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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Старший преподаватель |  |  |  | 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»**  по дисциплине: СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4134 |  |  |  | Н.А. Костяков |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2023

**СОДЕРЖАНИЕ**

[**1.** **Задание на курсовой проект** 3](#_Toc135101280)

[**2.** **Описание структур данных** 4](#_Toc135101281)

[**3.** **Описание программы и функций** 5](#_Toc135101282)

[**4.** **Результаты тестирования программы** 10](#_Toc135101330)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 17](#_Toc135101333)

[**Приложение А. Текст программы** 18](#_Toc135101334)

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 на тему регистрации постояльцев в гостинице».

**Вариант задания**

1. Предметная область: Регистрация постояльцев в гостинице

2. Метод хеширования: открытое хеширование

3. Метод сортировки: Извлечением

4. Вид списка: Линейный двунаправленный

5. Метод обхода дерева: Прямой

6. Алгоритм поиска слова в тексте: Боуера и Мура (БМ)

1. **Описание структур данных**

Данные о Постояльцах хранятся в структуре Guest, которая представляет собой хеш-таблицу и хранит в себе данные о паспорте, ФИО, годе рождения, адресе и цели визита.

**Данные о номерах** хранятся в структуре node, которая представляет собой АВЛ-дерево и содержит имя номера, вместительность, технику, информацию о санузле.

**Данные о регистрации** хранятся в структуре LinkedList, которая представляет собой линейный однонаправленный список и содержит паспорт гостя, номер комнаты и даты вселения и выселения.

**База данных** (список ранее зарегистрированных пассажиров и авиарейсов) хранится в файлах формата .txt.

Формат хранения данных в файле без пробелов:

– Номер паспорта – строка формата «NNNN-NNNNNN», где N – цифры; – ФИО – строка; – Год рождения – целое; – Адрес – строка; – Цель прибытия – строка.

. Данные о каждом гостиничном номере должны содержать: – Номер гостиничного номера – строка формата «ANNN», где A – буква, обозначающая тип номера (Л – люкс, П – полулюкс, О – одноместный, М – многоместный); NNN – порядковый номер (цифры); – Количество мест – целое; – Количество комнат – целое; – Наличие санузла – логическое; – Оборудование – строка.

**2.1 Алгоритмы и структуры данных**

**1)AVL-дерево**

-Дерево является сбалансированным двоичным деревом поиска, которое обеспечивает эффективность операций поиска, вставки и удаления элементов. В данном проекте AVL-дерево было выбрано для хранения информации о гостиничных номерах, где ключом является номер комнаты. Благодаря своей сбалансированной структуре, AVL-дерево гарантирует логарифмическое время выполнения операций, что обеспечивает высокую производительность и эффективность программы.

**2)Хеш-таблица**   
 В данном проекте для эффективного хранения и доступа к данным о снятых номерах. Эта структура данных основана на хэш-функции и обеспечивает быстрый доступ к данным. Благодаря хэш-таблице, мы можем быстро определить какие комнаты заселены и кем .Такой подход значительно ускоряет процесс заселения клиента и обеспечивает более эффективное управление гостиницей

**3) Линейный двунаправленный**

В данном проекте для эффективного хранения и обработки данных о постояльцах был использован линейный двунаправленный список. Эта структура данных позволяет последовательно хранить информацию и осуществлять операции добавления и удаления элементов в любой точке списка. Благодаря использованию линейного двунаправленного списка, мы можем эффективно управлять данными о гостях. Каждый элемент списка представлен узлом, содержащим данные и ссылки на предыдущий и следующий элементы списка, что обеспечивает гибкость и удобство работы с данными.  
**4) Сортировка извлечением**  
Сортировка извлечением - это алгоритм сортировки, основанный на принципе поиска минимального (или максимального) элемента в неотсортированной части массива и перемещении его в начало (или конец) уже отсортированной части. Данный алгоритм сортировки является не самым эффективным, но простым и легко реализуемым. Процесс сортировки извлечением следующий: 1. Установим указатель на начало массива. 2. Найдем минимальный элемент в неотсортированной части массива. 3. Поменяем местами минимальный элемент с элементом, на который указывает текущий указатель (то есть переместим минимальный элемент в начало отсортированной части). 4. Увеличим указатель на единицу и повторим шаги 2-4 до тех пор, пока указатель не достигнет конца массива. Таким образом, на каждой итерации алгоритма мы находим минимальный элемент в неотсортированной части массива и перемещаем его в начало отсортированной части. По мере выполнения алгоритма, количество отсортированных элементов увеличивается, а количество неотсортированных элементов уменьшается, пока не будут все элементы упорядочены. Преимущества сортировки извлечением: простота реализации и легкость понимания, не требует больших объемов памяти.

