МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ
И ПРОГРАММНОЙ ИНЖЕНЕРИИ (КАФЕДРА №43)

КУРСОВОЙ ПРОЕКТ
ЗАЩИЩЕН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

доц., канд. техн. наук		В.А. Матьяш
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОМУ ПРОЕКТУ

РАЗРАБОТКА ПРОГРАММЫ «ИСПОЛЬЗОВАНИЕ ЗАДАННЫХ СТРУКТУР ДАННЫХ И АЛГОРИТМОВ ПРИ РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ ПРОДАЖА АВИАБИЛЕТОВ»

по дисциплине: СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТКА ГР.№	4831		К.А. Корнющенков
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2020

Оглавление

1.Задание на курсовой проект

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

1.1 Предметная область - Продажа авиабилетов

Информационная система для предметной области «Продажа авиабилетов» должна осуществлять ввод, хранение, обработку и вывод данных о:

- Пассажирах
- Авиарейсах
- Продаже и возврате авиабилетов.

1.2 Данные о каждом пассажире должны содержать:

- N паспорта – строка формата «NNNN-NNNNNN», где N –цифры;
- Место и дата выдачи паспорта – строка;
- ФИО – строка;
- Дата рождения – строка.

1.3 Данные о пассажирах должны быть организованны в виде хеш-таблицы, первичным ключом которой является «N паспорта».

1.4 Данные о каждом авиарейсе должны содержать:

- No авиарейса – строка формата «AAA-NNN», где AAA – код авиакомпании (буквы латиницы), NNN –порядковый номер авиарейса (цифры);

- Авиакомпания – строка;

- Аэропорт отправления – строка;

- Аэропорт прибытия – строка;

- Дата отправления – строка;

- Время отправления – строка;

- Количество мест всего – целое;

- Количество мест свободных – целое.

1.5 Данные об авиарейсах должны быть организованны в виде АВЛ- дерева поиска, упорядоченного по «No авиарейса».

1.6 Данные о выдаче или возврате авиабилета должны содержать:

- N паспорта – строка, формат которой соответствует аналогичной строке в данных о пассажирах;
- N авиарейса – строка, формат которой соответствует аналогичной строке в данных о авиарейсах;

- N авиабилета – строка из 9 цифр;

Примечания:

Наличие в этих данных записи, содержащей в поле «N паспорта» значения X и в поле «N авиарейса» значения Y соответственно означает продажа авиабилета пассажиру с номером паспорта X на авиарейс с номером Y. Отсутствие такой записи означает, что пассажир с номером паспорта X не покупал билета на авиарейс с номером Y.

На один авиарейс может быть продано несколько билетов. Таким образом, могут быть данные, имеющие повторяющиеся значения в некоторых своих полях.

1.7 Данные о продаже или возврате авиабилетов должны быть организованны в виде списка, который упорядочен по первичному ключу – «N авиабилета». Вид списка и метод сортировки определяются вариантом задания.

1.8 Информационная система «Продажа авиабилетов» должна осуществлять следующие операции:

- Регистрация нового пассажира;

- Удаление данных о пассажире;

- Просмотр всех зарегистрированных пассажиров;

- Очистка данных о пассажирах;

- Поиск пассажира по «N паспорта». Результаты поиска – все сведения о найденном пассажире и номерах авиарейсов, на который он купил билет;

- Поиск пассажира по его ФИО. Результаты поиска – список найденных пассажиров с указанием номера паспорта и ФИО;

- Добавление нового авиарейса;

- Удаление сведений об авиарейсе;

- Просмотр всех авиарейсов;

- Очистка данных об авиарейсах;

- Поиск авиарейса по «No авиарейса». Результаты поиска – все сведения о найденном авиарейсе, а также ФИО и номера паспортов пассажиров, которые купили билет на этот авиарейс;

- Поиск авиарейса по фрагментам названия аэропорта прибытия. Результаты поиска – список найденных авиарейсов с указанием номера авиарейса, аэропорта прибытия, даты отправления, времени
отправления;

- Регистрация продажи пассажиру авиабилета;

- Регистрация возврата пассажиром авиабилета.

1.9 Регистрация продажи авиабилета на определенный авиарейс должна осуществляться только при наличии свободных мест на этот авиарейс.

1.10 При поиске авиарейса по фрагментам «Аэропорт прибытия» могут быть заданы как полное наименование аэропорта, так и его часть.

Варианты методов и алгоритмов:

Метод хеширования – закрытое с квадратичным опробованием

Метод сортировки - извлечением

Вид списка – циклический однонаправленный

Метод обхода дерева - обратный

Алгоритм поиска слова в тексте – БМ

2.Введение

В настоящее время информационные технологии используются повсеместно, в целях автоматизации рабочих мест ведется создание информационных систем, которые позволяют сократить время на обработку, оптимизируют хранение и учет данных. Поэтому необходимы специалисты по разработке и поддержке систем данного вида и актуальны навыки их создания, применения алгоритмов обработки данных.
Задача данной курсовой работы – разработка системы обслуживания пассажиров в авиакомпаниях. Данная задача относится к задачам написания информационных систем, требует разработки структур данных для хранения необходимой информации из предметной области и реализации методов и алгоритмов её обработки.

3.Алгоритмы и структуры данных

3.1. Хеш-Таблица

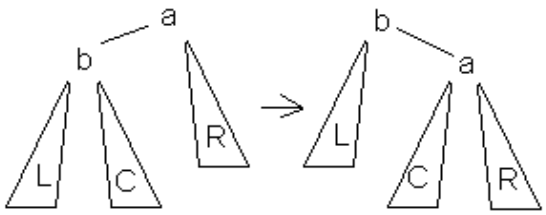
Хеш-таблица - это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары(ключ, значение) и выполнять 3 операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу.
Важное свойство хеш-таблиц состоит в том, что, при некоторых разумных допущениях, все три операции (поиск, вставка, удаление элементов) в среднем выполняются за время O(1). Но при этом не гарантируется, что время выполнения отдельной операции мало. Это связано с тем, что при достижении некоторого значения коэффициента заполнения необходимо осуществлять перестройку индекса хеш-таблицы: увеличить значение размера массива и заново добавить в пустую хеш-таблицу все пары.

