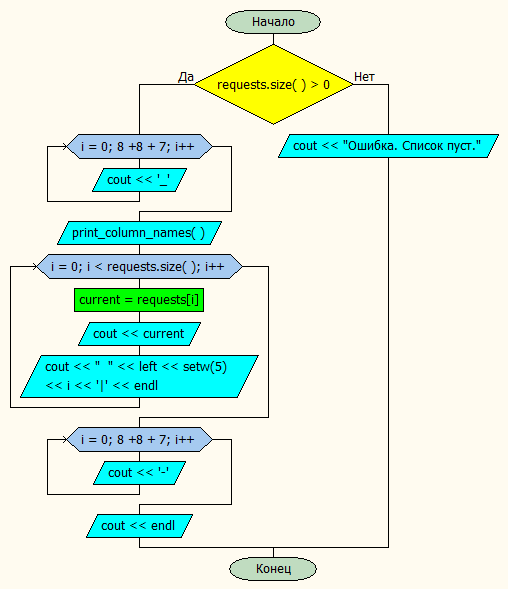
Save()



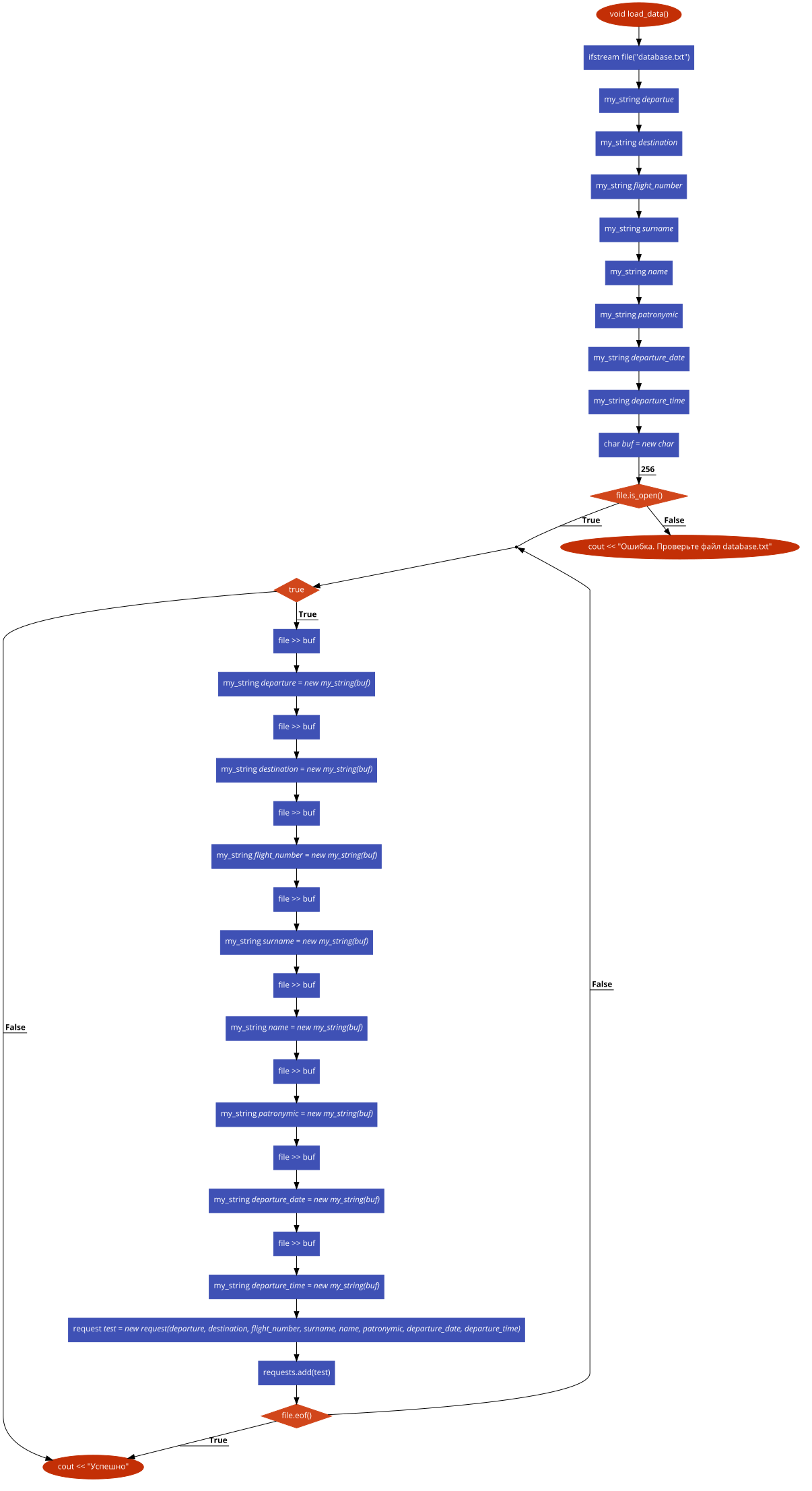
Remove()