**5) Метод обхода AVL-дерева прямой**

Прямой обход AVL-дерева является одним из методов обхода дерева. Этот метод использует следующий алгоритм: 1. Расположить текущую вершину в начале цикла. 2. Проверить текущую вершину на пустоту. 3. Если текущая вершина не пуста, то выполнить действия с вершиной (например, вывести ее значение на экран). 4. Перейти к левому поддереву текущей вершины: рекурсивно повторить шаги 1-3 для вершины, лежащей в левом поддереве. 5. Перейти к правому поддереву текущей вершины: рекурсивно повторить шаги 1-3 для вершины, лежащей в правом поддереве. Таким образом, в процессе прямого обхода AVL-дерева мы сначала обрабатываем текущую вершину, затем обрабатываем все вершины левого поддерева, а затем все вершины правого поддерева. Прямой обход является одним из методов обхода дерева в глубину, которые позволяют обрабатывать все вершины дерева. Этот метод является часто используемым при работе с бинарными деревьями поиска. Преимущества прямого обхода AVL-дерева: - простота реализации; - расход памяти в процессе выполнения минимальный; - возможность использования при поиске элементов.

**6) Открытое хеширование.**

Открытая хеш-таблица (открытое хеширование) - это структура данных, используемая для хранения пар "ключ-значение". В отличие от закрытой хеш-таблицы, в которой конфликтные элементы помещаются в отдельные списки, в открытой хеш-таблице конфликты разрешаются путем поиска свободной ячейки, в которой можно сохранить элемент.

Процесс добавления элемента в открытую хеш-таблицу: 1. Вычисляем хеш-код для ключа элемента. 2. Вычисляем индекс таблицы, используя хеш-код и размер таблицы. 3. Если ячейка с полученным индексом свободна, помещаем элемент в эту ячейку. 4. Если ячейка с полученным индексом занята, то мы продолжаем поиск следующей свободной ячейки. Этот процесс называется «опробованием» (probing). Например, можно последовательно просматривать соседние ячейки до тех пор, пока не будет найдена свободная ячейка. 5. После того, как мы найдём свободную ячейку, помещаем элемент в неё.

При поиске элемента в открытой хеш-таблице мы также: 1. Вычисляем хеш-код для ключа элемента. 2. Вычисляем индекс таблицы, используя хеш-код и размер таблицы. 3. Если ячейка соответствующая полученному индексу содержит нужный элемент, возвращаем его. 4. Иначе начинаем проверку ячеек, следующих за текущей, до тех пор, пока не найдём нужный элемент, либо не окажется, что такой элемент отсутствует.

Преимущества открытой хеш-таблицы: - Она имеет меньший расход памяти, чем закрытая хеш-таблица, так как не требует дополнительных списков для хранения элементов, имеющих одинаковый хеш-код. - В случае правильной конфигурации размера таблицы и функции хеширования (если хорошо сочетаются с набором входных данных), время выполнения операции поиска, добавления и удаления элемента может быть лучше, чем у закрытой хеш-таблицы.