Хэш-функции – это функции, предназначенные для «сжатия» произвольного сообщения или набора данных, записанных, как правило, в двоичном алфавите, в некоторую битовую комбинацию фиксированной длины, называемую сверткой. Основным требованием к хэш-функциям является равномерность распределения их значений при случайном выборе значений аргумента.

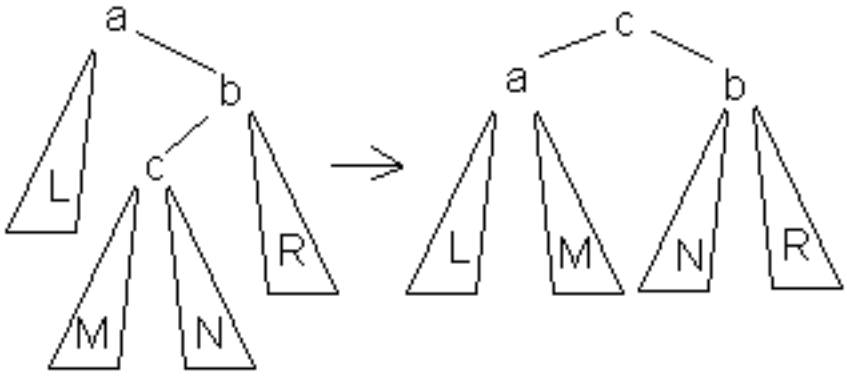
3.2. АВЛ-дерево

АВЛ-дерево — сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на 1
Относительно АВЛ-дерева балансировкой вершины называется операция, которая в случае разницы высот левого и правого поддеревьев = 2, изменяет связи предок-потомок в поддереве данной вершины так, что разница становится <= 1, иначе ничего не меняет. Указанный результат получается вращениями поддерева данной вершины.

Используются 4 типа вращений:
- Большое левое вращение
- Большое правое вращение
- Малое правое вращение
- Малое левое вращение



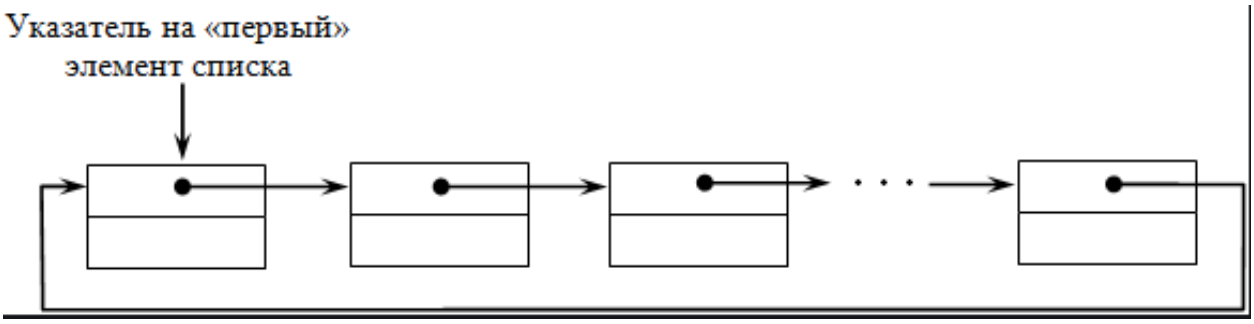
Малое правое вращение



Большое левое вращение

3.3 Циклический двунаправленный список

Циклические однонаправленные списки – это структура данных, представляющая собой последовательность элементов, последний элемент которой содержит указатель на первый элемент списка.



4.Описание программы

4.1 Краткое описание структуры программы

Информационная система представляет собой консольное приложение на языке SWIFT. В приложении используются встроенные в язык контейнеры (UserDefaults) для хранения значений по ключу.

Описание модулей программы:

AVLTree.swift – файл, в котором находится программная реализация структуры данных AVL дерева

HashTable.swift – файл, в котором находится программная реализация структуры данных хеш-таблица

List.swift – файл, в котором находится программная реализация структуры данных линейные циклический список

Boyer-Moore String Search.swift – файл, в котором находится программная реализация БМ поиска в тексте

Passenger.swift – файл, в котором находится реализация структуры данных пассажиры

Flight.swift – файл, в котором находится реализация структуры данных авиарейсты

Tickets.swift – файл, в котором находится реализация структуры данных билеты

data.swift – файл, в котором находится реализация сохранения/загрузки данных из БД

Menu.swift – файл, в котором находится все программная логика и меню программы

main.swift – файл для запуска программы

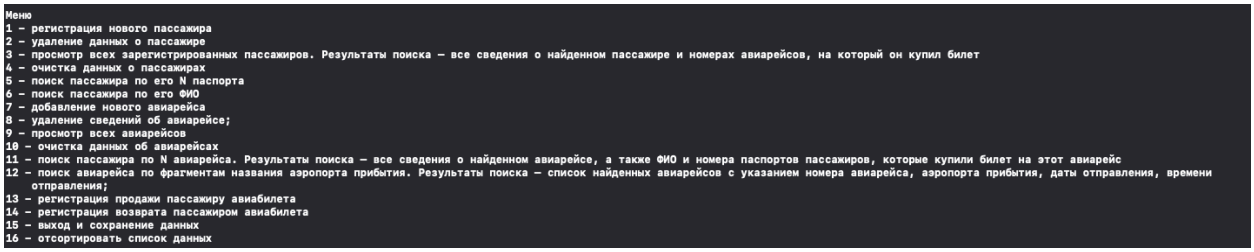
4.2 Руководство по использованию программы

