МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

КУРСОВОЙ ПРОЕКТ
ЗАЩИЩЕН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

ст. преподаватель		С.А. Рогачёв
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОМУ ПРОЕКТУ

ИСПОЛЬЗОВАНИЕ ЗАДАННЫХ СТРУКТУР ДАННЫХ И АЛГОРИТМОВ ПРИ РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ

по дисциплине: СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4131		Д.А.Чардымов
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2023

ОГЛАВЛЕНИЕ

1. Задание на курсовой проект

Цель курсового проектирования: изучение структур данных и алгоритмов их обработки, а также получение практических навыков их использования при разработке программ.

Задача курсового проекта: разработка информационной системы для заданной предметной области с использованием заданных структур данных и алгоритмов.

Тема курсового проекта: «Использование заданных структур данных и алгоритмов при разработке программного обеспечения информационной системы».

Вариант:

Компоненты	№	Наименование
Предметная область	4	Продажа авиабилетов
Метод хеширования	2	Закрытое хеширование с квадратичным опробованием
Метод сортировки	3	Шейкерная
Вид списка	1	Линейный двунаправленный
Метод обхода дерева	1	Обратный
Алгоритм поиска слова в тексте	0	Боуера-Мура

2. Введение

Необходимо реализовать информационную систему аэропорта, в функции которой входит работа с данными о пассажирах, об авиарейсах, относящихся к данному аэропорту, и с данными о билетах на рейсы.

3. Алгоритмы и структуры данных

Данные о пассажирах хранятся в структуре (хеш-таблица) Table. Данные о авиарейсах хранятся в структуре (АВЛ-дерево) tree. Данные об авиабилетах хранятся в структуре (список) spisok.

```
struct Table {
   passenger value;
   bool used;
        };

struct tree {
   flight value;
   int h;
   tree* left;
   tree* right;
   tree(flight v)
   {
      value = v;
      left = right = NULL;
      h = 1;
   }
};

struct spisok {
   std::string passport;
   std::string flightNum;
   std::string ticketNum;

   spisok* prev;
   spisok* next;
         };
```

Для поиска авиарейса по фрагменту аэропорта прибытия используется алгоритм Боуэра-Мура.

4. Описание программы и функций

Программа реализована на языке C++ в виде консольного приложения. В программе реализовано меню пользователя, в котором каждому действию

соответствует определенная цифра. Реализованы следующие функции для работы с данными:

1. Добавление нового пассажира
2. Удаление данных о пассажире
3. Просмотр всех зарегестрированных пассажиров
4. Очистка данных о пассажирах
5. Поиск пассажира по номеру пасспорта
6. Поиск пассажира по его ФИО
7. Добавление нового рейса
8. Удаление рейса
9. Удаление всех рейсов
10. Просмотр всех рейсов
11. Поиск рейса по его номеру
12. Поиск рейса по фрагменту названия аэропорта прибытия
13. Регистрация продажи билета пассажиру
14. Регистрация возврата билета
15. Выход из программы

Переходы между пунктами определяются в меню. Реализация всех функций находится в приложении А

Добавление записи в хеш-таблицу

```
void Add(Keys* k, int a, passenger pas, int length)
```

Пользователь вводит новые значения, производится проверка корректности введённого номера паспорта. Далее вносятся остальные данные о пассажире (место выдачи паспорта, ФИО, дата рождения). Они добавляются в хеш-таблицу.

Удаление записи из хеш-таблицы

```
bool DelElem(Keys* k, int length, string number)
```

Производится поиск пассажира по номеру паспорта. При успешном

нахождении производится проверка на то, что этот пассажир зарегистрирован на рейс. Если пассажир зарегистрирован, то его данные удаляются из хеш-таблицы. Если данного пассажира нет, программа выводит соответствующее сообщение.

Поиск в хеш-таблице по ключу

```
void showOnePassanger(Keys* k, int length, std::string numb)
```

Производится поиск пассажира по номеру паспорта. Если данного пассажира нет, программа выводит сообщение об его отсутствии. Если пассажир найден, то вся информация о пассажире выводится в консоль.

Поиск в хеш-таблице по значению

```
bool searchByFIO(Keys* k, int length, string numb)
```

Пользователь вводит ФИО пассажира. Производится поиск. Если данного пассажира нет, программа выводит сообщение об отсутствии пассажира. Если пассажир, или пассажира найдены, то вся информация о них выводится в консоль.

Вывод хеш-таблицы на экран

```
void Show(Keys* k, int length)
```

На экран выводятся все элементы хеш-таблицы.

Удаление всех записей из хеш-таблицы

```
void delAllTable(Keys* k, int length)
```

Удаляются все элементы хеш-таблицы. После выполнения выводится сообщение о завершении операции.

Добавление записи в дерево

```
tree* insertTree(tree* r, flight* val)
```

Пользователь вводит номер нового авиарейса, затем проверяется

корректность введённого номера. Если проверка пройдена успешно, вводятся остальные значения полей авиарейса. Далее они добавляются в дерево.

Удаление записи из дерева

```
tree* eraseTreeElem(tree* rr, string numb)
```

Производится поиск рейса по номеру авиарейса. Если рейс найден и не используется в других структурах, то он удаляется из дерева. Иначе программа выводит сообщение об его отсутствии или сообщит пассажирам данного рейса о его отмене.

Поиск в дереве по номеру

```
bool searchByFlightNum(tree* r, std::string str)
```

Производится поиск рейса по номеру авиарейса. Если данного рейса нет, программа выводит сообщение об отсутствии рейса. Если авиарейс найден, то выводится вся информация о нём.

Поиск в дереве по фрагменту аэропорта прибытия

```
bool searchByDestHelper(tree* r, std::string str)
```

Производится поиск по фрагменту аэропорта прибытия. Если данного рейса нет, программа выводит сообщение об отсутствии элемента. Если рейс найден, то выводится вся информация о нём.

Вывод дерева на экран

```
void showtree(tree* r, int l)
```

На экран выводятся все элементы дерева.

Удаление всех записей из дерева

```
void deleteAllTree(tree* r)
```

Удаляются все элементы дерева. После выполнения выводится сообщение о завершении операции. Если какие то рейсы были использованы,

то выводится соответсвующее сообщение для пассажиров.

Добавление билета в список

```
void List::addSellTicket(std::string pas, std::string flight)
```

Пользователь вводит значения номеров паспорта, авиарейса и номер авиабилета, производится проверка номеров на корректность (они должны быть в хеш-таблице и дереве соответственно). Билет добавляется в список.

Удаление билета из списка

```
bool List::delTicketsByFlightAndPas(std::string fl, std::string pas)
```

Производится поиск по введенному номеру пасспорта. Затем предлагается на ввыбор удалить один из билетов, на которые зарегестрирован данный пассажир. Удаляется элемент списка. Если данного элемента нет, программа выводит сообщение о его отсутствии. После выполнения сообщение об удалении выводится на экран.

Сортировка списка по пункту назначения

```
void List::sort()
```

Производится сортировка списка.

Выход из программы

Полное удаление всех данных и завершение работы приложения.

5. Результаты тестирования программы

5.1Словесное описание:

Основное меню:

№	Действие	Входные данные	Результат
1	Добавление нового пассажира (1)	Номер паспорта, место и даты выдачи, ФИО, дата рождения	Добавление пассажира
2	Удалить пассажира (2)	Номер паспорта	Сообщение об удалении пассажира
3	Просмотр всех зарегестрированных пассажиров (3)	-	Вывод всех зарегистрированных пассажиров на экран
4	Очистка всех записей с информацией о пассажирах (4)	-	Сообщение об удалении пассажиров
5	Поиск пассажира по номеру паспорта (5)	Номер паспорта	Вывод информации о найденном пассажире
6	Поиск пассажира по его ФИО (6)	ФИО пассажира	Вывод информации найденном пассажире
7	Добавление нового рейса (7)	Номер авиарейса, название компании, аэропорт	Добавление авиарейса

		отправления, аэропорт прибытия, время отправления, время прибытия, количество мест, количество свободных мест	
8	Удалить авиарейса (8)	Номер авиарейса	Сообщение об удалении авиарейса
9	Очистка всех авиарейсов (9)	-	Сообщение об удалении авиарейсов
10	Просмотр всех зарегистрированных авиарейсов (10)	-	Вывод всех зарегистрированных авиарейсов
11	Найти авиарейс по его номеру (11)	Номер авиарейса	Выводится авиарейс с соответствующим номером
12	Найти авиарейс по аэропорту прибытия (12)	Аэропорт прибытия	Вывод информации о найденных авиарейсах
13	Продажа билета пассажиру (13)	Номер паспорта, номер авиарейса	Добавляется новый билет в список
14	Возврат билета (14)	Номер пасспорта, номер авиарейса	Удаляется билет из списка

15	Выход из программы (15)		Сообщение о завершении работы программы

5.2 Тестовые данные:

Основное меню:

№	Действие	Входные данные	Результат
1	Зарегистрировать пассажира (1)	1331-640781 Бурый И. С. 07.05.1995 г. Москва 22.02.2003	Добавление пассажира
2	Удалить пассажира (2)	1111-880101	Сообщение об удалении пассажира
3	Вывести всех зарегистрированных пассажиров на экран (3)	-	Вывод всех зарегистрированных пассажиров на экран
4	Очистка всех записей с информацией о пассажирах (4)	-	Сообщение об удалении пассажиров
5	Найти пассажира по номеру паспорта (5)	1331-640781	1331-640781 г. Москва 22.02.2003 Бурый И. С. 07.05.1995 Авиарейсы: DFG-021
6	Найти пассажира по его ФИО (6)	Бурый И. С	1331-640781 Бурый И. С.
7	Зарегистрировать авиарейс (7)	DFG-021 Победа	Добавление авиарейса