**7) Алгоритм Боуера и Мура:**

Бойер-Мур - это алгоритм для поиска слова, который позволяет находить все вхождения данного слова в заданный текст за линейное время (O(n)) в среднем, где n - длина текста. Алгоритм Бойера-Мура является одним из самых быстрых алгоритмов поиска подстроки в больших текстах. Принцип работы алгоритма Бойера-Мура основан на использовании следующих двух эвристических правил: 1. Метод «неудачного сравнения», который основан на том, что производится поиск справа налево и сравниваются последние символы шаблона и текста. Если символы не совпадают, то можно сдвинуть шаблон на количество позиций, равное разности между индексом символа в шаблоне и числом совпавших символов в начале шаблона. 2. Метод "хорошего суффикса", который позволяет определить длину перекрытия хороших суффиксов с префиксами шаблона и осуществляет сдвиг шаблона на это значение. Сочетание этих эвристических правил, позволяет избежать сравнения подстроки со всеми возможными вхождениями в тексте и ускорить процесс поиска. Преимущества алгоритма Бойера-Мура: - Обеспечивает линейную сложность по времени и может быть лучше, чем другие алгоритмы, например, Кнута-Морриса-Пратта (KMP). - Не требует использования дополнительной памяти (за исключением временных переменных). - Может быть использован для поиска одной подстроки во многих текстах.

1. **Описание программы и функций**

Данная программа реализована на языке С++ и представляет собой консольное приложение. С помощью отдельной функции реализовано меню пользователя, которому необходимо выбрать какое-либо действие, указав определенную цифру.

1) регистрация нового постояльца – добавление в хеш-таблицу информации о госте

2) удаление данных о постояльце – удаление гостя из хеш-таблицы

3) просмотр всех зарегистрированных постояльцев – вывод списком информации о гостях из хеш-таблицы

4) очистка данных о постояльцах – удаление и создание новой хеш-таблицы

5) поиск постояльца по номеру паспорта – поиск по хеш-таблице по ключу паспорта, который хешируется

6) поиск постояльца по ФИО – поиск по вторичному ключу в хеш-таблице

7) добавление нового гостиничного номера – добавление в авл-дерево нового узла

8) удаление сведений о гостиничном номере – удаление узла и ребалансировка авл-дерева

9) просмотр всех имеющихся гостиничных номеров – обход авл-дерева и сопутствующий вывод информации об узле

10) очистку данных о гостиничных номерах – создание нового авл-дерева

11) поиск гостиничного номера по «номеру гостиничного номера» - обход дерева и поиск по ключу

12) поиск гостиничного номера по фрагментам «Оборудования» - при помощи алгоритма поиска обходится все дерево, пока не будет найдено совпадение

13) регистрацию вселения постояльца – добавление в список информации о вселении гостя в номер

14) регистрацию выселения постояльца – удаление заселения из списка о заселении гостей

**Более подробное описание функций изложено ниже**

Реализация всех функций находится в [приложении A](#_Приложение_1)

**Добавление данных в хеш-таблицу**

void put(string passNum, string fullName, int birthYear, string address, string purpose)

Пользователь вводит данные нового гостя, при этом номер паспорта должен быть уникальным

Полученные данные добавляются в хеш-таблицу, которая затем выводится на экран.

**Удаление данных из хеш-таблицы**

void remove(Passenger\*\* head, string passport, bool &flag)

Пользователь вводит номер паспорта гостя. Если указанный номер не найден – выводится соответствующее сообщение. Если найден – данные пассажира удаляются из хеш-таблицы и выводится соответствующее сообщение на экран.

**Вывод хеш-таблицы на экран**

void show\_all()

На экран выводится хеш-таблица.

**Поиск в хеш-таблице по ключу**

void show( string passport)

Пользователь вводит номер паспорта Гостя, который преобразуется хеш-функцией в ключ. По полученному ключу осуществляется поиск в хеш-таблице. Если в хеш-таблице по данному ключу лежат несколько полей, то производится проверка на совпадение самого паспорта.

Если поле данных найдено, оно выводится на экран. В противном случае на экран выводится сообщение об отсутствии искомого пассажира.

**Поиск в хеш-таблице по значению**

void show\_by\_name(string FIO)

Пользователь вводит ФИО гостя. По полученным данным производится поиск в хеш-таблице.

Если поле данных найдено, оно выводится на экран. В противном случае на экран выводится сообщение об отсутствии искомого пассажира.

void remove\_all() void Free\_hash(Passenger\*\* head, Flight\* root, Ticket\*\* Thead)

Происходит очистка всей хеш-таблицы с выводом сообщения об этом на экран.