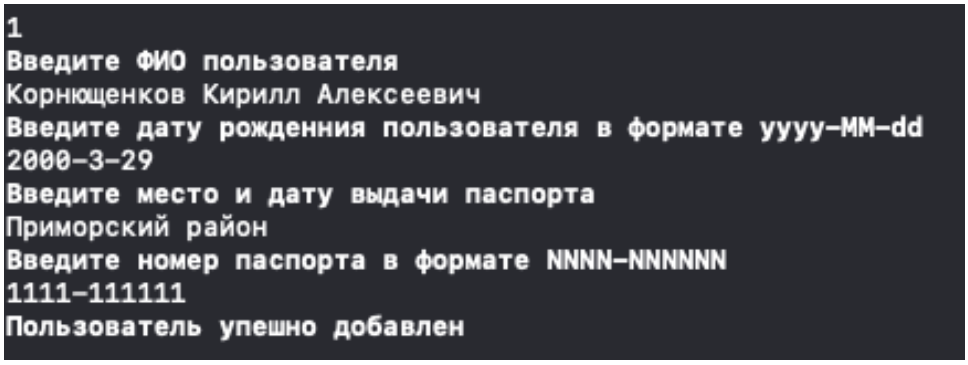
При входе в программу данные, хранящиеся в специальном контейнере, автоматически загружаются, помещаются в соответствующие переменные специально созданных контейнеров. Далее выводится основное меню, после чего программа переходит в режим ожидания какой-либо команды от пользователя. Для выбора действия требуется ввести его номер, так же действуют и другие меню. При выборе какого-либо пункта меню появляются запросы для ввода необходимых данных. Для выполнения нужного действия необходимо ввести запрашиваемые данные, затем программа выведет результат на экран.

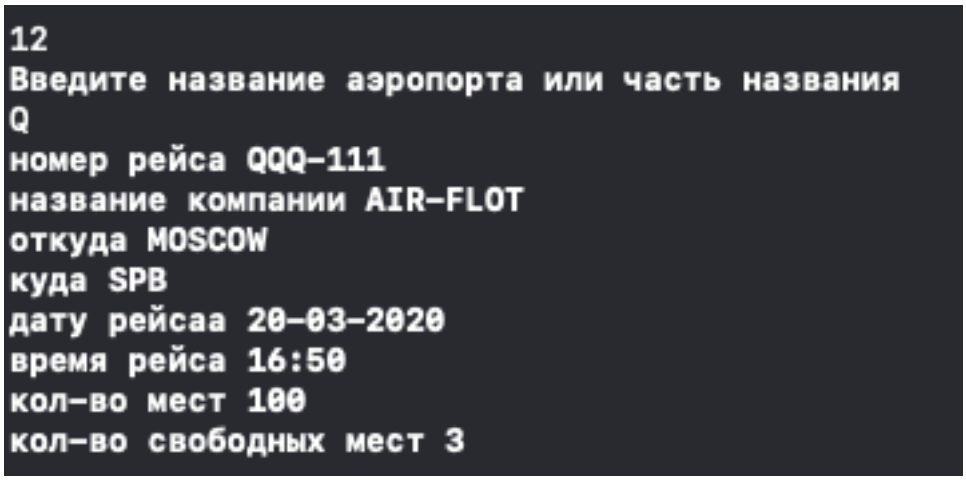
4.3 Листинг программы

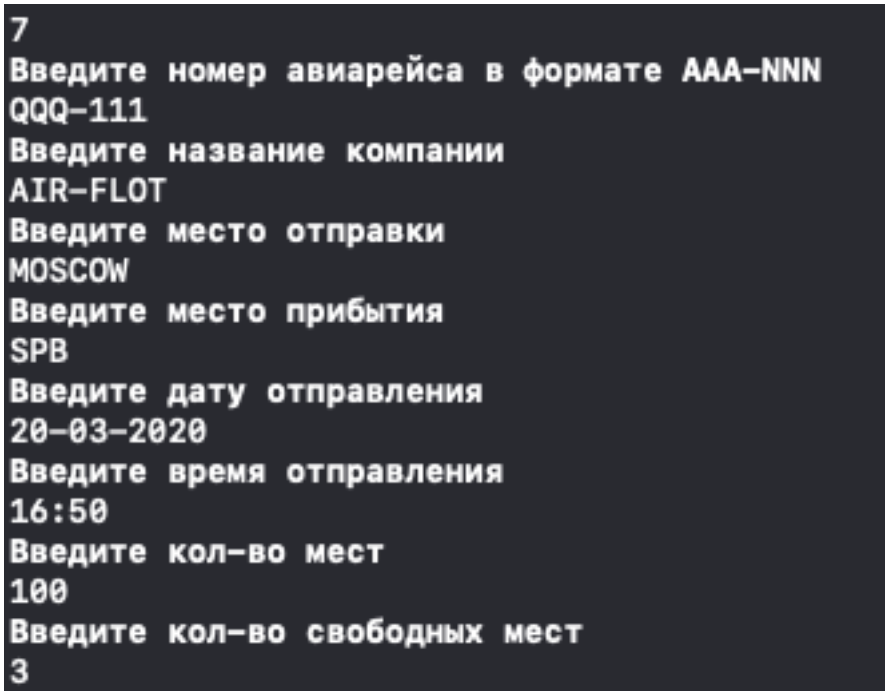
Полный листинг исходных файлов программы показан в приложении А

5.Тестирование программы









6.Заключение

В ходе выполнения проекта была разработана информационная система, выполняющая поставленные задачи, используя при этом структуры данных. Создана система, которая позволяет вводить информацию, осуществлять поиск и удаление данных о пассажирах, авиарейсах и информации о продаже и возврате авиабилетов.
Из достоинств можно выделить достаточно простой и понятный интерфейс.
Из недостатков можно отметить то, что некоторые алгоритмы поиска целесообразно заменить на более эффективные.