		Сочи Махачкала 12.04.2023 15:00 100 30	
8	Удалить авиарейс (8)	HGD-012	Сообщение об удалении авиарейса
9	Очистка всех авиарейсов (9)	-	Сообщение об удалении авиарейсов
10	Вывести все авиарейсы на экран (10)	-	Вывод всех зарегистрированных авиарейсов
11	Найти авиарейс по его номеру (11)	DFG-123	DFG-123 Russia Отправление: Ставрополь Назначение: Санкт-Петербург Дата вылета: 12.02.23 Время вылета: 16:50 Количество мест: 100 Свободных мест: 10
12	Найти авиарейс аэропорту прибытия (12)	бург	DFG-123 Russia Отправление: Ставрополь Назначение: Санкт-Петербург

			Дата вылета: 12.02.23 Время вылета: 16:50 Количество мест: 100 Свободных мест: 10
13	Зарегистрировать билет(13)	1331-640781 DFG-021	Добавление билета №000000001
14	Вернуть билет (14)	1331-640781 DFG-021	Возврат билета
15	Выход из программы (0)		Сообщение о завершении работы программы

5.3 Скриншоты выполнения тестовых данных

```
====== ГЛАВНОЕ МЕНЮ ======
1. Добавление нового пассажира
2. Удаление данных о пассажире
3. Просмотр всех зарегистрированных пассажиров
4. Очистка данных о пассажирах
5. Поиск пассажира по «номеру паспорта»
6. Поиск пассажира по его ФИО
7. Добавление нового рейса
8. Удаление авиарейса
9. Удаление всех авиарейсов
10. Просмотр всех авиарейсов
11. Поиск авиарейса по «номеру авиарейса»
12. Поиск авиарейса по фрагментам названия аэропорта прибытия
13. Регистрация продажи билета пассажиру
14. Регистрация возврата билета
15. Выход из программы
```

Рисунок 1 – Главное меню

```
Выберите номер главного меню: 1
Введите номер паспорта. Строка должна быть формата «NNNN-NNNNNN»: 3333-ooooooo
Введены некорректные серия и номер паспорта. Строка должна быть формата «NNNN-NNNNNN»! Попробуйте снова:
    3333-333333
Введите место регистрации паспорта: St Petersburg
Введите дату регистрации паспорта: 33 9 2009
Введена некорректная дата! Дата должна иметь следующий вид: день месяц год. Попробуйте снова: 30 9 2009
Введите свою ФИО: Ivanov Ivan
Введены некорректные ФИО. Попробуйте снова: Ivanov Ivan Ivanovich
Введите свою дату рождения: 4 4 2010
Введена некорректная дата рождения! Дата должна иметь следующий вид: день месяц год. Попробуйте снова: 4 4
    1998
Пассажир успешно добавлен!
36: 3333-333333 Ivanov Ivan Ivanovich
```

Рисунок 2 – Добавление нового пассажира

```
Выберите номер главного меню: 2
454: 4222-444444 Ivanov Petr Artomovich
593: 1112-345567 Murahtanov Artem Andreevich
664: 4222-444455 Knyazev Ivan Andreevich
Введите номер паспорта пассажира, которого хотите удалить: 4222-uuuuuu
Введены некорректные серия и номер паспорта. Строка должна быть формата «NNNN-NNNNNN»! Попробуйте снова:
    4222-444666
Не было найдено пассажира с введнным номером паспорта!

Выберите номер главного меню: 2
454: 4222-444444 Ivanov Petr Artomovich
593: 1112-345567 Murahtanov Artem Andreevich
664: 4222-444455 Knyazev Ivan Andreevich
Введите номер паспорта пассажира, которого хотите удалить: 4222-444444
Пассажир с номером паспорта 4222-444444 был успешно удален!
```

Рисунок 3 – Удаление данных о пассажире

```
====== ГЛАВНОЕ МЕНЮ ======
1. Добавление нового пассажира
2. Удаление данных о пассажире
3. Просмотр всех зарегистрированных пассажиров
4. Очистка данных о пассажирах
5. Поиск пассажира по «номеру паспорта»
6. Поиск пассажира по его ФИО
7. Добавление нового рейса
8. Удаление авиарейса
9. Удаление всех авиарейсов
10. Просмотр всех авиарейсов
11. Поиск авиарейса по «номеру авиарейса»
12. Поиск авиарейса по фрагментам названия аэропорта прибытия
13. Регистрация продажи билета пассажиру
14. Регистрация возврата билета
15. Выход из программы

Выберите номер главного меню: 3
454: 4222-444444 Ivanov Petr Artomovich
593: 1112-345567 Murahtanov Artem Andreevich
664: 4222-444455 Knyazev Ivan Andreevich
```

Рисунок 4 – Просмотр всех зарегистрированных пассажиров

```
====== ГЛАВНОЕ МЕНЮ ======
1. Добавление нового пассажира
2. Удаление данных о пассажире
3. Просмотр всех зарегистрированных пассажиров
4. Очистка данных о пассажирах
5. Поиск пассажира по «номеру паспорта»
6. Поиск пассажира по его ФИО
7. Добавление нового рейса
8. Удаление авиарейса
9. Удаление всех авиарейсов
10. Просмотр всех авиарейсов
11. Поиск авиарейса по «номеру авиарейса»
12. Поиск авиарейса по фрагментам названия аэропорта прибытия
13. Регистрация продажи билета пассажиру
14. Регистрация возврата билета
15. Выход из программы

Выберите номер главного меню: 4
Данные обо всех пассажирах были удалены!
```

Рисунок 5 – Очистка данных о пассажирах

```
Выберите номер главного меню: 5
454: 4222-444444 Ivanov Petr Artomovich
593: 1112-345567 Murahtanov Artem Andreevich
664: 4222-444455 Knyazev Ivan Andreevich
Введите номер паспорта пассажира, которого хотите просмотреть: 4222-555555
Пассажира с введенным номером паспорта не найдено!

Выберите номер главного меню: 5
454: 4222-444444 Ivanov Petr Artomovich
593: 1112-345567 Murahtanov Artem Andreevich
664: 4222-444455 Knyazev Ivan Andreevich
Введите номер паспорта пассажира, которого хотите просмотреть: 1112-345567
Номер паспорта: 1112-345567
Место выдачи: uvd 667
Дата выдачи: 3 5 2017
ФИО: Murahtanov Artem Andreevich
Дата рождения: 1 3 2003
Пассажир не зарегестрирован ни на один рейс
```

Рисунок 6 – Поиск пассажира по номеру паспорта

```
Выберите номер главного меню: 6
Введите ФИО пассажира, которого хотите просмотреть: Ivanov Ivan Ivanovich
Не было найдено пассажиров с заданными ФИО

Выберите номер главного меню: 6
Введите ФИО пассажира, которого хотите просмотреть: Murahtanov Artem Andreevich
1112-345567 Murahtanov Artem Andreevich
```

Рисунок 7 – Поиск пассажира по его ФИО

```
Выберите номер главного меню: 7
Введите номер авиарейса. Строка должна быть формата «AAA-NNN»:AAA-00A
Введен некорректный номер рейса! Попробуйте снова: AAA-000
Введите место отправления:St Petersburg
Введите направление рейса: Moskov
Введите название авиакомпании:Aeroflot
Введите дату полета:9 9 2023
Введите количество мест:5
Новый рейс успешно добавлен!
```

Рисунок 8 – Добавление нового авиарейса

```
====== ГЛАВНОЕ МЕНЮ ======
1. Добавление нового пассажира
2. Удаление данных о пассажире
3. Просмотр всех зарегистрированных пассажиров
4. Очистка данных о пассажирах
5. Поиск пассажира по «номеру паспорта»
6. Поиск пассажира по его ФИО
7. Добавление нового рейса
8. Удаление авиарейса
9. Удаление всех авиарейсов
10. Просмотр всех авиарейсов
11. Поиск авиарейса по «номеру авиарейса»
12. Поиск авиарейса по фрагментам названия аэропорта прибытия
13. Регистрация продажи билета пассажиру
14. Регистрация возврата билета
15. Выход из программы

Выберите номер главного меню: 8
Введите номер авиарейса, который хотите удалить. Строка должна быть формата «AAA-NNN»:BBB-000
Авиарейс успешно удален!
```

Рисунок 9 – Удаление авиарейса

```
====== ГЛАВНОЕ МЕНЮ ======
1. Добавление нового пассажира
2. Удаление данных о пассажире
3. Просмотр всех зарегистрированных пассажиров
4. Очистка данных о пассажирах
5. Поиск пассажира по «номеру паспорта»
6. Поиск пассажира по его ФИО
7. Добавление нового рейса
8. Удаление авиарейса
9. Удаление всех авиарейсов
10. Просмотр всех авиарейсов
11. Поиск авиарейса по «номеру авиарейса»
12. Поиск авиарейса по фрагментам названия аэропорта прибытия
13. Регистрация продажи билета пассажиру
14. Регистрация возврата билета
15. Выход из программы

Выберите номер главного меню: 9
Все авиарейсы успешно удалены!
```