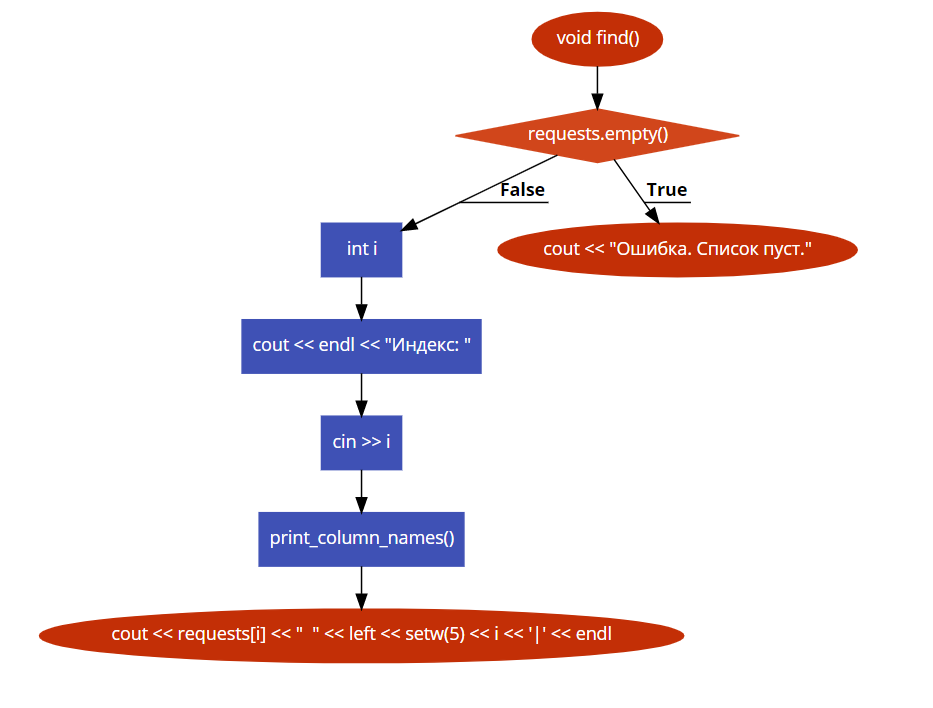
print\_all\_r()



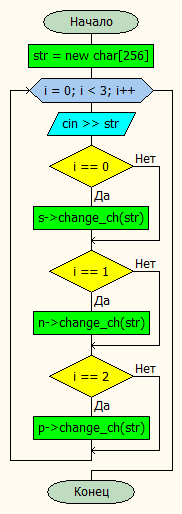
load\_data()\_



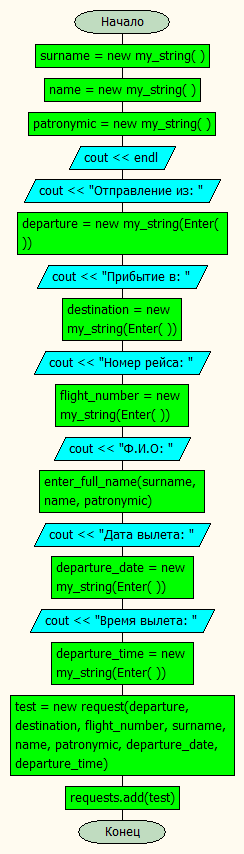
Find()



enter\_full\_name()



adding()



2.4. Разработка программы. Листинг.

#include "pch.h"

#include <Windows.h>

#include <iostream>

#include <iomanip>

#include <fstream>

void print\_all\_r();

using namespace std;

int max\_len = 13;

class my\_string {

private:

char\* \_ch;

int \_length;

public:

my\_string() {

\_ch = nullptr;

\_length = 0;

}

~my\_string() {

char\* current = \_ch;

delete current;

}

my\_string(char\* c) {

\_length = strlen(c);

if (\_length > max\_len)

max\_len = \_length;

\_ch = new char[\_length + 1];

strcpy\_s(\_ch, \_length + 1, c);

}

my\_string(const char\* c) {

\_length = strlen(c);

if (\_length > max\_len)

max\_len = \_length;

\_ch = new char[\_length + 1];

strcpy\_s(\_ch, \_length + 1, c);

}

int Length() {

return \_length;

}

void change\_ch(char\* str) {

\_length = strlen(str);

delete[] \_ch;

\_ch = new char[\_length + 1];

strcpy\_s(\_ch, \_length + 1, str);

if (\_length > max\_len)

max\_len = \_length;

}

const char\* get\_ch() {

return \_ch;

}

bool operator ==(const my\_string& right) {

if (this->\_length == right.\_length) {

for (int i = 0; i < this->\_length; i++) {

if (\_ch[i] != right.\_ch[i]) {

return false;