7.Список использованной литературы

1. Ключарев А.А., Матьяш В.А., Щекин С.В. Структуры и алгоритмы обработки данных: Учебное пособие / ГУАП. СПб., 2004.
2. Успенский В.А., Семенов А.Л. Теория алгоритмов: основные открытия и приложения. М.: Наука, 1987.
3. Swift. Основы разработки приложений под iOS и macOS | Усов Василий
4. Кнут Д.Е. Искусство программирования. В 3-х томах. М."Вильямс", 2000.

Приложение А

Flight.swift

```
struct Flight {
    var number:String //«AAA-NNN», где AAA – код авиакомпании (буквы латиницы), NNN – порядковый номер авиарейса (цифры);
    var company:String
    var from:String
    var to:String
    var date:String
    var time:String
    var count:Int
    var countFree:Int

    init() {
        self.number = ""
        self.company = ""
        self.from = ""
        self.to = ""
        self.date = ""
        self.time = ""
        self.count = -1
        self.countFree = -1
     }

    static func checkNumber(number:String) -> Bool {
        guard number.count == 7 else { return false}
        var count = 0
        for i in number{
            if count < 3 && i.isSymbol { return false }
            if count == 3 && i != "-" { return false }
            if count > 4 && !i.isNumber { return false }
            count += 1
        }
        return true
    }

    mutating func addNewFlight()->Bool{
        print("Введите номер авиарейса в формате AAA-NNN")
        self.number = readLine()!
        guard self.number != "", Flight.checkNumber(number: self.number) == true else { return false}

        print("Введите название компании")
        self.company = readLine()!
        guard self.company != "" else { return false }

        print("Введите место отправки")
        self.from = readLine()!
        guard self.from != "" else { return false }

        print("Введите место прибытия")
        self.to = readLine()!
```

```
        guard self.to != "" else { return false }

        print("Введите дату отправления")
        self.date = readLine()!
        guard self.date != "" else { return false }

        print("Введите время отправления")
        self.time = readLine()!
        guard self.time != "" else { return false }

        print("Введите кол-во мест")
        let cnt = readLine()!
        count = Int(cnt) ?? -1
        guard count != -1 else { return false }

        print("Введите кол-во свободных мест")
        let cntFree = readLine()!
        countFree = Int(cntFree) ?? -1
        guard self.countFree != -1 else { return false }

        if count < countFree { return false }

        return true
    }

    func printData(){
        print("номер рейса \(number)")
        print("название компании \(company)")
        print("откуда \(from)")
        print("куда \(to)")
        print("дату рейсаа \(date)")
        print("время рейса \(time)")
        print("кол-во мест \(count)")
        print("кол-во свободных мест \(countFree)")
    }
}
```

Passenger.swift

```
struct Passenger {
    var passport:String? //NNNN-NNNNNN проверка формата
    var place:String?
    var fio:String?
    var date:String?

    init() {
        self.passport = nil
        self.place = nil
        self.fio = nil
        self.date = nil
```

```
    }

    init(passport:String,place:String,fio:String,data:String) {
        self.passport = passport
        self.place = place
        self.fio = fio
        self.date = data
    }

    mutating func registerPeople()->Bool{
        print("Введите ФИО пользователя")
        self.fio = readLine()
        guard self.fio != nil, self.fio != "" else { errorReadData(); return false}

        print("Введите дату рожденния пользователя в формате yyyy-MM-dd")
        self.date = readLine()
        guard self.date != nil, checkData(date: self.date), self.date != "" else { errorReadData();
return false}

        print("Введите место и дату выдачи паспорта")
        self.place = readLine()
        guard self.place != nil, self.place != "" else { return false}

        print("Введите номер паспорта в формате NNNN-NNNNNN")
        self.passport = readLine()
        guard self.passport != nil, self.passport != "", Passenger.checkPassport(passport:
self.passport) else { errorReadData(); return false}
        return true
    }

    private func checkData(date:String?) -> Bool{
        let dateFormatter = DateFormatter()
        dateFormatter.dateFormat = "yyyy-MM-dd"
        guard let date = date, let _ = dateFormatter.date(from:date) else { return false}
        return true
    }


    static func checkPassport(passport:String?) -> Bool{
        guard let passport = passport, passport.count == 11 else { return false }
```

```
        var count = 0
        for i in passport{
            if i != "-", count == 4 { return false }
            if !i.isNumber, count != 4 { return false }
            count += 1
        }
        return true
    }

    private func errorReadData(){
        print("Вы ввели неверно данные")
    }
}
```

Tickets.swift

```
struct Tickets {
    var passport: String
    var airFlight: String
    var airTickets: String

    func prindData()  {
        print("Пасспорт пассажира \(passport)")
        print("номер рейса \(airFlight)")
        print("номер билета \(airTickets)")
    }

    static func checkAirTickets(airTickets:String) -> Bool{
        guard airTickets.count == 9 else { return false }
        for i in airTickets{
            if i.isNumber == false{
                return false
            }
        }
        return true
    }
}
```