Рисунок 10 – Удаление всех авиарейсов

```
====== ГЛАВНОЕ МЕНЮ ======
1. Добавление нового пассажира
2. Удаление данных о пассажире
3. Просмотр всех зарегистрированных пассажиров
4. Очистка данных о пассажирах
5. Поиск пассажира по «номеру паспорта»
6. Поиск пассажира по его ФИО
7. Добавление нового рейса
8. Удаление авиарейса
9. Удаление всех авиарейсов
10. Просмотр всех авиарейсов
11. Поиск авиарейса по «номеру авиарейса»
12. Поиск авиарейса по фрагментам названия аэропорта прибытия
13. Регистрация продажи билета пассажиру
14. Регистрация возврата билета
15. Выход из программы

Выберите номер главного меню: 10
    III-556
III-555
        BBB-000
    AAA-000
```

Рисунок 11 – Просмотр всех авиарейсов

```
====== ГЛАВНОЕ МЕНЮ ======
1. Добавление нового пассажира
2. Удаление данных о пассажире
3. Просмотр всех зарегистрированных пассажиров
4. Очистка данных о пассажирах
5. Поиск пассажира по «номеру паспорта»
6. Поиск пассажира по его ФИО
7. Добавление нового рейса
8. Удаление авиарейса
9. Удаление всех авиарейсов
10. Просмотр всех авиарейсов
11. Поиск авиарейса по «номеру авиарейса»
12. Поиск авиарейса по фрагментам названия аэропорта прибытия
13. Регистрация продажи билета пассажиру
14. Регистрация возврата билета
15. Выход из программы

Выберите номер главного меню: 11
Введите номер авиарейса, который хотите найти. Строка должна быть формата «AAA-NNN»:III-556
Номер авиарейса: III-556
Название компании: Aeroflot
В Sochi
Из Moskov
Дата полета: 28 7 2023
Время полета: 6:30
Всего мест: 1
Свободных мест: 1
```

Рисунок 12 – Поиск авиарейса по номеру авиарейса

```
====== ГЛАВНОЕ МЕНЮ ======
1. Добавление нового пассажира
2. Удаление данных о пассажире
3. Просмотр всех зарегистрированных пассажиров
4. Очистка данных о пассажирах
5. Поиск пассажира по «номеру паспорта»
6. Поиск пассажира по его ФИО
7. Добавление нового рейса
8. Удаление авиарейса
9. Удаление всех авиарейсов
10. Просмотр всех авиарейсов
11. Поиск авиарейса по «номеру авиарейса»
12. Поиск авиарейса по фрагментам названия аэропорта прибытия
13. Регистрация продажи билета пассажиру
14. Регистрация возврата билета
15. Выход из программы

Выберите номер главного меню: 12
Введите название аэропорта прибытия полностью или его часть: chi
Номер авиарейса: III-555
Название компании: American Airlines
В Sochi
Из Moskov
Дата полета: 6 9 2023
Время полета: 23:55
Всего мест: 250
Свободных мест: 250

Номер авиарейса: III-556
Название компании: Aeroflot
В Sochi
Из Moskov
Дата полета: 28 7 2023
Время полета: 6:30
Всего мест: 1
Свободных мест: 1
```

Рисунок 13 – Поиск авиарейса фрагменту названия аэропорта прибытия

```
====== ГЛАВНОЕ МЕНЮ ======
1. Добавление нового пассажира
2. Удаление данных о пассажире
3. Просмотр всех зарегистрированных пассажиров
4. Очистка данных о пассажирах
5. Поиск пассажира по «номеру паспорта»
6. Поиск пассажира по его ФИО
7. Добавление нового рейса
8. Удаление авиарейса
9. Удаление всех авиарейсов
10. Просмотр всех авиарейсов
11. Поиск авиарейса по «номеру авиарейса»
12. Поиск авиарейса по фрагментам названия аэропорта прибытия
13. Регистрация продажи билета пассажиру
14. Регистрация возврата билета
15. Выход из программы

Выберите номер главного меню: 13
Введите номер паспорта пассажира, для котроого требуется оформить билет: 4222-444444
Введите номер авиарейса, на который требуется оформить билет. Строка должна быть формата «AAA-NNN»: BBB-000
Количество свободных мест = 3
Билет был успешно продан! Номер вашего билета: 475248688
```

Рисунок 14 – Регистрация продажи билета пассажиру

```
====== ГЛАВНОЕ МЕНЮ ======
1. Добавление нового пассажира
2. Удаление данных о пассажире
3. Просмотр всех зарегистрированных пассажиров
4. Очистка данных о пассажирах
5. Поиск пассажира по «номеру паспорта»
6. Поиск пассажира по его ФИО
7. Добавление нового рейса
8. Удаление авиарейса
9. Удаление всех авиарейсов
10. Просмотр всех авиарейсов
11. Поиск авиарейса по «номеру авиарейса»
12. Поиск авиарейса по фрагментам названия аэропорта прибытия
13. Регистрация продажи билета пассажиру
14. Регистрация возврата билета
15. Выход из программы

Выберите номер главного меню: 14
Паспорт № 4222-444444
Номер рейса: BBB-000
Номер билета: 108680277

Введите номер паспорта пассажира, для котроого требуется вернуть билет: 4222-444444
Зарегестрирован на рейсы:
BBB-000
Введите номер авиарейса, на котором требуется вернуть билет. Строка должна быть формата «AAA-NNN»: BBB-000
Билет был успешно возвращен!
```

Рисунок 15 – Регистрация возврата билета

```
====== ГЛАВНОЕ МЕНЮ ======
1. Добавление нового пассажира
2. Удаление данных о пассажире
3. Просмотр всех зарегистрированных пассажиров
4. Очистка данных о пассажирах
5. Поиск пассажира по «номеру паспорта»
6. Поиск пассажира по его ФИО
7. Добавление нового рейса
8. Удаление авиарейса
9. Удаление всех авиарейсов
10. Просмотр всех авиарейсов
11. Поиск авиарейса по «номеру авиарейса»
12. Поиск авиарейса по фрагментам названия аэропорта прибытия
13. Регистрация продажи билета пассажиру
14. Регистрация возврата билета
15. Выход из программы

Выберите номер главного меню: 15
Program ended with exit code: 0
```

Рисунок 16 – Выход из программы

6. Заключение

Изучены структуры данных и алгоритмы их обработки. При помощи трёх видов структур данных реализована программа для работы с данными аэропорта. Выполненная работа с комплексным применением знаний прошлого семестра демонстрирует освоение курса «Структуры и алгоритмы обработки данных».

7. Список использованной литературы

1. Ключарев, А. А. Структуры и алгоритмы обработки данных: учебное пособие / СПбГУАП. СПб., 2003
2. https://stackoverflow.com
3. https://www.cyberforum.ru
4. https://habr.com
5. https://ru.cppreference.com
6. https://learn.microsoft.com