}

}

}

else

return false;

return true;

}

my\_string& operator = (const my\_string& right) {

if (this == &right) {

return \*this;

}

if (\_length != right.\_length || \_length == 0) {

delete[] \_ch;

\_length = right.\_length;

\_ch = new char[\_length + 1];

}

strcpy\_s(\_ch, \_length, right.\_ch);

if (\_length > max\_len)

max\_len = \_length;

}

friend ostream& operator << (ostream &os, const my\_string& current) {

os << current.\_ch;

return os;

}

friend ostream& operator << (ostream &os, my\_string\* current) { // cout

os << current->\_ch;

return os;

}

friend istream& operator >> (istream &in, my\_string& current) {

current.\_ch = new char();

in >> current.\_ch;

current.\_length = strlen(current.\_ch);

return in;

}

friend istream& operator >> (istream &in, my\_string\* current) {

current->\_ch = new char[256];

in >> current->\_ch;

current->\_length = strlen(current->\_ch);

return in;

}

};

class request {

private:

my\_string\* departure;

my\_string\* destination;

my\_string\* flight\_number;

my\_string\* surname;

my\_string\* name;

my\_string\* patronymic;

my\_string\* departure\_date;

my\_string\* departure\_time;

public:

request() {

departure = nullptr;

destination = nullptr;

flight\_number = nullptr;

surname = nullptr;

name = nullptr;

patronymic = nullptr;

departure\_date = nullptr;

departure\_time = nullptr;

}

request(my\_string\* dep, my\_string\* des, my\_string\* fn, my\_string\* s, my\_string\* n, my\_string\* p, my\_string\* dd, my\_string\* dt) {

departure = dep;

destination = des;

flight\_number= fn;

surname = s;

name = n;

patronymic = p;

departure\_date = dd;

departure\_time = dt;

}

~request() {

delete departure;

delete destination;

delete flight\_number;

delete surname;

delete name;

delete patronymic;

delete departure\_time;

delete departure\_date;

}

my\_string\* get\_date() {

return departure\_date;

}

my\_string\* get\_flight\_n() {

return flight\_number;

}

friend ostream& operator << (ostream &os, request\* current) {

os << ' ' << left << setw(max\_len + 1) << current->departure << ' ';

os << setw(max\_len + 1) << current->destination << ' ';

os << setw(max\_len + 1) << current->flight\_number << ' ' << setw(max\_len + 1) << current->surname << ' ';

os << setw(max\_len + 1) << current->name << ' ' << setw(max\_len + 1) << current->patronymic << ' ';

os << setw(max\_len) << current->departure\_date << setw(max\_len - 6) << right << current->departure\_time;

return os;

}

};

class Stack {

private:

class node {

friend Stack;

int key;

node\* next;

};

node\* head;

public:

Stack() {

head = nullptr;

}

void push(int key) {

node\* t = new node();

if (head)

t->next = head;

head = t;

head->key = key;

}

int pop() {

if (head) {

node\* buf = head;

int t = head->key;

head = head->next;

delete(buf);

return t;

}

else {

cout << "ERROR";

return -1;

}

}

void print(node\* current) {

cout << current->key << endl;

if (current->next) {

print(current->next);

}

}

void print() {

if (head)

print(head);

else

cout << "ERROR. Stack is empty";

}

void copy(const Stack& other) {

if (other.head != nullptr) {

if (other.head->next)

copy((const Stack&)other.head->next);

push(other.head->key);

}

}

Stack(const Stack& other) {

node\* head;

copy(other);

}

bool empty() {

return head == nullptr;

}

~Stack() {

node\* buf = head;

if (head != nullptr)

while (buf->next) {

buf = head->next;

delete[] buf;

}

delete[] head;

}

};

class List {

class node {

friend List;

request\* R;

node\* next;

node\* previous;

};

node\* head;

node\* last;

node\* find\_by\_index(int i) {

node\* buf = head;

if (i >= 0) {

int j = 0;

while (j++ < i) {

buf = buf->next;

}

return buf;

}

cout << "Ошибка";

return nullptr;

}

public:

List() {

head = nullptr;

last = nullptr;

}

~List() {

delete head;

delete last;

}

void add(request\* r) {

node\* buf = new node();

buf->R = r;

if (head == nullptr)

head = buf;

if (last != nullptr) {

last->next = buf;

last->next->previous = last;

}

last = buf;

}

bool empty() {

return head == nullptr;

}

int size() {

node\* current = head;

if (current != nullptr) {

int counter = 0;

while (current != nullptr) {

counter++;

current = current->next;

}

return counter;

}

else

return 0;

}

void delete\_by\_index(int i) {

node\* buf = find\_by\_index(i);

if (buf != nullptr) {

if (buf == head) {

head = buf->next;

if (head != nullptr)

head->previous = nullptr;

if (buf == head && size() == 1)

last = nullptr;

delete buf;