Menu.swift

```
class Menu {
```

```
var table = UserDefaults.passangerLoad()
var tree = UserDefaults.flightLoad()
var list = UserDefaults.ticketsLoad()

func menu() {
    var check = true
    while check {
        print("")
        print("Меню")
        print("1 - регистрация нового пассажира")
        print("2 - удаление данных о пассажире")
        print("3 - просмотр всех зарегистрированных пассажиров. Результаты поиска – все сведения о найденном пассажире и номерах авиарейсов, на который он купил билет")
        print("4 - очистка данных о пассажирах")
        print("5 - поиск пассажира по его N паспорта")
        print("6 - поиск пассажира по его ФИО")
        print("7 - добавление нового авиарейса")
        print("8 - удаление сведений об авиарейсе;")
        print("9 - просмотр всех авиарейсов")
        print("10 - очистка данных об авиарейсах")
        print("11 - поиск пассажира по N авиарейса. Результаты поиска – все сведения о найденном авиарейсе, а также ФИО и номера паспортов пассажиров, которые купили билет на этот авиарейс")
        print("12 - поиск авиарейса по фрагментам названия аэропорта прибытия. Результаты поиска – список найденных авиарейсов с указанием номера авиарейса, аэропорта прибытия, даты отправления, времени отправления;")
        print("13 - регистрация продажи пассажиру авиабилета")
        print("14 - регистрация возврата пассажиром авиабилета")
        print("15 - выход и сохранение данных")
        print("16 - отсортировать список данных")
        print("")

        let value = readLine()
        switch value {
        case "1":
            registNewPeople()
            break
        case "2":
            deletePeopleData()
            break
```

```
case "3":
    lookAllPeople()
    break
case "4":
    deleteAllPeople()
    break
case "5":
    findForPassport()
    break
case "6":
    findForFio()
    break
case "7":
    addNewAirFlight()
    break
case "8":
    removeAirFligth()
    break
case "9":
    lookAllAirFligth()
    break
case "10":
    removeAllAirFligth()
    break
case "11":
    findForAirFlight()
    break
case "12":
    find12()
    break
case "13":
    saleTickets()
    break
case "14":
    returnTickets()
    break
case "15":
    UserDefaults.flightSave(tree: tree)
    UserDefaults.ticketsSave(list: list)
```

```
            UserDefaults.passangerSave(table: table)
            check = false
            break
        case "16":
            sortList()
            break
        default:
            check = false
        }
    }
}

private func registNewPeople(){
    var people = Passenger()
    if people.registerPeople(){
        table.newElement(passenger: people)
        print("Пользователь упешно добавлен")
    }
}

private func deletePeopleData(){
    print("Введите номер паспорта пассажира")
    let passportNumber = readLine()
    if Passenger.checkPassport(passport: passportNumber){
        table.removeElement(passportNumber: passportNumber!)
    }else{ print("Вы ввели неверно данные") }
}

private func lookAllPeople(){
    table.lookAllPeple()
}

private func deleteAllPeople(){
    table.deleteAllPeople()
}

private func findForPassport(){
    print("Введите номер паспорта пассажира")
    let passportNumber = readLine()
```

```
    guard passportNumber != nil, passportNumber != "" else {
        print("Вы ввели неверно данные");
        return
    }
    guard let people = table.returnPeople(passportNumber: passportNumber!) else {
        print("Пользователя с таким паспортом нету");
        return
    }

    print("ФИО  - \(people.fio!)")
    print("Номер паспорта - \(people.passport!)")
    print("Дата и место получения - \(people.date!)")
    print("Место получения - \(people.place!)")

    list.printAirFlightNumber(passport: people.passport!)
}

private func findForFio(){
    print("Введите номер ФИО пассажира")
    let fio = readLine()
    guard fio != nil, fio != "" else {
        print("Вы ввели неверно данные");
        return
    }
    if !table.findForFIO(fio: fio!){
        print("Таких пользователей нету")
    }
}

private func addNewAirFlight(){
    var flight = Flight()
    if flight.addNewFlight(){
        tree.insert(value: flight)
    }else{ print("Вы ввели неверно данные") }
}

private func removeAirFligth(){
    print("Введите номер авиарейса")
    let number = readLine()
```

```
    guard number != nil, number != "" else { print("Вы ввели неверно данные"); return}
    if Flight.checkNumber(number: number!){
        tree.remove(value: number!)
        print("Пользователь удален ")
    }else{ print("Вы ввели неверно данные") }
}

private func lookAllAirFligth(){
    let data:[Flight] = tree.values
    if data.count == 0 { print("Данных нету") }
    for i in data{
        print(i.number)
        print(i.company)
        print(i.from)
        print(i.to)
        print("")
    }
}

private func removeAllAirFligth(){
    tree.removeAllData()
    print("Все данные об авиарейсах удалены")
}

private func findForAirFlight(){
    print("Введите номер авиарейса")
    let number = readLine()
    guard number != nil, number != "" else { print("Вы ввели неверно данные"); return}

    if Flight.checkNumber(number: number!){
        let data:[Flight] = tree.values
        for i in data{
            if i.number == number!{
                i.printData()
            }
        }
    }else{ print("Вы ввели неверно данные") }
}
```

```
private func find12(){
    print("Введите название аэропорта или часть названия ")
    let name = readLine()
    guard name != "", name != nil else { return }

    let result:[Flight] = tree.searchBM(to: name!)
    if result.count == 0{ print("Данных нету") }
    for i in result{
        i.printData()
        print("")
    }
}

private func saleTickets(){
    print("Введите номер паспорта")
    let passport = readLine()
    guard passport != nil, true == Passenger.checkPassport(passport: passport!) else {
        print("некорректные данные")
        return
    }
    if table.returnPeople(passportNumber: passport!) == nil{
        print("Пользователя с таким паспортом нету")
        return
    }

    print("Введите номер авиарейса")
    let airFlight = readLine()
    guard airFlight != nil, true == Flight.checkNumber(number: airFlight!) else {
        print("некорректные данные")
        return
    }

    if tree.searchBM(to: airFlight!).count == 0{
        print("Такого авиарейса нету")
    }

    print("Введите номер битела")
    let airTickets = readLine()
    guard airTickets != nil, true == Tickets.checkAirTickets(airTickets: airTickets!) else {
```

```
            print("некорректные данные")
            return
        }

        let tickets = Tickets(passport: passport!, airFlight: airFlight!, airTickets: airTickets!)

        if tree.freePlace(company: airFlight!){
            list.addToListEnd(data: tickets)
            tree.minusFreePlace(number: airFlight!)
        }else{
            print("Нету свободных мест")
        }
    }

    private func returnTickets(){
        print("Введите номер билета")
        let airTickets = readLine()
        guard airTickets != nil, true == Tickets.checkAirTickets(airTickets: airTickets!) else {
            print("некорректные данные")
            return
        }

        list.popItem(data: airTickets!)
    }

    private func sortList(){
        list.sortList()
    }
}
```