main.cpp

```
#include <iostream>
#include <string>
#include <iomanip>
#include <vector>
#include <windows.h>

#include "list.h"
#include "hash_table.h"
#include "AVLTree.h"


using namespace std;

bool checkDateFormat(string str) {
	if (str.length() != 10) return false;
	if (str[2] != '.' || str[5] != '.') return false;
	for (int i = 0; i < 10; i++) {
		if (i == 2 || i == 5) continue;
		if (str[i] < '0' || str[i] > '9') return false;
	}
	int day = stoi(str.substr(0, 2));
	int month = stoi(str.substr(3, 2));
	int year = stoi(str.substr(6, 4));
	if (day < 1 || day > 31 || month < 1 || month > 12 || year < 0) return false;
	if (day > 30 && (month == 4 || month == 6 || month == 9 || month == 11))
return false;
	if (day > 28 && month == 2 && ((year % 4 != 0) || (year % 100 == 0 &&
year % 400 != 0))) return false;
	return true;
}
string getDate(string text) {
	string date;
	cout << text;
	getline(cin, date);
	while (!checkDateFormat(date)) {
		cout << "Неверный формат! Введите строку в формате
dd.mm.yyyy: ";
		getline(cin, date);
	}
	return date;
}
```

```
bool checkTimeFormat(string str) {
	if (str.length() != 5) return false;
	if (str[2] != ':') return false;
	for (int i = 0; i < 5; i++) {
		if (i == 2) continue;
		if (str[i] < '0' || str[i] > '9') return false;
	}
	int hours = stoi(str.substr(0, 2));
	int minutes = stoi(str.substr(3, 2));
	if (hours < 0 || hours > 23 || minutes < 0 || minutes > 59) return false;
	return true;
}
string getTime() {
	string time;
	cout << "Введите время отправления: ";
	getline(cin, time);
	while (!checkTimeFormat(time)) {
		cout << "Неверный формат! Введите строку в формате hh:mm: ";
		getline(cin, time);
	}
	return time;
}
bool checkPassFormat(string str) {
	if (str.length() != 11) return false;
	if (str[4] != '-') return false;
	for (int i = 0; i < 11; i++) {
		if (i == 4) continue;
		if (str[i] < '0' || str[i] > '10') return false;
	}
	return true;
}
string getnPass() {
	string nPass;
	cout << "Введите номер паспорта: ";
	getline(cin, nPass);
	while (!checkPassFormat(nPass)) {
		cout << "Неверный формат! Введите строку в формате nnnn-
nnnnnn: ";
		getline(cin, nPass);

	}
	return nPass;
}
bool checknFlightFormat(string str) {
	if (str.length() != 7) return false;
```

```
        if (str[3] != '-') return false;
        for (int i = 0; i < 7; i++) {
                if (i == 3) continue;
                if (i < 3) {
                        if (!isalpha(str[i])) return false;
                }
                else {
                        if (!isdigit(str[i])) return false;
                }
        }
        return true;
}
string getnFlight() {
        std::string nFlight;
        cout << "Введите номер авиарейса: ";
        getline(cin, nFlight);
        while (!checknFlightFormat(nFlight)) {
                cout << "Неверный формат! Введите строку в формате aaa-nnn: ";
                getline(cin, nFlight);
        }
        return nFlight;
}
int getnSeats() {
        int nSeats;
        cout << "Введите количество мест в самолете: ";
        cin >> nSeats;
        while (nSeats < 0) {
                cout << "Некорректное количество мест! Введите корректное
количество мест в самолете: ";
                cin >> nSeats;
        }
        return nSeats;
}
int getnFreeSeats(int nSeats) {
        int nFSeats;
        cout << "Введите количество свободных мест: ";
        cin >> nFSeats;
        while (nFSeats > nSeats) {
                cout << "Количество свободных мест слишком велико! Введите
корректное количество свободных мест: ";
                cin >> nFSeats;
        }
        return nFSeats;
}
bool checknTickFormat(string str) {
```

```
        if (str.length() != 9) return false;
        for (int i = 0; i < 9; i++) {
                if (!isdigit(str[i])) return false;
        }
        return true;
}
string getnTick() {
        string nTick;
        cout << "Введите номер билета: ";
        getline(cin, nTick);
        while (!checknTickFormat(nTick)) {
                cout << "Номер авиабилета должен содержать 9 цифр! Введите
корректный номер авиабилета: ";
                getline(cin, nTick);
        }
        return nTick;
}
node* addFlight(node* tree, string nFl, string nA, string dep, string dest, string
dateDep, string timeDep, int nSeats, int nFSeats) {
        bool isUniq = true;
        checkUniq(tree, getNumber(nFl), &isUniq);
        if (!isUniq) {
                cout << "Номер рейса " << nFl << " неуникален" << endl;
                return tree;
        }
        int k = getNumber(nFl);
        tree = addVert(tree, k, nFl, nA, dep, dest, dateDep, timeDep, nSeats,
nFSeats);
        cout << "Добавление рейса " << nFl << " прошло успешно!" << endl;
        return tree;
}
node* inputFlight(node* tree) {
        string nFl, nA, dep, dest, dateDep, timeDep;
        int nSeats, nFSeats;
        cin.ignore();
        cout << setw(25) << "Добавление авиарейса!" << endl;
        nFl = getnFlight();
        cout << "Введите название авиакомпании: ";
        getline(cin, nA);
        cout << "Введите аэропорт отправления: ";
        getline(cin, dep);
        cout << "Введите аэропорт назначения: ";
        getline(cin, dest);
        dateDep = getDate("Введите дату отправления: ");
        timeDep = getTime();
```

```
	nSeats = getnSeats();
	nFSeats = getnFreeSeats(nSeats);
	tree = addFlight(tree, nFl, nA, dep, dest, dateDep, timeDep, nSeats,
nFSeats);
	return tree;
}
vector<Elem*> addPassanger(vector<Elem*> hT, string nPass, string fName,
string bD, string info) {
	if (!isUniqPassword(hT, nPass)) {
		cout << "Номер паспорта " << nPass << " неуникален" << endl;
		return hT;
	}
	hT = AddElem(hT, nPass, fName, bD, info);
	cout << "Добавление пассажира с №паспорта " << nPass << " прошло
успешно!" << endl;
	return hT;
}
vector<Elem*> inputPassanger(vector<Elem*> hT) {
	string nPass, fName, bD, info;
	cin.ignore();
	cout << setw(25) << "Добавление пассажира!" << endl;
	nPass = getnPass();
	cout << "Введите ФИО: ";
	getline(cin, fName);
	bD = getDate("Введите дату рождения: ");
	cout << "Введите дату и место выдачи паспорта: ";
	getline(cin, info);
	hT = addPassanger(hT, nPass, fName, bD, info);
	return hT;
}
cycList* saleRegistration(cycList* lst, vector<Elem*> hT, node* tree, string
nPass, string nFl, string nTick) {
	bool isUniq = true;
	checkUniq(tree, getNumber(nFl), &isUniq);
	if (isUniq) {
		cout << "Отсутствует рейс № " << nFl << endl;
		return lst;
	}
	if (isUniqPassword(hT, nPass)) {
		cout << "Отсутствует пассажир с №паспорта " << nPass << endl;
		return lst;
	}
	if (!isUniqAll(lst, nTick, nPass, nFl)) {
		cout << "Номер билета неуникален или этот пассажир с
№паспорта " << nPass << " уже зарегистрирован на рейс " << nFl << endl;
```

```
            return lst;
        }
        int count = 0;
        countFreeSeats(tree, getNumber(nFl), &isUniq, &count);
        if (count == 0) {
            cout << "Отсутствуют свободные места на  рейс № " << nFl <<
endl;
            return lst;
        }
        lst = addelem(lst, nPass, nFl, nTick);
        cout << "Регистрация пассажира с № паспорта " << nPass << " на рейс
№" << nFl << " прошла успешно!" << endl;
        cout << "Оставшееся количество свободных мест на рейсе № " << nFl
<< " равно " << count - 1 << endl;
        return lst;
}
cycList* inputRegistration(cycList* lst, vector<Elem*> hT, node* tree) {
        cout << setw(24) << "Регистрация на рейс!" << endl;
        string nPass, nFl, nTick;
        cin.ignore();
        nPass = getnPass();
        nFl = getnFlight();
        nTick = getnTick();
        lst = saleRegistration(lst, hT, tree, nPass, nFl, nTick);
        return lst;
}
cycList* returnRegistration(cycList* lst, node* tree, string nTick) {
        if (isUniqAll(lst, nTick)) {
            cout << "Нет зарегистрированного пассажира с № билета " <<
nTick << endl;
            return lst;
        }
        bool flag = false;;
        int count = 0;
        string nFl = getNumberFlight(lst, nTick);
        returnFreeSeats(tree, getNumber(nFl), &flag, &count);
        lst = deletelem(lst, nTick);
        cout << "Возврат регистрации пассажира с № билета " <<  nTick << "
прошел успешно!" << endl;
        cout << "Оставшееся количество свободных мест на рейсе № " << nFl
<< " равно " << count << endl;
        return lst;
}
node* deleteFlight(node* tree, cycList* lst, string nFl) {
        bool isUniq = true;
```

```
	checkUniq(tree, getNumber(nFl), &isUniq);
	if (isUniq) {
		cout << "Номер рейса " << nFl << " отсутствует" << endl;
		return tree;
	}
	if (isRegisterOfFlight(lst, nFl)) {
		cout << "На рейс № " << nFl << " зарегистрированы. Удаление
невозможно!" << endl;
		return tree;
	}
	tree = removeVert(tree, getNumber(nFl));
	cout << "Удаление рейса " << nFl << " прошло успешно!" << endl;
	return tree;
}
vector<Elem*> deletePassanger(vector<Elem*> hT, cycList* lst, string nPass) {
	if (isUniqPassword(hT, nPass)) {
		cout << "Пассажир с  №паспорта " << nPass << " отсутствует" <<
endl;
		return hT;
	}
	if (isRegister(lst, nPass)) {
		cout << "Пассажир с №паспорта " << nPass << " уже
зарегистрирован. Удаление невозможно!" << endl;
		return hT;
	}
	hT = deleteElem(hT, nPass);
	cout << "Удаление пассажира с №паспорта " << nPass << " прошло
успешно!" << endl;
	return hT;
}
void findByNumberOfPassword(vector<Elem*> hT, cycList* lst, string nPass) {
	if (isUniqPassword(hT, nPass)) {
		cout << "Пассажир с  №паспорта " << nPass << " отсутствует" <<
endl;
		return;
	}
	findElemByPass(hT, nPass);
	findelemByPass(lst, nPass);
	return;
}
void findPassangersByName(vector<Elem*> hT, string fName) {
	findElemByName(hT, fName);
}
void findFlightByNumber(node* tree, cycList* lst, string nFl) {
	bool isUniq = true;