**Добавление данных в АВЛ-дерево**

node\* insert(node\* p, int k, unsigned char t = 'A', int cap = 2, string te = "-", bool b = 1);

Пользователь вводит данные о номере, при этом номер комнаты должен быть уникальным.

Полученные данные добавляются в АВЛ-дерево

**Удаление данных из АВЛ-дерева**

node\* remove(node\* p, int k);

Пользователь вводит номер комнаты. Если указанный номер не найден – выводится соответствующее сообщение. Если найден – данные о номере удаляются из АВЛ-дерева

**Вывод АВЛ-дерева на экран**

void show(node\* root, int space = 0);На экран выводится АВЛ-дерево

**Поиск в АВЛ-дереве**

void serch(node\* tree, int key);

Пользователь вводит номер. По полученным данным осуществляется поиск.

Если поле обнаружено – оно выводится на экран. В противном случае – Ничего

**Поиск в АВЛ-дереве по фрагменту**

void find\_by\_technic(node\* tree, string technic);

Пользователь вводит фрагмент техники из номера. По полученным данным производится поиск.

Если поле данных найдено, оно выводится на экран.

**Добавление данных в список**

void add(string p, string r, string in, string out)

Пользователь вводит номер паспорта и номер

пассажир будет зарегистрирован на указанный рейс, и выведется соответствующее сообщение на экран. В противном случае – выведется соответствующее сообщение.

**Удаление данных из списка**

Node\* remove\_by\_room(string room)

Пользователь вводит номер.

При наличии всех введенных данных в соответствующих структурах введенные данные будут удалены. В противном случае удаление данных не произойдет, и выведется соответствующее сообщение на экран.

**Выход из программы**

Программа завершает свою работу с выводом соответствующего сообщения на экран.

1. **Результаты тестирования программы**

**4.1 Словесное описание**

**Основное меню**

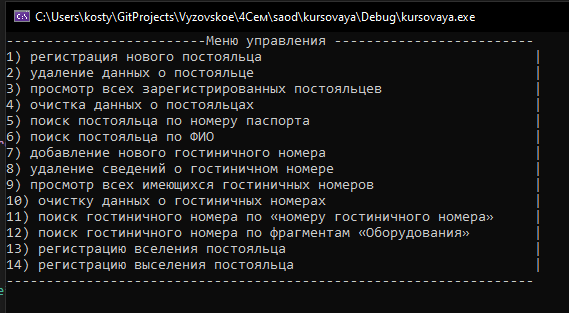
| № | Действие | Входные данные | Результат |
| --- | --- | --- | --- |
| 1 | 1) регистрация нового постояльца | Паспорт:  ФИО:  год рождения:  Адрес:  цель визита: | регистрация нового постояльца |
| 2 | 2) удаление данных о постояльце | Паспорт | удаление данных о постояльце |
| 3 | 3) просмотр всех зарегистрированных постояльцев | Номер операции | Информация о постояльцах |
| 4 | 4) очистка данных о постояльцах | Номер операции | очистка данных о постояльцах |
| 5 | 5) поиск постояльца по номеру паспорта | Паспорт | Информация о постояльцах |
| 6 | 6) поиск постояльца по ФИО | ФИО | Информация о постояльцах |
| 7 | 7) добавление нового гостиничного номера | Номер  Буква  Вместительность  Техника  1 или 0 за наличие ванной | добавление нового гостиничного номера |
| 8 | 8) удаление сведений о гостиничном номере | Номер | удаление сведений о гостиничном номере |
| 9 | 9) просмотр всех имеющихся гостиничных номеров | Номер операции | просмотр всех имеющихся гостиничных номеров |
| 10 | 10) очистку данных о гостиничных номерах | Номер операции | очистка данных о гостиничных номерах |
| 11 | 11) поиск гостиничного номера по «номеру гостиничного номера» | Номер | Информация о номере |
| 12 | 12) поиск гостиничного номера по фрагментам «Оборудования» | Оборудование | Информация о номере |
| 13 | 13) регистрацию вселения постояльца | Номер  паспорт постояльца  дату заселения  дату выселения | регистрация вселения постояльца |
| 14 | 14) регистрацию выселения постояльца | Номер | регистрация выселения постояльца |