main.swift

```
let a = Menu()
a.menu()
```

data.swift

```
extension UserDefaults{
    static func passangerSave(table:HashTable){
        var passport:[String] = []
        var place:[String] = []
        var fio:[String] = []
```

```
        var data:[String] = []

        for i in table.data{
            if i.date != nil{
                passport.append(i.passport!)
                place.append(i.place!)
                fio.append(i.fio!)
                data.append(i.date!)
            }
        }
        UserDefaults.standard.set(passport, forKey: "passportPassenger")
        UserDefaults.standard.set(place, forKey: "placePassenger")
        UserDefaults.standard.set(fio, forKey: "fioPassenger")
        UserDefaults.standard.set(data, forKey: "datePassenger")
    }

    static func passangerLoad() -> HashTable{
        let table = HashTable()
        let passport = UserDefaults.standard.array(forKey: "passportPassenger") ?? []
        let place = UserDefaults.standard.array(forKey: "placePassenger") ?? []
        let fio = UserDefaults.standard.array(forKey: "fioPassenger") ?? []
        let data = UserDefaults.standard.array(forKey: "datePassenger") ?? []
        if passport.count != place.count || place.count != fio.count || fio.count != data.count{
            print("Ошибка при загрузке данные в Хэш-таблицу")
            return table
        }

        if passport.count > 0{
            for i in 0..<passport.count{
                let people = Passenger(passport: passport[i] as! String, place: place[i] as! String, fio:
fio[i] as! String, data: data[i] as! String)
                table.newElement(passenger: people)
            }
        }
        return table
    }

    static func ticketsSave(list:CircularLinkedList){
        var passport:[String] = []
        var airFlight:[String] = []
```

```
        var airTickets:[String] = []

        let data = list.returnAllData()
        for i in data{
            passport.append(i.passport)
            airFlight.append(i.airFlight)
            airTickets.append(i.airTickets)
        }

        UserDefaults.standard.set(passport, forKey: "passportTickets")
        UserDefaults.standard.set(airFlight, forKey: "airFlightTickets")
        UserDefaults.standard.set(airTickets, forKey: "airTicketsTickets")
    }

    static func ticketsLoad() -> CircularLinkedList{
        let list = CircularLinkedList()
        let passport = UserDefaults.standard.array(forKey: "passportTickets") ?? []
        let airFlight = UserDefaults.standard.array(forKey: "airFlightTickets") ?? []
        let airTickets = UserDefaults.standard.array(forKey: "airTicketsTickets") ?? []

        if passport.count != airFlight.count || airFlight.count != airTickets.count{
            print("Ошибка при загрузке данных в Список")
            return list
        }

        if passport.count > 0{
            for i in 0..<passport.count{
                let data = Tickets(passport: passport[i] as! String, airFlight: airFlight[i] as! String,
airTickets: airTickets[i] as! String)
                list.addToListEnd(data: data)
            }
        }
        return list
    }

    static func flightSave(tree:AVLTree){
        var number:[String] = []
        var company:[String] = []
        var from:[String] = []
        var to:[String] = []
```

```
        var date:[String] = []
        var time:[String] = []
        var count:[Int] = []
        var countFree:[Int] = []

        for i in tree.values{
            number.append(i.number)
            company.append(i.company)
            from.append(i.from)
            to.append(i.to)
            date.append(i.date)
            time.append(i.time)
            count.append(i.count)
            countFree.append(i.countFree)
        }

        UserDefaults.standard.set(number, forKey: "numberAVLTree")
        UserDefaults.standard.set(company, forKey: "companyAVLTree")
        UserDefaults.standard.set(from, forKey: "fromAVLTree")
        UserDefaults.standard.set(to, forKey: "toAVLTree")
        UserDefaults.standard.set(date, forKey: "dateAVLTree")
        UserDefaults.standard.set(time, forKey: "timeAVLTree")
        UserDefaults.standard.set(count, forKey: "countAVLTree")
        UserDefaults.standard.set(countFree, forKey: "countFreeAVLTree")
    }

    static func flightLoad() -> AVLTree{
        let tree = AVLTree()
        let number = UserDefaults.standard.array(forKey: "numberAVLTree") ?? []
        let company = UserDefaults.standard.array(forKey: "companyAVLTree") ?? []
        let from = UserDefaults.standard.array(forKey: "fromAVLTree") ?? []
        let to = UserDefaults.standard.array(forKey: "toAVLTree") ?? []
        let date = UserDefaults.standard.array(forKey: "dateAVLTree") ?? []
        let time = UserDefaults.standard.array(forKey: "timeAVLTree") ?? []
        let count = UserDefaults.standard.array(forKey: "countAVLTree") ?? []
        let countFree = UserDefaults.standard.array(forKey: "countFreeAVLTree") ?? []

        if number.count != company.count || from.count != to.count || date.count != time.count ||
count.count != countFree.count {
            print("Ошибка при загрузке данных в Дерево")
```

```
            return tree
        }

        if number.count > 0 {
            for i in 0..<number.count{
                var data = Flight()
                data.number = number[i] as! String
                data.company = company[i] as! String
                data.from = from[i] as! String
                data.to = to[i] as! String
                data.date = date[i] as! String
                data.time = time[i] as! String
                data.count = count[i] as! Int
                data.countFree = countFree[i] as! Int
                tree.insert(value: data)
            }
        }

        return tree
    }
}
```