```

```
	checkUniq(tree, getNumber(nFl), &isUniq);
	if (isUniq) {
		cout << "Номер рейса " << nFl << " отсутствует" << endl;
		return;
	}
	findElem(tree, getNumber(nFl), &isUniq);
	findelemByNumber(lst, nFl);
}
void findFlightByDestination(node* tree, string text) {
	bool isFind = false;
	findByBM(tree, text, &isFind);
	if (!isFind) cout << "Не найдено авиарейсрв с таким пунктом
назначения!" << endl;
}
void showPassangers(vector<Elem*> hT) {
	bool f = false;
	viewTable(hT, &f);
	if (!f) {
		cout << "Нет зарегистрированных пассажиров!" << endl;
	}
}
vector<Elem*> clearPassangers(vector<Elem*> hT, cycList* lst) {
	bool noDel = false;
	bool isDel = false;
	for (auto it = hT.begin(); it != hT.end(); ++it) {
		if (*it == nullptr) continue;
		if (isRegister(lst, (*it)->numberOfPassword)) {
			noDel = true;
			continue;
		}
		isDel = true;
		*it = clearElem(*it);
	}
	if (!isDel && noDel) {
		cout << "Не были удалены данные о пассажирах, так как они
зарегистрированы на рейс!" << endl;
	}
	else if (noDel) {
		cout << "Не были удалены некоторые данные о пассажирах, так
как они зарегистрированы на рейс!" << endl;
	}
	else if (!isDel && !noDel) {
		cout << "Нет данных о пассажирах для удаления!" << endl;
	}
	else {
```

```
			cout << "Успешно удалены все данные о пассажирах!" << endl;
		}
		return(hT);
}
void showFlight(node* tree) {
		bool f = false;
		ShowElements(tree, &f);
		if (!f) {
			cout << "Нет зарегистрированных авиарейсов!" << endl;
		}
}
void findEl4Delete(node* tree, cycList* lst, vector<int>* res, bool* isDel, bool*
noDel) {
		if (tree) {
			findEl4Delete(tree->left, lst, res, isDel, noDel);
			findEl4Delete(tree->right, lst, res, isDel, noDel);
			if (!isRegisterOfFlight(lst, tree->numberOfFlight)) {
				*isDel = true;
				res->push_back(getNumber(tree->numberOfFlight));
				return;
			}
			*noDel = true;
		}
		return;
}
node* clearFlights(node* tree, cycList* lst) {
		bool noDel = false;
		bool isDel = false;
		vector<int> num4delete;
		findEl4Delete(tree, lst, &num4delete, &isDel, &noDel);
		for (auto it = num4delete.begin(); it != num4delete.end(); ++it) tree =
removeVert(tree, *it);

		if (!isDel && noDel) {
			cout << "Не были удалены данные об авиарейсах, так как на них
зарегистрированы пассажиры!" << endl;
		}
		else if (noDel) {
			cout << "Не были удалены некоторые данные об авиарейсах, так
как на них зарегистрированы пассажиры!" << endl;
		}
		else if (!isDel && !noDel) {
			cout << "Нет данных об авиарейсах для удаления!" << endl;
		}
		else {
```

```
            cout << "Успешно удалены все данные об авиарейсах!" << endl;
        }
        return tree;
}
void showRegisterPassangers(cycList* lst) {
        listprint(lst);
}
void clearAll(cycList* lst, vector<Elem*>* hT, node* tree) {
        clearTree(tree);
        clearElems(*hT);
        clearList(lst);
        hT->clear();
        hT->shrink_to_fit();
}
int main(){
        setlocale(LC_ALL, "Rus");
        SetConsoleCP(1251);
        SetConsoleOutputCP(1251);
        vector<Elem*> hTablePassangers(n);
        node* avlTreeFlights = nullptr;
        string numberPassword, numberFlight, numberTicket, fullName, text;
        cycList* listRegistration = nullptr;
        int click;
        bool stop = false;

        cout << setw(80) << "Информационная система продажа авиабилетов.
`Made by Артемий Хорошилов`" << endl;

        while (!stop) {
                cout << endl << setw(28) << "Меню управления системой" <<
endl;{
                cout << "1. Добавление авиарейса" << endl;
                cout << "2. Добавление пассажира" << endl;
                cout << "3. Регистрация продажи пассажиру авиабилета" << endl;
                cout << "4. Удаление данных о пассажире" << endl;
                cout << "5. Удаление сведений об авиарейсе" << endl;
                cout << "6. Регистрация возврата пассажиром авиабилета" <<
endl;
                cout << "7. Просмотр всех зарегистрированных пассажиров" <<
endl;
                cout << "8. Просмотр всех авиарейсов" << endl;
                cout << "9. Просмотр зарегистрированных на авиарейсы
пассажиров" << endl;
                cout << "10. Поиск пассажира по «номеру паспорта»" << endl;
                cout << "11. Поиск пассажира по его ФИО" << endl;
```

```
            cout << "12. Поиск авиарейса по «номеру авиарейса»" << endl;
            cout << "13. Поиск авиарейса по фрагментам названия аэропорта
прибытия" << endl;
            cout << "14. Очистка данных о пассажирах" << endl;
            cout << "15. Очистка данных об авиарейсах" << endl;
            cout << "0. Выход из системы" << endl; }
            cout << "Введите номер пункта меню: ";
            cin >> click;
            switch (click) {
            case 1:
                  avlTreeFlights = inputFlight(avlTreeFlights);
                  system("pause");
                  //system("cls");
                  break;
            case 2:
                  hTablePassangers = inputPassanger(hTablePassangers);
                  system("pause");
                  //system("cls");
                  break;
            case 3:
                  listRegistration = inputRegistration(listRegistration,
hTablePassangers, avlTreeFlights);
                  system("pause");
                  //system("cls");
                  break;
            case 4:
                  cout << setw(31) << "Удаление данных о пассажире" <<
endl;
                  cin.ignore();
                  numberPassword = getnPass();
                  hTablePassangers = deletePassanger(hTablePassangers,
listRegistration, numberPassword);
                  system("pause");
                  //system("cls");
                  break;
            case 5:
                  cout << setw(34) << "Удаление сведений об авиарейсе" <<
endl;
                  cin.ignore();
                  numberFlight = getnFlight();
                  avlTreeFlights = deleteFlight(avlTreeFlights, listRegistration,
numberFlight);
                  system("pause");
                  //system("cls");
                  break;
```

```
            case 6:
                cout << setw(46) << "Регистрация возврата пассажиром
авиабилета" << endl;
                cin.ignore();
                numberTicket = getnTick();
                listRegistration = returnRegistration(listRegistration,
avlTreeFlights, numberTicket);
                system("pause");
                //system("cls");
                break;
            case 7:
                cout << setw(47) << "Просмотр всех зарегистрированных
пассажиров" << endl;
                showPassangers(hTablePassangers);
                system("pause");
                //system("cls");
                break;
            case 8:
                cout << setw(28) << "Просмотр всех авиарейсов" << endl;
                showFlight(avlTreeFlights);
                system("pause");
                //system("cls");
                break;
            case 9:
                cout << setw(55) << "Просмотр зарегистрированных на
авиарейсы пассажиров" << endl;
                showRegisterPassangers(listRegistration);
                system("pause");
                //system("cls");
                break;
            case 10:
                cout << setw(40) << "Поиск пассажира по «номеру
паспорта»" << endl;
                cin.ignore();
                numberPassword = getnPass();
                findByNumberOfPassword(hTablePassangers, listRegistration,
numberPassword);
                system("pause");
                //system("cls");
                break;
            case 11:
                cout << setw(30) << "Поиск пассажира по его ФИО" << endl;
                cin.ignore();
                cout << "Введите ФИО пассажира для поиска: ";
                getline(cin, fullName);
```

```
                findPassangersByName(hTablePassangers, fullName);
                system("pause");
                //system("cls");
                break;
            case 12:
                cout << setw(41) << "Поиск авиарейса по «номеру
авиарейса»" << endl;
                cin.ignore();
                numberFlight = getnFlight();
                findFlightByNumber(avlTreeFlights, listRegistration,
numberFlight);
                system("pause");
                //system("cls");
                break;
            case 13:
                cout << setw(61) << "Поиск авиарейса по фрагментам
названия аэропорта прибытия" << endl;
                cin.ignore();
                cout << "Введите название аэропорта назначения: ";
                getline(cin, text);
                findFlightByDestination(avlTreeFlights, text);
                system("pause");
                //system("cls");
                break;
            case 14:
                cout << setw(31) << "Очистка данных о пассажирах" <<
endl;
                hTablePassangers = clearPassangers(hTablePassangers,
listRegistration);
                system("pause");
                //system("cls");
                break;
            case 15:
                cout << setw(32) << "Очистка данных об авиарейсах" <<
endl;
                avlTreeFlights = clearFlights(avlTreeFlights, listRegistration);
                system("pause");
                //system("cls");
                break;
            case 0:
                cout << setw(34) << "Произведен выход из программы!" <<
endl;
                clearAll(listRegistration, &hTablePassangers, avlTreeFlights);
                stop = true;
                break;
```

```
            default:
                cout << "Введен некоректный номер!" << endl;
                system("pause");
                system("cls");
                break;
            }
        }
        system("pause");
}
```