**Тестовые данные:**

| № | Действие | Входные данные |
| --- | --- | --- |
| 1 | 1) регистрация нового постояльца | 3017-123245  Костяков\_Никита  2003  Спб  отдых |
| 2 | 2) удаление данных о постояльце | 3017-123245 |
| 3 | 3) просмотр всех зарегистрированных постояльцев | - |
| 4 | 4) очистка данных о постояльцах | - |
| 5 | 5) поиск постояльца по номеру паспорта | 3017-123245 |
| 6 | 6) поиск постояльца по ФИО | Сидоров\_Сидор\_Сидорович |
| 7 | 7) добавление нового гостиничного номера | 1333  A  4  None  1 |
| 8 | 8) удаление сведений о гостиничном номере | 1333 |
| 9 | 9) просмотр всех имеющихся гостиничных номеров | - |
| 10 | 10) очистку данных о гостиничных номерах | - |
| 11 | 11) поиск гостиничного номера по «номеру гостиничного номера» | 1233 |
| 12 | 12) поиск гостиничного номера по фрагментам «Оборудования» | tv |
| 13 | 13) регистрацию вселения постояльца | 1233E  3017-551122  12.10.23  12.11.2023 |
| 14 | 14) регистрацию выселения постояльца | 123G |

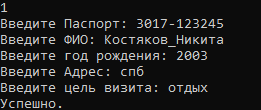
**Скриншоты выполнения тестовых данных**

**Основное меню**

****

**Пример тестовых данных**

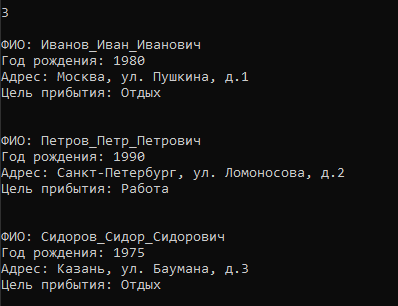
1. регистрация нового постояльца



1. удаление данных о постояльце



1. просмотр всех зарегистрированных постояльцев



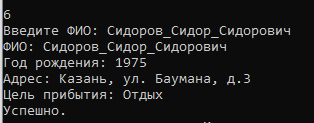
1. очистка данных о постояльцах



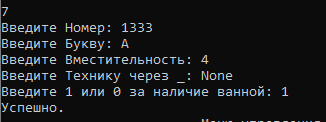
1. поиск постояльца по номеру паспорта



1. поиск постояльца по ФИО



1. добавление нового гостиничного номера



1. удаление сведений о гостиничном номере



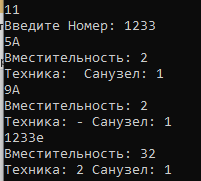
1. просмотр всех имеющихся гостиничных номеров



1. очистку данных о гостиничных номерах



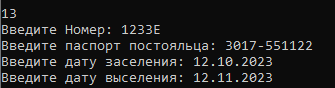
1. поиск гостиничного номера по «номеру гостиничного номера»



1. поиск гостиничного номера по фрагментам «Оборудования»



13) регистрацию вселения постояльца



14) регистрацию выселения постояльца



**Заключение**

Изучены структуры данных и алгоритмы их обработки, а также получены практические навыки их использования при разработке программ. Разработана информационная система для заданной предметной области с использованием заданных структур данных и алгоритмов.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Павлов, Первова: “Структуры и алгоритмы обработки данных”.
2. Дроздов С. Н.: “Структуры и алгоритмы обработки данных:”
3. <https://habr.com>
4. <https://www.cyberforum.ru>
5. <https://stackoverflow.com>

**Приложение А. Текст программы**

**booking.h**

#pragma once

#ifndef booking\_h

#define booking\_h

#include *<iostream>*

**using** **namespace** **std**;

**class** **Node** {

**public**:

string passport;

string room;

string goesin;

string goesout;