}

else if (buf == last) {

last = buf->previous;

last->next = nullptr;

delete buf;

}

else {

buf->previous->next = buf->next;

buf->next->previous = buf->previous;

delete buf;

}

}

}

void insert\_by\_index(int i, request\* key) {

if (!empty()) {

node\* buf = new node();

buf->R = key;

if (buf != nullptr) {

if (i == 0) {

buf->next = head;

head->previous = buf;

head = buf;

}

else {

node\* buf2 = find\_by\_index(i);

buf->next = buf2;

buf2->previous->next = buf;

buf->previous = buf2->previous;

buf2->previous = buf;

}

}

}

}

void swap(int i, int j) {

node\* n1 = find\_by\_index(i);

node\* n2 = find\_by\_index(j);

if (i < j && j == size() - 1) {

insert\_by\_index(i, n2->R);

delete\_by\_index(j + 1);

add(n1->R);

delete\_by\_index(i + 1);

}

else if (i < j) {

insert\_by\_index(i, n2->R);

delete\_by\_index(j + 1);

insert\_by\_index(j, n1->R);

delete\_by\_index(i + 1);

}

else if (i > j && i == size() - 1) {

insert\_by\_index(j, n1->R);

delete\_by\_index(i + 1);

add(n2->R);

delete\_by\_index(j + 1);

}

else if ( i > j) {

insert\_by\_index(j, n1->R);

delete\_by\_index(i + 1);

insert\_by\_index(i, n2->R);

delete\_by\_index(j + 1);

}

}

request\* operator [](int i) {

request\* t = find\_by\_index(i)->R;

return t;

}

};

List requests;

char\* Enter() {

char\* str = new char[256];

cin >> str;

return str;

}

void print\_column\_names() {

cout << endl << '|' << left << setw(max\_len + 1) << "Отправление: " << '|' << setw(max\_len + 1) << "Прибытие: " << '|' << setw(max\_len + 1) << "Номер рейса:" << '|';

cout << setw(max\_len + 1) << "Имя:" << '|' << setw(max\_len + 1) << "Фамилия:" << '|' << setw(max\_len + 1) << "Отчество:" << '|';

cout << setw(max\_len) << "Дата:" << '|' << setw(max\_len - 6) << "Время:" << '|' << setw(5) << "i:" << '|' << endl;

}

void enter\_full\_name(my\_string\* s, my\_string\* n, my\_string\* p) {

char\* str = new char[256];

for (int i = 0; i< 3; i++) {

cin >> str;

if (i == 0)

s->change\_ch(str);

if (i == 1)

n->change\_ch(str);

if (i == 2)

p->change\_ch(str);

}

}

void adding() {

my\_string\* departure;

my\_string\* destination;

my\_string\* flight\_number;

my\_string\* surname = new my\_string();

my\_string\* name = new my\_string();

my\_string\* patronymic = new my\_string();

my\_string\* departure\_date;

my\_string\* departure\_time;

cout << endl;

cout << "Отправление из: ";

departure = new my\_string(Enter());

cout << "Прибытие в: ";

destination = new my\_string(Enter());

cout << "Номер рейса: ";

flight\_number = new my\_string(Enter());

cout << "Ф.И.О: ";

enter\_full\_name(surname, name, patronymic);

cout << "Дата вылета: ";

departure\_date = new my\_string(Enter());

cout << "Время вылета: ";

departure\_time = new my\_string(Enter());

request\* test = new request(departure , destination, flight\_number, surname, name , patronymic, departure\_date, departure\_time);

requests.add(test);

}

void print\_all\_r() {

if (requests.size() > 0) {

for (int i = 0; i < max\_len \* 8 + 8 + 7; i++)

cout << '\_';

print\_column\_names();

for (int i = 0; i < requests.size(); i++) {

request\* current = requests[i];

cout << current;

cout <<" "<< left<< setw(5) << i << '|' << endl;

}

for (int i = 0; i < max\_len \* 8 + 8 + 7; i++)

cout << '-';

cout << endl;

}

else

cout << "Ошибка. Список пуст.";

}

void remove() {

if (0 != requests.size()) {

int t;

cout << "1. Удалить по индексу \n";

cout << "2. Удалить по дате \n";

cout << "3. Удалить по номеру рейса\n";

cout << "4. Удалить все заявки \n:";

cin >> t;

if (t == 1) {

int i;

cout << "Индекс: ";

cin >> i;

if (i > -1 && i < requests.size()) {

requests.delete\_by\_index(i);

cout << "Успешно";

}

else

cout << "Ошибка. Выхд за рамки списка.";

}

else if (t == 2) {

my\_string\* date;

cout << "Дата: ";

bool f = false;

date = new my\_string(Enter());

for (int j = 0; j < requests.size(); j++) {

if (\*requests[j]->get\_date() == \*date) {

requests.delete\_by\_index(j);

f = true;

j = 0;