list.h

```
#include <iostream>
#include <string>
#include <iomanip>
#include <vector>

using namespace std;

struct cycList
{
        string numberOfPassword;
        string numberOfFlight;
        string numberOfTicket;
        int field;
        cycList* next;
        cycList* prev;
};

struct sortli {
        int count = 0;
        string numberOfPassword;
        string numberOfFlight;
        string numberOfTicket;
        int field;
};

cycList* init(string nPass, string nFl, string nTick);
cycList* addelem(cycList* lst, string nPass, string nFl, string nTick);
cycList* deletelemptr(cycList* lst);
cycList* deletelem(cycList* lst, string nTick);
void listprint(cycList* lst);
void listprintr(cycList* lst);
int findMax(cycList* lst);
cycList* sortlist(cycList* lst);
```

```
bool isUniqAll(cycList* lst, string nTick, string nFl = "", string nPass = "");
string getNumberFlight(cycList* lst, string nTick);
bool isRegisterOfFlight(cycList* lst, string nFl);
bool isRegister(cycList* lst, string nPass);
void findelemByPass(cycList* lst, string nPass);
void findelemByNumber(cycList* lst, string nFl);
void clearList(cycList* lst);
```

list.cpp

```
#include"list.h"

cycList* init(string nPass, string nFl, string nTick)  // a- значение первого узла
{
	cycList* lst = new cycList;
	// выделение памяти под корень списка
	lst->numberOfFlight = nFl;
	lst->numberOfPassword = nPass;
	lst->numberOfTicket = nTick;
	lst->next = lst; // указатель на следующий узел
	lst->prev = lst; // указатель на предыдущий узел
	return(lst);
}
cycList* addelem(cycList* lst, string nPass, string nFl, string nTick)  // number - значение узла
{
	if (lst == nullptr) {
		lst = init(nPass, nFl, nTick);
		return(lst);
	}
	cycList* temp = new cycList, * p;
	p = lst->next; // сохранение указателя на следующий узел
	lst->next = temp; // предыдущий узел указывает на создаваемый
	temp->numberOfFlight = nFl;
	temp->numberOfPassword = nPass;
	temp->numberOfTicket = nTick;
	temp->next = p; // созданный узел указывает на следующий узел
	temp->prev = lst; // созданный узел указывает на предыдущий узел
	p->prev = temp;
	return(sortlist(temp));
}
cycList* deletelemptr(cycList* lst)
{
	cycList* prev, * next;
	prev = lst->prev; // узел, предшествующий lst
```

```
        next = lst->next; // узел, следующий за lst
        if (prev == lst && next == lst) {
                free(lst); // освобождаем память удаляемого элемента
                return(nullptr);
        }
        prev->next = lst->next; // переставляем указатель
        next->prev = lst->prev; // переставляем указатель
        free(lst); // освобождаем память удаляемого элемента
        return(prev);
}
cycList* deletelem(cycList* lst, string nTick) {
        if (lst == nullptr) {
                return lst;
        }
        cycList* p;
        bool isDel = false;
        if (lst->numberOfTicket == nTick) {
                p = lst->prev;
        }
        else {
                p = lst;
        }
        do {
                if (lst->numberOfTicket == nTick) {
                        lst = deletelemptr(lst);
                        isDel = true;
                }
                else {
                        lst = lst->next; // переход к следующему узлу
                }
        } while (p != lst && lst != nullptr); // условие окончания обхода
        if (!isDel) {
                cout << "No value for delete" << endl;
        }
        return sortlist(lst);
}
void listprint(cycList* lst)
{
        if (lst == nullptr) {
                cout << "Нет пассажиров зарегистрированных на рейсы!" << endl;
                return;
        }
        cout << setw(15) << "Номер билета" << setw(18) << "Номер паспорта"
<< setw(14) << "Номер рейса" << endl;
        cycList* p;
```

```
	p = lst;
	do {
		cout << setw(15) << p->numberOfTicket << setw(18) << p->numberOfPassword << setw(14) << p->numberOfFlight << endl; // вывод значения элемента p
		p = p->next; // переход к следующему узлу
	} while (p != lst); // условие окончания обхода
	cout << endl;
}
void listprintr(cycList* lst)
{
	if (lst == nullptr) {
		return;
	}
	cout << setw(15) << "Номер билета" << setw(18) << "Номер паспорта" << setw(14) << "Номер рейса" << endl;
	cycList* p;
	p = lst;
	do {
		p = p->prev;  // переход к предыдущему узлу
		cout << setw(15) << p->numberOfTicket << setw(18) << p->numberOfPassword << setw(14) << p->numberOfFlight << endl; // вывод значения элемента p
	} while (p != lst); // условие окончания обхода
	cout << endl;
}
int findMax(cycList* lst) {
	int max = 0;
	cycList* p;
	p = lst;
	do {
		if (stoi(p->numberOfTicket) > max) {
			max = stoi(p->numberOfTicket);
		}
		p = p->next;
	} while (p != lst); // условие окончания обхода
	return max;
}
cycList* sortlist(cycList* lst) {
	if (lst == nullptr) {
		return lst;
	}
	vector <sortli> additional(findMax(lst) + 1);
	cycList* p;
	p = lst;
```

```
        int index;
        do {
                index = stoi(p->numberOfTicket);
                additional[index].count += 1;
                additional[index].numberOfFlight = p->numberOfFlight;
                additional[index].numberOfPassword = p->numberOfPassword;
                additional[index].numberOfTicket = p->numberOfTicket;
                p = p->next;  // переход к предыдущему узлу
        } while (p != lst); // условие окончания обхода
        for (int i = 0; i < additional.size(); i++) {
                if (additional[i].count > 0) {
                        lst->numberOfFlight = additional[i].numberOfFlight;
                        lst->numberOfPassword = additional[i].numberOfPassword;
                        lst->numberOfTicket = additional[i].numberOfTicket;
                        lst = lst->next;
                }
        }
        return lst;
}
struct cycList* swap(struct cycList* lst1, struct cycList* lst2, struct cycList* head)
{
        // Возвращает новый корень списка
        struct cycList* prev1, * prev2, * next1, * next2;
        prev1 = lst1->prev; // узел предшествующий lst1
        prev2 = lst2->prev; // узел предшествующий lst2
        next1 = lst1->next; // узел следующий за lst1
        next2 = lst2->next; // узел следующий за lst2
        if (lst2 == next1)  // обмениваются соседние узлы
        {
                lst2->next = lst1;
                lst2->prev = prev1;
                lst1->next = next2;
                lst1->prev = lst2;
                next2->prev = lst1;
                prev1->next = lst2;
        }
        else if (lst1 == next2)  // обмениваются соседние узлы
        {
                lst1->next = lst2;
                lst1->prev = prev2;
                lst2->next = next1;
                lst2->prev = lst1;
                next1->prev = lst2;
                prev2->next = lst1;
        }
```

```
        else  // обмениваются отстоящие узлы
        {
                prev1->next = lst2;
                lst2->next = next1;
                prev2->next = lst1;
                lst1->next = next2;
                lst2->prev = prev1;
                next2->prev = lst1;
                lst1->prev = prev2;
                next1->prev = lst2;
        }
        if (lst1 == head)
                return(lst2);
        if (lst2 == head)
                return(lst1);
        return(head);
}
bool isUniqAll(cycList* lst, string nTick, string nPass, string nFl) {
        if (lst == nullptr) {
                return true;
        }
        cycList* p;
        p = lst;
        do {
                if (p->numberOfTicket == nTick || (p->numberOfPassword == nPass
&& p->numberOfFlight == nFl)) return false;
                p = p->next; // переход к следующему узлу
        } while (p != lst); // условие окончания обхода
        return true;
}
string getNumberFlight(cycList* lst, string nTick) {
        cycList* p;
        p = lst;
        do {
                if (p->numberOfTicket == nTick) return p->numberOfFlight;
                p = p->next; // переход к следующему узлу
        } while (p != lst); // условие окончания обхода
}
bool isRegisterOfFlight(cycList* lst, string nFl) {
        if (lst == nullptr) {
                return false;
        }
        cycList* p;
        p = lst;
        do {
```

```
            if (p->numberOfFlight == nFl) return true;
            p = p->next; // переход к следующему узлу
        } while (p != lst); // условие окончания обхода
        return false;
}
bool isRegister(cycList* lst, string nPass) {
        if (lst == nullptr) {
            return false;
        }
        cycList* p;
        p = lst;
        do {
            if (p->numberOfPassword == nPass) return true;
            p = p->next; // переход к следующему узлу
        } while (p != lst); // условие окончания обхода
        return false;
}
void findelemByPass(cycList* lst, string nPass) {
        bool isFind = false;
        if (lst == nullptr) {
            cout << "Пассажир не зарегистрирован на рейсы!" << endl;
            return;
        }
        cycList* p;
        p = lst;
        do {
            if (p->numberOfPassword == nPass) {
                if (!isFind) {
                    cout << setw(15) << "Номер билета" << setw(18) <<
"Номер паспорта" << setw(14) << "Номер рейса" << endl;
                    isFind = true;
                }
                cout << setw(15) << p->numberOfTicket << setw(18) << p-
>numberOfPassword << setw(14) << p->numberOfFlight << endl;
            }
            p = p->next; // переход к следующему узлу
        } while (p != lst); // условие окончания обхода
        if (!isFind) cout << "Пассажир не зарегистрирован на рейсы!" << endl;
        return;
}
void findelemByNumber(cycList* lst, string nFl) {
        bool isFind = false;
        if (lst == nullptr) {
            cout << "На рейс не зарегистрированны пассажиры!" << endl;
            return;
```

```
	}
	cycList* p;
	p = lst;
	do {
		if (p->numberOfFlight == nFl) {
			if (!isFind) {
				cout << setw(15) << "Номер билета" << setw(18) <<
"Номер паспорта" << setw(14) << "Номер рейса" << endl;
				isFind = true;
			}
			cout << setw(15) << p->numberOfTicket << setw(18) << p-
>numberOfPassword << setw(14) << p->numberOfFlight << endl;
		}
		p = p->next; // переход к следующему узлу
	} while (p != lst); // условие окончания обхода
	if (!isFind) cout << "На рейс не зарегистрированны пассажиры!" << endl;
	return;
}

void clearList(cycList* head) {
	cycList* curr = head;
	cycList* next = nullptr;
	while (curr != nullptr) {
		next = curr->next;
		delete curr;
		curr = next;
		if (curr == head) {
			break;  // список замкнут, выходим из цикла
		}
	}
}
```