Node\* next;

Node\* prev;

Node(string p, string r, string in, string out) {

passport = p;

room = r;

goesin = in;

goesout = out;

**this**->next = **nullptr**;

**this**->prev = **nullptr**;

}

};

**class** **LinkedList** {

**public**:

Node\* head;

Node\* tail;

LinkedList() {

**this**->head = **nullptr**;

**this**->tail = **nullptr**;

}

void add(string p, string r, string in, string out) {

Node\* node = **new** Node(p, r, in, out);

**if** (head == **nullptr**) {

head = node;

tail = node;

}

**else** {

tail->next = node;

node->prev = tail;

tail = node;

}

}

Node\* remove\_by\_room(string room) {

Node\* current = head;

**if** (current->room == room)

{

remove(current);

**return** 0;

}

current = current->next;

**while** (current != head) {

**if** (current->room == room)

{

remove(current);

**return** 0;

}

}

**return** 0;

}

void remove(Node\* node) {

**if** (node == head) {

head = node->next;

**if** (head != **nullptr**) {

head->prev = **nullptr**;

}

}

**else** **if** (node == tail) {

tail = node->prev;

**if** (tail != **nullptr**) {

tail->next = **nullptr**;

}

}

**else** {

node->prev->next = node->next;

node->next->prev = node->prev;

}

**delete** node;

}

void print() {

Node\* node = head;

**while** (node != **nullptr**) {

cout << node->room << " "<<node -> passport<<" "<< node->goesin<<" "<<node->goesout<<"**\n**";

node = node->next;

}

cout << endl;

}

};

#endif

**guests.h**

﻿#pragma once

#ifndef guests\_h

#define guests\_h

#include *<iostream>*

#include *<string>*

#include *<vector>*

**using** **namespace** **std**;

*// Класс, представляющий постояльца отеля*

**class** **Guest** {

**public**:

Guest(string passNum, string fullName, int birthYear, string address, string purpose)

: passportNumber(passNum), fullName(fullName), birthYear(birthYear), address(address), purpose(purpose) {}

string passportNumber; *// Номер паспорта*

string fullName; *// ФИО*

int birthYear; *// Год рождения*

string address; *// Адрес*

string purpose; *// Цель прибытия*

};

*// Класс, представляющий элемент хеш-таблицы*

**class** **Entry** {

**public**:

Guest\* guest; *// Указатель на объект типа Guest*

bool deleted; *// Флаг, показывающий, был ли элемент удален*

Entry() : guest(**nullptr**), deleted(false) {}

};

*// Класс, представляющий хеш-таблицу*

**class** **HashTable** {

**public**:

HashTable() : size(10), count(0), table(size) {}

*// Функция для добавления элемента в хеш-таблицу*

void put(string passNum, string fullName, int birthYear, string address, string purpose) {

**if** (count >= size / 2) {

resize();

}

int hash = getHash(passNum); *// Вычисляем хеш-код*

**while** (table[hash].guest != **nullptr** && table[hash].guest->passportNumber != passNum) {

hash = (hash + 1) % size;

}

**if** (table[hash].guest == **nullptr**) {

table[hash].guest = **new** Guest(passNum, fullName, birthYear, address, purpose);

count++;

}

**else** {

table[hash].guest->fullName = fullName;

table[hash].guest->birthYear = birthYear;

table[hash].guest->address = address;

table[hash].guest->purpose = purpose;

}

}

*// Функция для получения элемента из хеш-таблицы*

Guest\* get(string passNum) {

int hash = getHash(passNum); *// Вычисляем хеш-код*

**while** (table[hash].guest != **nullptr**) {

**if** (table[hash].guest->passportNumber == passNum) {

**return** table[hash].guest;

}

hash = (hash + 1) % size;

}

**return** **nullptr**;

}

*// Функция для удаления элемента из хеш-таблицы*

void remove(string passNum) {

int hash = getHash(passNum); *// Вычисляем хеш-код*

**while** (table[hash].guest != **nullptr**) {

**if** (table[hash].guest->passportNumber == passNum) {

table[hash].deleted = true;

count--;