}

}

if (f)

cout << "Успешно";

else

cout << "Не найдено";

}

else if (t == 3) {

my\_string\* fn;

cout << "Номер рейса: ";

fn = new my\_string(Enter());

bool f = false;

for (int j = 0; j < requests.size(); j++) {

if (\*requests[j]->get\_flight\_n() == \*fn) {

requests.delete\_by\_index(j);

j = 0;

f = true;

}

}

if (f)

cout << "Успешно";

else

cout << "Не найдено";

}

else if (t == 4) {

int n = requests.size();

for (int j = 0; j < n; j++) {

requests.delete\_by\_index(0);

}

}

}

else

cout << "Ошибка. Список пуст.";

}

void find() {

if (!requests.empty()) {

int i;

cout << endl << "Индекс: ";

cin >> i;

print\_column\_names();

cout << requests[i] << " " << left << setw(5) << i << '|' << endl;

}

else

cout << "Ошибка. Список пуст.";

}

void load\_data() {

ifstream file("database.txt");

my\_string\* departue;

my\_string\* destination;

my\_string\* flight\_number;

my\_string\* surname;

my\_string\* name;

my\_string\* patronymic;

my\_string\* departure\_date;

my\_string\* departure\_time;

char\* buf = new char[256];

if (file.is\_open()) {

while (true) {

file >> buf;

my\_string\* departure = new my\_string(buf);

file >> buf;

my\_string\* destination = new my\_string(buf);

file >> buf;

my\_string\* flight\_number = new my\_string(buf);

file >> buf;

my\_string\* surname = new my\_string(buf);

file >> buf;

my\_string\* name = new my\_string(buf);

file >> buf;

my\_string\* patronymic = new my\_string(buf);

file >> buf;

my\_string\* departure\_date = new my\_string(buf);

file >> buf;

my\_string\* departure\_time = new my\_string(buf);

request\* test = new request(departure, destination, flight\_number, surname, name, patronymic, departure\_date, departure\_time);

requests.add(test);

if (file.eof())

break;

}

cout << "Успешно";

}

else {

cout << "Ошибка. Проверьте файл database.txt";

}

}

void find\_by\_d\_fl() {

int i;

bool flag = true;

cout << endl;

cout << "|1. По номеру рейса |\n";

cout << "|2. По дате |\n";

cout << "|3. По дате и номеру рейса|\n";

cout << ':';

cin >> i;

cout << endl;

if (requests.size() > 0) {

if (i == 1) {

my\_string\* fn;

cout << "Номер рейса: ";

fn = new my\_string(Enter());

for (int j = 0; j < requests.size(); j++) {

if (\*requests[j]->get\_flight\_n() == \*fn) {

if (flag) {

print\_column\_names();

flag = false;

}

cout << requests[j] << setw(5) << j << '|' << endl;

}

}

if (flag) {

cout << "Заявок с данными параметрами не найдены.";

}

}

else if (i == 2) {

my\_string\* date;

cout << "Дата: ";

date = new my\_string(Enter());

for (int j = 0; j < requests.size(); j++) {

if (\*requests[j]->get\_date() == \*date) {

if (flag) {

print\_column\_names();

flag = false;

}

cout << requests[j] << setw(5) << j << '|' << endl;

}

}

if (flag) {

cout << "Заявок с данными параметрами не найдены.";

}

}

else if (i == 3) {

my\_string\* fn;

cout << "Номер рейса: ";

fn = new my\_string(Enter());

my\_string\* date;

cout << "Дата: ";

date = new my\_string(Enter());

for (int j = 0; j < requests.size(); j++) {

if ((\*requests[j]->get\_date() == \*date) && (\*requests[j]->get\_flight\_n() == \*fn)) {

if (flag) {

print\_column\_names();

flag = false;

}

cout << requests[j] << setw(5) << j << '|' << endl;

}

}

if (flag) {

cout << "Заявок с данными параметрами не найдено.";

}

}

}

else

cout << "Ошибка. Список пуст.";

}

void p\_menu() {

cout << "-------------------------Menu------------------------------|\n";

cout << "| 0. Выход |\n";

cout << "| 1. Вывести меню |\n";

cout << "| 2. Добавить заявку |\n";

cout << "| 3. Вывести на экран весь список заявок |\n";

cout << "| 4. Найти по индекску заявку |\n";

cout << "| 5. Удалить заявку/заявки |\n";

cout << "| 6. Вывод заявок по заданному номеру рейса и дате вылета |\n";

cout << "| 7. Сохранить данные |\n";

cout << "| 8. Загрузить |\n";

cout << "| 9. Сортировка по дате |\n";

cout << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n";