AVLTree.h

```
#include <iostream>
#include <string>
#include <iomanip>
#include <vector>

using namespace std;


struct node
{
	int data;
```

```
        int height;
        int depth;
        string numberOfFlight;
        string nameOfAirline;
        string departure;
        string destination;
        string dateOfDeparture;
        string timeOfDeparture;
        int numberOfSeats;
        int numberOfFreeSeats;
        node* left;
        node* right;
        node(int k, string nFl, string nA, string dep, string dest, string dateDep,
string timeDep, int nSeats, int nFSeats) {
                data = k; left = right = nullptr; height = 1; depth = 1; numberOfFlight
= nFl; nameOfAirline = nA; departure = dep; destination = dest;
                dateOfDeparture = dateDep; timeOfDeparture = timeDep;
numberOfSeats = nSeats; numberOfFreeSeats = nFSeats;
        }
};

unsigned char height(node* tree);
int balanceFactor(node* tree);
void fixHeight(node* tree);
void fixDepth(node* tree);
node* smallRotateRight(node* tree);
node* smallRotateLeft(node* tmp);
node* balance(node* tree);
node* addVert(node* tree, int k, string nFl, string nA, string dep, string dest, string
dateDep, string timeDep, int nSeats, int nFSeats);
node* findMin(node* tree);
node* removeMin(node* tree);
node* removeVert(node* tree, int k);
void ShowTree(node* tree, int indent);
void ShowElements(node* tree, bool* f);
int boyerMoore(string text, string pattern);
void findByBM(node* tree, string pattern, bool* f);
void findElem(node* tree, int data, bool* flag);
node* clearTree(node* tree);
int getNumber(string str);
bool checkUniq(node* tree, string nFl, bool f = true);
void checkUniq(node* tree, int k, bool* flag);
void countFreeSeats(node* tree, int k, bool* flag, int* count);
void returnFreeSeats(node* tree, int k, bool* flag, int* count);
```

AVLTree.cpp

```
#pragma once
#include "AVLTree.h"

unsigned char height(node* tree)
{
	if (tree)
		return tree->height;
	else
		return 0;
}
int balanceFactor(node* tree)
{
	return height(tree->right) - height(tree->left);
}
void fixHeight(node* tree)
{
	int heightLeft = height(tree->left);
	int heughtRight = height(tree->right);
	tree->height = (heightLeft > heughtRight ? heightLeft : heughtRight) + 1;
}
void fixDepth(node* tree) {

	if (!tree) return;
	if (tree->left)
		tree->left->depth = tree->depth + 1;
	if (tree->right)
		tree->right->depth = tree->depth + 1;
	fixDepth(tree->left);
	fixDepth(tree->right);
}
node* smallRotateRight(node* tree)
{
	node* tmp = tree->left;
	tree->left = tmp->right;
	tmp->right = tree;
	fixHeight(tree);
	fixHeight(tmp);
	return tmp;
}
node* smallRotateLeft(node* tmp)
{
	node* tree = tmp->right;
	tmp->right = tree->left;
```

```
        tree->left = tmp;
        fixHeight(tmp);
        fixHeight(tree);
        return tree;
}
node* balance(node* tree)
{
        fixHeight(tree);
        if (balanceFactor(tree) == 2)
        {
                if (balanceFactor(tree->right) < 0)
                        tree->right = smallRotateRight(tree->right);
                node* tmp = smallRotateLeft(tree);
                return tmp;
        }
        if (balanceFactor(tree) == -2)
        {
                if (balanceFactor(tree->left) > 0)
                        tree->left = smallRotateLeft(tree->left);
                node* tmp = smallRotateRight(tree);
                return tmp;
        }
        return tree;
}
node* addVert(node* tree, int k, string nFl, string nA, string dep, string dest, string
dateDep, string timeDep, int nSeats, int nFSeats)
{
        if (!tree) return new node(k, nFl, nA, dep, dest, dateDep, timeDep, nSeats,
nFSeats);
        if (k < tree->data)
                tree->left = addVert(tree->left, k, nFl, nA, dep, dest, dateDep,
timeDep, nSeats, nFSeats);
        else if (k > tree->data)
                tree->right = addVert(tree->right, k, nFl, nA, dep, dest, dateDep,
timeDep, nSeats, nFSeats);
        else {
                cout << "The vertex is already in the tree" << endl;
                return tree;
        }
        return balance(tree);
}
node* findMin(node* tree)
{
        return tree->left ? findMin(tree->left) : tree;
}
```

```
node* removeMin(node* tree)
{
	if (tree->left == 0)
		return tree->right;
	tree->left = removeMin(tree->left);
	return balance(tree);
}
node* removeVert(node* tree, int k)
{
	if (!tree) return 0;
	if (k < tree->data)
		tree->left = removeVert(tree->left, k);
	else if (k > tree->data)
		tree->right = removeVert(tree->right, k);
	else
	{
		node* q = tree->left;
		node* r = tree->right;
		delete tree;
		if (!r) return q;
		node* min = findMin(r);
		min->right = removeMin(r);
		min->left = q;
		return balance(min);
	}
	return balance(tree);
}
void ShowTree(node* tree, int indent) {
	if (tree) {
		ShowTree(tree->right, indent + 1);
		for (int i = 1; i <= indent; i++) cout << "   ";
		cout << tree->numberOfFlight << endl;
		ShowTree(tree->left, indent + 1);
	}
}
void ShowElements(node* tree, bool* f) {
	if (tree) {
		ShowElements(tree->left, f);
		ShowElements(tree->right, f);
		if (!*f) {
			cout << setw(14) << "Номер рейса" << setw(21) << "Имя
авиакомпании" << setw(20) << "Пункт вылета" << setw(20) << "Пункт
назначения" << setw(29) << "Количество свободных мест" << endl;
			*f = true;
		}
```

```
            cout << setw(14) << tree->numberOfFlight;
            cout << setw(21) << tree->nameOfAirline;
            cout << setw(20) << tree->departure;
            cout << setw(20) << tree->destination;
            cout << setw(29) << tree->numberOfFreeSeats << endl;
        }
}
int boyerMoore(string text, string pattern) {
        int n = text.length(); // длина текстовой строки
        int m = pattern.length(); // длина искомой подстроки
        if (m == 0) return 0; // если искомая подстрока пустая, возвращаем
индекс 0
        if (n < m) return -1; // если текстовая строка короче искомой подстроки,
то подстрока в ней не найдется
        vector<int> jump(65536); // контейнер для таблицы смещений
        for (int i = 0; i < 65536; i++) jump[i] = m; // инициализируем контейнер
максимальным значением смещения
        for (int i = 0; i < m - 1; i++) jump[(unsigned short)pattern[i]] = m - i - 1; //
вычисляем смещение для каждого символа искомой подстроки
        int i = m - 1; // начинаем сравнение с конца искомой подстроки
        int j = m - 1; // начинаем сравнение с конца искомой подстроки
        while (i < n) { // пока не достигнут конец текстовой строки
            if (text[i] == pattern[j]) { // если символы совпадают
                if (j == 0) return i; // если все символы искомой подстроки
найдены, возвращаем индекс первого символа найденной подстроки
                i--; // переходим к следующим символам в обеих строках
                j--;
            }
            else { // если символы не совпадают
                i += jump[(unsigned short)text[i]]; // сдвигаем индекс начала
сравнения на максимально возможное смещение
                j = m - 1; // начинаем сравнение с конца искомой подстроки
                if (i > n - 1) break; // если индекс вышел за границы
текстовой строки, то выходим из цикла
            }
        }
        return -1; // если подстрока не найдена, возвращаем -1
}
// Функция для выполнения алгоритма Бойера-Мура
void findByBM(node* tree, string pattern, bool* f) {
        if (tree) {
            findByBM(tree->left, pattern, f);
            findByBM(tree->right, pattern, f);
            int index = boyerMoore(tree->destination, pattern);
            if (index == -1) return;
```

```
            if (!*f) cout << setw(18) << "Номер рейса" << setw(25) << "Пункт
назначения" << setw(20) << "Дата вылета"  << setw(20) << "Время вылета"  <<
endl;
            *f = true;
            cout << setw(18) << tree->numberOfFlight;
            cout << setw(25) << tree->destination;
            cout << setw(20) << tree->dateOfDeparture;
            cout << setw(20) << tree->timeOfDeparture << endl;
        }
}
void findElem(node* tree, int k, bool* flag) {
        if (*flag)
            return;
        if (!tree) {
            return;
        }
        if (tree->data == k) {
            *flag = true;
            cout << setw(28) << "Результаты поиска" << endl;
            cout << setw(15) << "Номер рейса: "   << tree->numberOfFlight <<
endl;
            cout << setw(20) << "Имя авиакомпании: "   << tree-
>nameOfAirline << endl;
            cout << setw(21) << "Пункт отправления: "  << tree->departure <<
endl;
            cout << setw(20) << "Пункт назначения: "  << tree->destination <<
endl;
            cout << setw(15) << "Дата вылета: "  << tree->dateOfDeparture <<
endl;
            cout << setw(16) << "Время вылета: "  << tree->timeOfDeparture <<
endl;
            cout << setw(25) << "Общее количество мест: "  << tree-
>numberOfSeats << endl;
            cout << setw(29) << "Количество свободных мест: "  << tree-
>numberOfFreeSeats << endl;
            return;
        }
        else if (tree->data < k) {
            findElem(tree->right, k, flag);
        }
        else {
            findElem(tree->left, k, flag);
        }
}
int getNumber(string str){
```

```
        return stoi(str.substr(4, 6)) + str[0] + str[1]*str[1] + str[2]* str[2]* str[2];
}
node* clearTree(node* tree) {
        if (tree) {
                clearTree(tree->left);
                clearTree(tree->right);
                delete tree;
        }
        return nullptr;
}
void checkUniq(node* tree, int k, bool* flag) {
        if (!*flag)
                return;
        if (!tree) {
                return;
        }
        if (tree->data == k) {
                *flag = false;
                return;
        }
        else if (tree->data < k) {
                checkUniq(tree->right, k, flag);
        }
        else {
                checkUniq(tree->left, k, flag);
        }
}
bool checkUniq(node* tree, string nFl, bool f) {
        if (tree) {
                f = checkUniq(tree->left, nFl, f);
                f = checkUniq(tree->right, nFl, f);
                if (getNumber(nFl) == getNumber(tree->numberOfFlight) && nFl ==
tree->numberOfFlight) return (f && false);
                return f;
        }
        return f;
}
void countFreeSeats(node* tree, int k, bool* flag, int* count) {
        if (*flag)
                return;
        if (!tree) {
                return;
        }
        if (tree->data == k) {
                *flag = true;
```

```
            if (tree->numberOfFreeSeats == 0) return;
            *count = tree->numberOfFreeSeats;
            tree->numberOfFreeSeats -= 1;
            return;
        }
        else if (tree->data < k) {
            countFreeSeats(tree->right, k, flag, count);
        }
        else {
            countFreeSeats(tree->left, k, flag, count);
        }
}
void returnFreeSeats(node* tree, int k, bool* flag, int* count) {
        if (*flag)
            return;
        if (!tree) {
            return;
        }
        if (tree->data == k) {
            *flag = true;
            tree->numberOfFreeSeats += 1;
            *count = tree->numberOfFreeSeats;
            return;
        }
        else if (tree->data < k) {
            returnFreeSeats(tree->right, k, flag, count);
        }
        else {
            returnFreeSeats(tree->left, k, flag, count);
        }
}
```