AVLTree.swift

```
public class AVLTree {
    internal var root: AVLNode?
    init() { }

    func insert(value: Flight) {
        if let root = self.root {
            self.root = root.insert(value)
        } else {
            self.root = AVLNode(value)
        }
    }

    func remove(value: String) {
        guard let root = self.root else { return }
        self.root = root.remove(value)
    }
```

```
    func removeAllData(){
        for i in values{
            remove(value: i.number)
        }
    }

    var isEmpty: Bool { return self.root == nil }

    var values: [Flight] {
        guard let root = self.root else { return [] }
        var result: [Flight] = []
        root.preorderTraversal { result.append($0) }
        return result
    }

    func searchBM(to:String) -> [Flight]{
        guard let root = self.root else { return [] }
        var result: [Flight] = []
        root.preorderTraversalAndSearchBM(to: to, { result.append($0) })
        return result
    }

    func freePlace(company:String) -> Bool {
        for i in values{
            if i.company == company{
                if i.countFree > 0 { return true }
                else { return false }
            }
        }
        return false
    }

    func minusFreePlace(number:String){
        self.root?.preorderTraversalFree(number: number)
    }
}

internal class AVLNode{
    var height: Int = 1
```

```
var value: Flight
var left, right: AVLNode?

var balance: Int {
    return self.leftHeight - self.rightHeight
}

var leftHeight: Int {
    return self.left?.height ?? 0
}

var rightHeight: Int {
    return self.right?.height ?? 0
}

init(_ value: Flight) {
    self.value = value
}

private func rotateRight() -> AVLNode {
    let b = self.left!
    self.left = b.right
    b.right = self
    self.recalculateHeight()
    b.recalculateHeight()
    return b
}

private func rotateLeft() -> AVLNode {
    let b = self.right!
    self.right = b.left
    b.left = self
    self.recalculateHeight()
    b.recalculateHeight()
    return b
}

private func rotateLeftRight() -> AVLNode {
    self.left = self.left!.rotateLeft()
```

```
    self.recalculateHeight()
    return self.rotateRight()
  }

  private func rotateRightLeft() -> AVLNode {
    self.right = self.right!.rotateRight()
    self.recalculateHeight()
    return self.rotateLeft()
  }

  private func recalculateHeight() {
    self.height = 1 + max(self.leftHeight, self.rightHeight)
  }

  internal func insert(_ value: Flight) -> AVLNode {
    defer { self.recalculateHeight() }
    if value.number < self.value.number {
      if let left = self.left {
        self.left = left.insert(value)
        return self.rebalanceIfNeeded()
      } else {
        self.left = AVLNode(value)
        return self
      }
    } else {
      if let right = self.right {
        self.right = right.insert(value)
        return self.rebalanceIfNeeded()
      } else {
        self.right = AVLNode(value)
        return self
      }
    }
  }

  private func removeLeftmostNode() -> AVLNode {
    guard let left = self.left else {
      assertionFailure("this only works on nodes with a left child")
      return self
```

```
    }
    defer { self.recalculateHeight() }
    if let _ = left.left {
        let result = left.removeLeftmostNode()
        self.left = left.rebalanceIfNeeded()
        return result
    }
    self.left = left.right
    return left
}

private func rebalanceIfNeeded() -> AVLNode {
    let balance = self.balance
    if balance > 1 {
        if let leftBalance = self.left?.balance, leftBalance > 0 {
            return self.rotateRight()
        } else {
            return self.rotateLeftRight()
        }
    } else if balance < -1 {
        if let rightBalance = self.right?.balance, rightBalance < 0 {
            return self.rotateLeft()
        } else {
            return self.rotateRightLeft()
        }
    }
    return self
}

internal func remove(_ value: String) -> AVLNode? {
    if value < self.value.number, let left = self.left {
        self.left = left.remove(value)
        self.recalculateHeight()
        return self.rebalanceIfNeeded()
    } else if value > self.value.number, let right = self.right {
        self.right = right.remove(value)
        self.recalculateHeight()
        return self.rebalanceIfNeeded()
    }
```

```
    if let right = self.right {
        let sucessor: AVLNode
        if let _ = right.left {
            sucessor = right.removeLeftmostNode()
            self.right = right.rebalanceIfNeeded()
            sucessor.left = self.left
            sucessor.right = self.right
        } else {
            sucessor = right
            sucessor.left = self.left
        }
        sucessor.recalculateHeight()
        return sucessor.rebalanceIfNeeded()
    } else if let left = self.left {
        return left
    }
    return nil
}

//обратный проход дерева
internal func preorderTraversal(_ callback: (Flight) -> Void) {
    self.left?.preorderTraversal(callback)
    self.right?.preorderTraversal(callback)
    callback(self.value)
}

internal func preorderTraversalFree(number:String) {
    self.left?.preorderTraversalFree(number: number)
    self.right?.preorderTraversalFree(number: number)
    if self.value.number == number{
        self.value.countFree -= 1
        return
    }
  }

//обратный проход дерева
internal func preorderTraversalAndSearchBM(to:String,_ callback: (Flight) -> Void) {
    self.left?.preorderTraversal(callback)
    self.right?.preorderTraversal(callback)
```

```
        if self.value.to.searchBM(pattern: to) != nil{
            callback(self.value)
        }
    }
}
```