}

void save(){

if (requests.size() != 0) {

cout << "Удалить предыдущие данные?\n";

cout << "1.Да\n";

cout << "2.Нет\n:";

int t;

cin >> t;

if (t == 1) {

ofstream file("database.txt", ios::trunc);

for (int i = 0; i < requests.size(); i++) {

file << requests[i];

if (i != requests.size() - 1)

file << endl;

}

cout << "Успешно";

}

else if (t == 2) {

ofstream file("database.txt", ios::app);

file << endl;

for (int i = 0; i < requests.size(); i++) {

file << requests[i];

if (i != requests.size() - 1)

file << endl;

}

cout << "Успешно";

}

}

else

cout << "Ошибка. Список пуст.";

}

int compare(my\_string\* d1 , my\_string\* d2) {

int y1, y2, m1, m2 , day1, day2;

(d1->get\_ch()[0] & 0xCF) != 0 ? day1 = (d1->get\_ch()[0] & 0xCF) \* 10 + (d1->get\_ch()[1] & 0xCF) : day1 = (d1->get\_ch()[1] & 0xCF);

(d2->get\_ch()[0] & 0xCF) != 0 ? day2 = (d2->get\_ch()[0] & 0xCF) \* 10 + (d2->get\_ch()[1] & 0xCF) : day2 = (d2->get\_ch()[1] & 0xCF);

(d1->get\_ch()[3] & 0xCF) != 0 ? m1 = (d1->get\_ch()[3] & 0xCF) \* 10 + (d1->get\_ch()[4] & 0xCF) : m1 = (d1->get\_ch()[4] & 0xCF);

(d2->get\_ch()[3] & 0xCF) != 0 ? m2 = (d2->get\_ch()[3] & 0xCF) \* 10 + (d2->get\_ch()[4] & 0xCF) : m2 = (d2->get\_ch()[4] & 0xCF);

(d1->get\_ch()[6] & 0xCF) != 0 ? y1 = (d1->get\_ch()[6] & 0xCF) \* 10 + (d1->get\_ch()[7] & 0xCF) : y1 = (d1->get\_ch()[7] & 0xCF);

(d2->get\_ch()[6] & 0xCF) != 0 ? y2 = (d2->get\_ch()[6] & 0xCF) \* 10 + (d2->get\_ch()[7] & 0xCF) : y2 = (d2->get\_ch()[7] & 0xCF);

if (y1 > y2)

return 1;

else if (y1 < y2)

return -1;

else {

if (m1 > m2)

return 1;

else if (m1 < m2)

return -1;

else {

if (day1 > day2)

return 1;

else if (day1 < day2)

return -1;

else

return 0;

}

}

return 0;

}

Stack s;

void sort\_by\_date() {

if (!requests.empty()) {

int n = requests.size();

s.push(0);

s.push(n - 1);

while (!s.empty()) {

int l = s.pop();

int f = s.pop();

while (f < l) {

int f1 = f;

int l1 = l;

my\_string\* mid = requests[(f1 + l1) / 2]->get\_date();

while (f1 < l1) {

while (compare(requests[f1]->get\_date(), mid) == -1) {

f1++;

}

while (compare(requests[l1]->get\_date(), mid) == 1)

l1--;

if (f1 <= l1) {

requests.swap(f1, l1);

f1++;

l1--;

}

}

if (f1 < l) {

s.push(f1);

s.push(l);

}

l = l1;

}

}

cout << "Успешно";

}

else

cout << "Ошибка список пуст";

}

void menu() {

p\_menu();

int ent = -1;

while (ent != 0) {

cout << endl << "Ввод: ";

cin >> ent;

switch (ent)

{

case(1):

p\_menu();

break;

case(2):

adding();

break;

case(3):

print\_all\_r();

break;

case(4):

find();

break;

case(5):

remove();

break;

case(6):

find\_by\_d\_fl();

break;

case(7):

save();

break;

case(8):

load\_data();

break;

case(9):

sort\_by\_date();

break;

default:

break;

}

}

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

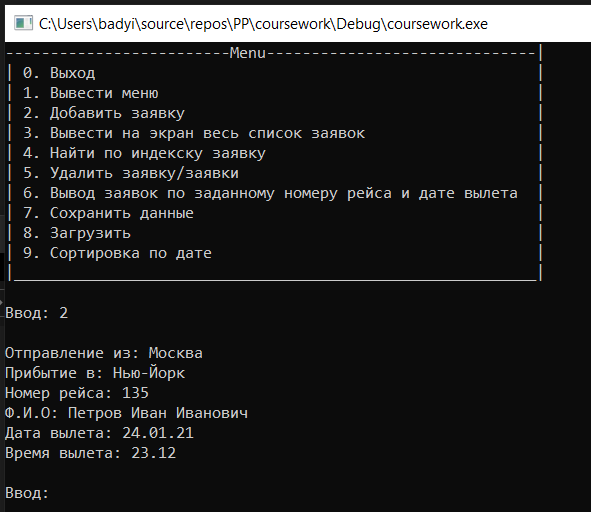
menu();