hash-table.h

```
#pragma once
#include <iostream>
#include <string>
#include <vector>
#include <iomanip>

using namespace std;

const int n = 100;

struct Elem {
```

```
        string numberOfPassword;
        string Info;
        string bDay;
        string fullName;
        Elem* next;
        Elem() { next = nullptr; }
};
Elem* init(string nPass, string fName, string bD, string info);

Elem* AddElem(Elem* lst, string nPass, string fName, string bD, string info);
vector<Elem*> AddElem(vector<Elem*> hT, stringa nPass, string fName, string
bD, string info);
void viewTable(vector<Elem*> hT, bool* f);
void findElemByPass(Elem* lst, string nPass, bool* isFind);
void findElemByPass(vector<Elem*> hT, string nPass);
void findElemByName(vector<Elem*> hT, string fName);
Elem* deleteElem(Elem* lst, string nPass, bool* isDel);
vector<Elem*> deleteElem(vector<Elem*> hT, string nPass);
Elem* clearElem(Elem* pr);
vector<Elem*> clearElems(vector<Elem*> hT);
bool isUniqPassword(vector<Elem*> hT, string nPass);
```

hash-table.cpp

```
#pragma once
#include "hash_table.h"

using namespace std;


Elem* init(string nPass, string fName, string bD, string info)
{
        Elem* lst = new Elem;
        // выделение памяти под корень списка
        lst->numberOfPassword = nPass;
        lst->bDay = bD;
        lst->fullName = fName;
        lst->Info = info;
        lst->next = nullptr; // указатель на следующий узел
        return(lst);
}
Elem* AddElem(Elem* lst, string nPass, string fName, string bD, string info) {
        if (lst == nullptr) {
                lst = init(nPass, fName, bD, info);
                return lst;
```

```
	}
	Elem* p = lst;
	while (p->next != nullptr) {
		p = p->next;
	}
	Elem* temp = new Elem;
	p->next = temp;
	temp->numberOfPassword = nPass;
	temp->bDay = bD;
	temp->fullName = fName;
	temp->Info = info;
	temp->next = nullptr;
	return lst;

}
vector<Elem*> AddElem(vector<Elem*> hT, string nPass, string fName, string bD, string info) {
	unsigned int index;
	index = index = ((nPass[0] * nPass[0] + nPass[1] * nPass[1] + nPass[2] * nPass[2] + nPass[3] * nPass[3] + nPass[5] * nPass[5] + nPass[6] * nPass[6] + nPass[7] * nPass[7] + nPass[8] * nPass[8] + nPass[9] * nPass[9] + nPass[10] * nPass[10]) % n);
	hT[index] = AddElem(hT[index], nPass, fName, bD, info);
	return hT;

}
void viewTable(vector<Elem*> hT, bool* f) {
	Elem* p;
	for (int i = 0; i < n; i++) {
		p = hT[i];
		while (p != nullptr) {
			if (!*f) {
				cout << "Пассажиры" << endl << setw(16) << "Номер паспорта" << setw(23) << "ФИО" << setw(16) << "Дата рождения" << setw(43) << "Дата и место выдачи" << endl;
				*f = true;
			}
			cout << setw(16) << p->numberOfPassword << setw(23) << p->fullName << setw(16) << p->bDay << setw(43) << p->Info << endl;
			p = p->next;
		}
	}
}
void findElemByPass(Elem* lst, string nPass, bool* isFind) {
	if (lst->numberOfPassword == nPass) {
```

```
                *isFind = true;
                cout << setw(28) << "Результаты поиска" << endl;
                cout << setw(7) << "ФИО: "  << lst->fullName << endl;
                cout << setw(18) << "Номер паспорта: "  << lst-
>numberOfPassword << endl;
                cout << setw(17) << "Дата рождения: "  << lst->bDay << endl;
                cout << setw(23) << "Дата и место выдачи: "  << lst->Info << endl;
                return;
        }
        Elem* p = lst;
        while (p->next != nullptr && p->next->numberOfPassword != nPass) p = p-
>next; //под вопросом, может быть исключение
        if (p->next == nullptr) return;
        *isFind = true;
        cout << setw(28) << "Результаты поиска" << endl;
        cout << setw(7) << "ФИО: " << lst->fullName << endl;
        cout << setw(18) << "Номер паспорта: " << lst->numberOfPassword <<
endl;
        cout << setw(17) << "Дата рождения: " << lst->bDay << endl;
        cout << setw(23) << "Дата и место выдачи: " << lst->Info << endl;
        return;
}
void findElemByPass(vector<Elem*> hT, string nPass) {
        unsigned int index;
        bool isFind = false;
        index = index = ((nPass[0] * nPass[0] + nPass[1] * nPass[1] + nPass[2] *
nPass[2] + nPass[3] * nPass[3] + nPass[5] * nPass[5] + nPass[6] * nPass[6] +
nPass[7] * nPass[7] + nPass[8] * nPass[8] + nPass[9] * nPass[9] + nPass[10] *
nPass[10]) % n);
        findElemByPass(hT[index], nPass, &isFind);
}
void findElemByName(vector<Elem*> hT, string fName) {
        bool isFind = false;
        for (auto it = hT.begin(); it != hT.end(); ++it) {
                if (*it == nullptr) continue;
                Elem* p = *it;
                do {
                        if (p->fullName != fName) { p = p->next; continue; }
                        if (!isFind) cout << setw(15) << "ФИО" << setw(25) <<
"Номер паспорта" << endl;
                        isFind = true;
                        cout << setw(20) << p->fullName << setw(20) << p-
>numberOfPassword << endl;
                        p = p->next;
                } while (p != nullptr);
```

```
	}
}
Elem* deleteElem(Elem* lst, string nPass, bool* isDel) {
	if (lst == nullptr) return lst;
	if (lst->numberOfPassword == nPass) {
		Elem* p = lst->next;
		free(lst);
		*isDel = true;
		return(p);
	}
	Elem* p = lst;
	while (p->next != nullptr && p->next->numberOfPassword != nPass) p = p-
>next; //под вопросом, может быть исключение
	if (p->next == nullptr) return(lst);
	Elem* temp = p->next->next;
	free(p->next);
	*isDel = true;
	p->next = temp;
	return(lst);
}
vector<Elem*> deleteElem(vector<Elem*> hT, string nPass) {
	unsigned int index;
	bool isDel = false;
	index = index = ((nPass[0] * nPass[0] + nPass[1] * nPass[1] + nPass[2] *
nPass[2] + nPass[3] * nPass[3] + nPass[5] * nPass[5] + nPass[6] * nPass[6] +
nPass[7] * nPass[7] + nPass[8] * nPass[8] + nPass[9] * nPass[9] + nPass[10] *
nPass[10]) % 100);
	hT[index] = deleteElem(hT[index], nPass, &isDel);
	return(hT);
}
Elem* clearElem(Elem* pr) {
	Elem* p = pr;
	if (p != nullptr) {
		p = p->next;
		clearElem(p);
		//free(pr->next);
		delete pr;
	}
	return nullptr;
}
vector<Elem*> clearElems(vector<Elem*> hT) {
	for (auto it = hT.begin(); it != hT.end(); ++it) {
		if (*it == nullptr) continue;
		*it = clearElem(*it);
	}
```

```
        return hT;
}
bool isUniqPassword(vector<Elem*> hT, string nPass) {
        unsigned int index;
        bool isFind = false;
        index = index = ((nPass[0] * nPass[0] + nPass[1] * nPass[1] + nPass[2] *
nPass[2] + nPass[3] * nPass[3] + nPass[5] * nPass[5] + nPass[6] * nPass[6] +
nPass[7] * nPass[7] + nPass[8] * nPass[8] + nPass[9] * nPass[9] + nPass[10] *
nPass[10]) % n);
        if (hT[index] == nullptr) return true;
        if (hT[index]->numberOfPassword == nPass) {
                return false;
        }
        Elem* p = hT[index];
        while (p->next != nullptr && p->next->numberOfPassword != nPass) p = p-
>next; //под вопросом, может быть исключение
        if (p->next == nullptr) return true;
        return false;
}
```