Boyer-Moore String Search.swift

```
extension String {
    func searchBM(pattern: String) -> Index? {
        let patternLength = pattern.count
        guard patternLength > 0, patternLength <= count else { return nil }

        var skipTable = [Character: Int]()
        for (i, c) in pattern.enumerated() {
            skipTable[c] = patternLength - i - 1
        }

        let p = pattern.index(before: pattern.endIndex)
        let lastChar = pattern[p]

        var i = index(startIndex, offsetBy: patternLength - 1)

        func backwards() -> Index? {
            var q = p
            var j = i
            while q > pattern.startIndex {
                j = index(before: j)
                q = index(before: q)
                if self[j] != pattern[q] { return nil }
            }
            return j
        }

        while i < endIndex {
            let c = self[i]
            if c == lastChar {
                if let k = backwards() { return k }
                let jumpOffset = max(skipTable[c] ?? patternLength, 1)
                i = index(i, offsetBy: jumpOffset, limitedBy: endIndex) ?? endIndex
            } else {
```

```
            i = index(i, offsetBy: skipTable[c] ?? patternLength, limitedBy: endIndex) ?? endIndex
        }
    }
    return nil
  }
}
```

HashTable.swift

```
class HashTable{
  var data:[Passenger] = []
  private let c = 2, d = 5
  private var sizeData:Int{
    get{ return data.count }
  }

  init() { uppendNilData() }

  private func uppendNilData(){
    for _ in 0..<1500{ data.append(Passenger()) }
  }

  private func hash(newValue:String) -> Int {
    return abs(newValue.hashValue % sizeData)
  }

  func newElement(passenger:Passenger){
    var index = hash(newValue: passenger.passport!)
    var tryCount = 0,check = true
    while check {
      index += c*tryCount + d*(tryCount^2)
      while index > sizeData { uppendNilData() }
      if data[index].passport == passenger.passport {
        print("Пользователь с таким паспортом уже есть")
        check = false
        return
      }
      if data[index].passport == nil{
        data[index] = passenger
        check = false
      }
```

```
        tryCount += 1
    }
}

func removeElement(passportNumber:String){
    var index = hash(newValue: passportNumber)
    var tryCount = 0,check = true
    while check {
        index += c*tryCount + d*(tryCount^2)
        if index > sizeData{
            print("Пользователя с таким паспортом нету")
            check = false
            return
        }
        if data[index].passport == passportNumber{
            data[index] = Passenger()
            check = false
        }
        tryCount += 1
    }
}

func lookAllPeple() {
    var count = 0
    for i in data{
        if i.passport != nil{
            count += 1
            print(i.date!," ",i.fio!," ",i.passport!," ",i.place!)
        }
    }
    if count == 0{
        print("Пользователей ещё нету")
    }
}

func deleteAllPeople() {
    for i in 0..<sizeData {
        if data[i].passport != nil{ data[i] = Passenger() }
    }
```

```
        print("Все данные о пассажирах удалены")
    }

    func returnPeople (passportNumber:String) -> Passenger?{
        var index = hash(newValue: passportNumber)
        var tryCount = 0,check = true
        while check {
            index += c*tryCount + d*(tryCount^2)
            if index > sizeData { return nil }
            if data[index].passport == passportNumber{
                check = false
                return data[index]
            }
            tryCount += 1
        }
    }

    func findForFIO(fio:String) -> Bool{
        var check = false
        for i in data{
            if i.fio == fio{
                print("ФИО - \(fio)   номер паспорта - \(i.passport!)")
                check = true
            }
        }
        return check
    }
}
```

list.swift

```
class Node:Equatable{
    static func == (lhs: Node, rhs: Node) -> Bool {
        return lhs.data.airTickets == rhs.data.airTickets
    }

    var next:Node?
    var data:Tickets

    init(_ data:Tickets) {
        self.next = nil
```

```
        self.data = data
    }
}

class CircularLinkedList{
    var last:Node?
    var currentTurn:Node?

    init() {
        self.last = nil
        self.currentTurn = nil
    }

    func addToEmptyList(data:Tickets){
        let temp = Node(data)
        self.last = temp
        self.last?.next = self.last
        self.currentTurn = self.last?.next!
    }

    func addToListEnd(data:Tickets){
        if self.last == nil{
            self.addToEmptyList(data: data)
            return
        }
        let temp = Node(data)
        temp.next = self.last?.next
        self.last!.next! = temp
        self.last = temp
    }

    func popItem(data:String){
        var currentNode = self.last
        if currentNode == nil{ print("Данных нету") }
        while (currentNode != nil){
            if currentNode?.next?.data.airTickets == data{
                currentNode?.next = currentNode?.next?.next
                break
            }
```

```
            currentNode = currentNode?.next

            if currentNode == self.last{
                print("Данных таких нету")
                break
            }
        }
    }

    func traverse(){
        var currentNode = self.last?.next
        while (currentNode != nil){
            print(currentNode!.data)
            currentNode = currentNode?.next
            if currentNode == self.last?.next{
                break
            }
        }
    }

    func returnAllData() -> [Tickets] {
        var data:[Tickets] = []
        var currentNode = self.last?.next
        while (currentNode != nil){
            data.append(currentNode!.data)
            currentNode = currentNode?.next
            if currentNode == self.last?.next{
                return data
            }
        }
        return data
    }

    func retrieveTurn() -> Tickets{
        return self.currentTurn!.data
    }

    func updateTurn(){
        self.currentTurn = self.currentTurn?.next
```

```
    }

    //вывод номеров авиарейсов на которые куплены билеты на паспорту
    func printAirFlightNumber(passport:String) {
        var currentNode = self.last?.next
        while (currentNode != nil){
            print(currentNode!.data.airFlight)
            currentNode = currentNode?.next
            if currentNode == self.last?.next{
                break
            }
        }
    }
}
```