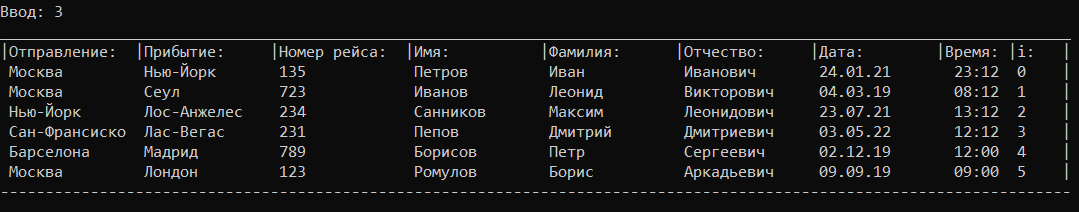
return 0;

}

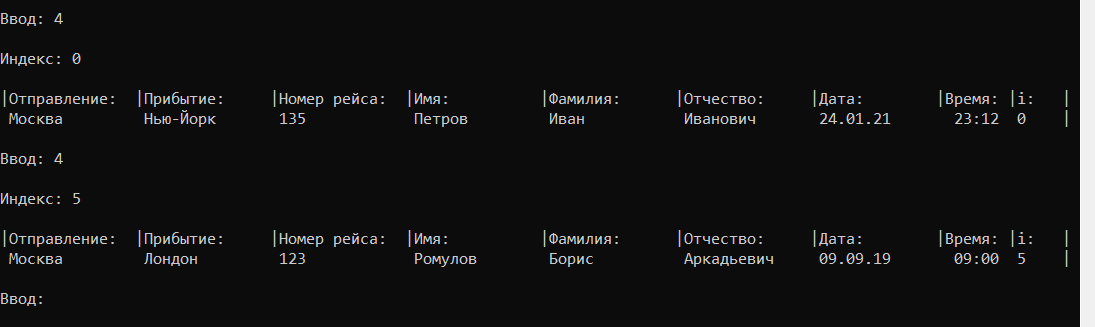
# 3. Результаты работы программы.

1. Добавим заявку

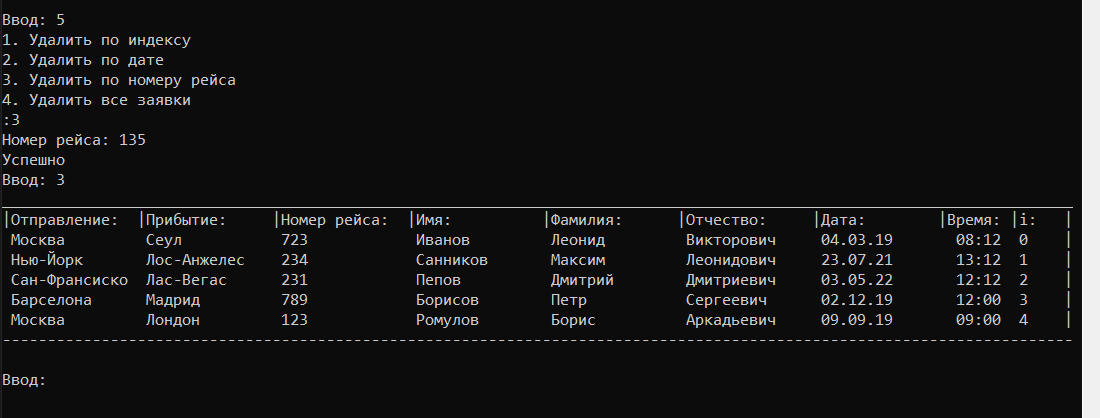
2.Повторим это еще несколько раз

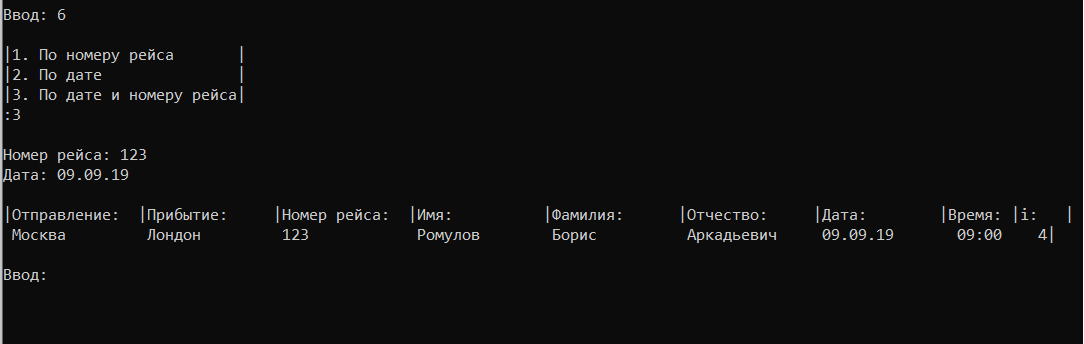
3. Выведем на экран

4.Поиск по индексу



5. Удалим заявки по индексу и выведем список

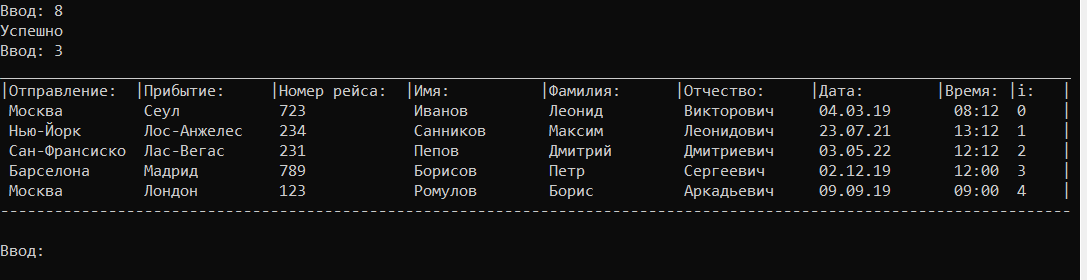


6. Найдем заявку по дате и номеру рейса

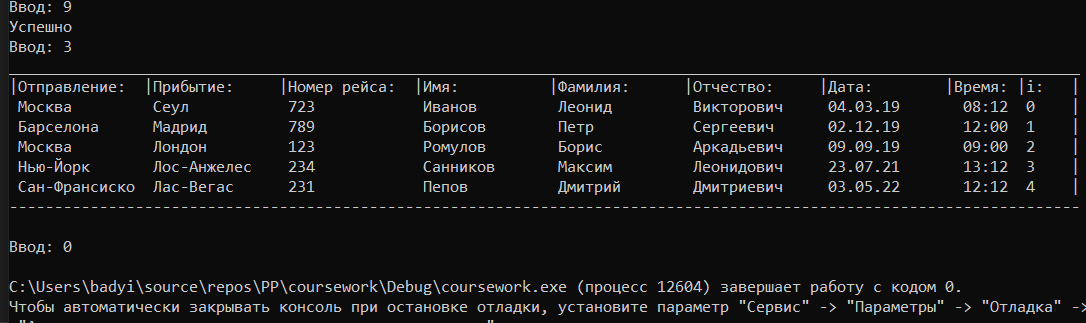
7. Сохраним в файл текущий список. Удалим все заявки и попробуем вывести список.



8. Загрузим данные и выведем список.



9. Отсортируем по дате, выведем список и завершим работу.



# 4. Заключение.

Подводя итог, в моей курсовой работе были:

* рассмотрены и сравнены императивная и декларативная методологии программирования;
* проанализированы различные инструментальные среды разработки (IDE).
* Был рассмотрен контейнерный класс – связный список. Односвзяный список и двусвзяный список. Методы добавления, удаления, поиска, вывода элементов списка. Было проведено сравнение с одномерным массивом в эффективности.
* Была разработана программа на объектно-ориентированном языке программирования высокого уровня С++ в MICROSOFT VISUAL STUDIO 2017, а именно прикладная программа по учету заявок на авиабилеты. В программе реализовано добавление заявок в список; удаление заявок; вывод заявок по заданному номеру рейса и дате вылета; вывод всех заявок;
* При создании программы был разработан контейнерный класс структуры данных – связный список, а также – стек.

В ходе выполнения работы я узнал об отличиях декларативной и императивной методологий, а также узнал об различных инструментов разработки. Также я узнал о динамических структурах данных – связный список и стек. Научился применять их для решения такой прикладной задачи как учет заявок на авиабилеты. Я узнал много нового, в частности, о работе с UML диаграммами классов, а именно то, что они могут быть так полезны при разработке программного средства. Также я научился работать с блок-схемами алгоритмов.

Выводом моей деятельности является то, что на сегодняшний день разработки прикладных программ не может обойтись без использования императивной методологии программирования, а именно объектно-ориентированного подхода. Также на сегодняшний день прикладная программа не может обойтись без использования различных структур данных, которые хранят в себе информацию задействованную в программе.

# Источники

* <http://lib.ru/CPPHB/cpptut.txt> - Бьерн Страуструп. Язык программирования С++
* <https://ru.wikipedia.org/wiki/UML> - Диаграммы классов
* <https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%BB%D0%B0%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5> – Декларативное программирование
* <https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5> – Императивное программирование
* <https://spravochnick.ru/programmirovanie/yazyki_programmirovaniya/klassifikaciya_yazykov_programmirovaniya/> - классификация языков программирования
* <https://docs.microsoft.com/ru-ru/visualstudio/?view=vs-2019> – документация MICROSOFT VISUAL STUDIO
* <https://javarush.ru/quests/lectures/questharvardcs50.level05.lecture05> - Harvard CS50, Связный список