**return**;

}

hash = (hash + 1) % size;

}

}

void show( string passport) {

Guest\* guest = get(passport);

**if** (guest != **nullptr**) {

cout << "ФИО: " << guest->fullName << endl;

cout << "Год рождения: " << guest->birthYear << endl;

cout << "Адрес: " << guest->address << endl;

cout << "Цель прибытия: " << guest->purpose << endl;

}

**else**

{

cout << "Error, try again**\n**";

}

}

void show\_by\_name(string FIO) {

**for** (**auto** i : table) {

**if** (i.guest != **nullptr** ) {

**if** (FIO != i.guest->fullName) **continue**;

cout << "ФИО: " << i.guest->fullName << endl;

cout << "Год рождения: " << i.guest->birthYear << endl;

cout << "Адрес: " << i.guest->address << endl;

cout << "Цель прибытия: " << i.guest->purpose << endl;

}

}

}

void show\_all() {

**for** (**auto** i : table) {

**if** (i.guest != **nullptr** && i.deleted==false) {

cout << "**\n**ФИО: " << i.guest->fullName << endl;

cout << "Год рождения: " << i.guest->birthYear << endl;

cout << "Адрес: " << i.guest->address << endl;

cout << "Цель прибытия: " << i.guest->purpose << "**\n\n**";

}

}

}

void remove\_all() {

**for** (**auto** i : table) {

**if** (i.guest != **nullptr**) {

i.deleted = true;

count--;

}

}

}

**private**:

int size;

int count; *// Количество элементов в хеш-таблице*

vector<Entry> table; *// Массив элементов типа Entry*

*// Функция для вычисления хеш-кода*

int getHash(string passNum) {

int hash = 0;

**for** (char c : passNum) {

hash = (hash \* 31 + c) % size;

}

**return** hash;

}

*// Функция для изменения размера хеш-таблицы*

void resize() {

size \*= 2;

vector<Entry> newTable(size);

**for** (**auto**& entry : table) {

**if** (entry.guest != **nullptr** && !entry.deleted) {

int hash = getHash(entry.guest->passportNumber);

**while** (newTable[hash].guest != **nullptr**) {

hash = (hash + 1) % size;

}

newTable[hash].guest = entry.guest;

}

}

table = newTable;

}

*// Размер хеш-таблицы*

};

*//*

*//int main()*

*//{*

*// setlocale(LC\_ALL, "RUS");*

*// HashTable hotel;*

*// // Добавление элементов*

*// hotel.put("1234-567890", "Иванов Иван Иванович", 1980, "Москва, ул. Пушкина, д.1", "Отдых");*

*// hotel.put("2345-678901", "Петров Петр Петрович", 1990, "Санкт-Петербург, ул. Ломоносова, д.2", "Работа");*

*// hotel.put("3456-789012", "Сидоров Сидор Сидорович", 1975, "Казань, ул. Баумана, д.3", "Отдых");*

*//*

*// // Получение элемента*

*// Guest\* guest = hotel.get("1234-567890");*

*// if (guest != nullptr) {*

*// cout << "ФИО: " << guest->fullName << endl;*

*// cout << "Год рождения: " << guest->birthYear << endl;*

*// cout << "Адрес: " << guest->address << endl;*

*// cout << "Цель прибытия: " << guest->purpose << endl;*

*// }*

*// else {*

*// cout << "Постоялец не найден" << endl;*

*// }*

*//*

*// // Удаление элемента*

*// hotel.remove("2345-678901");*

*//*

*// return 0;*

*//*

*//}*

#endif

**Header.h**

#pragma once

main.cpp

#include *"rooms.h"*

#include *"guests.h"*

#include *"booking.h"*

#include*<windows.h>*

#define \_CRT\_SECURE\_NO\_WARNINGS

int main() {

setlocale(LC\_ALL, "RUS");

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

node\* rooms = **new** node(5, 'A');

**for** (int i = 0; i < 10; i++) {

insert(rooms, i);

}

*//guests*

string passport;

string FIO;

int year;

string city;

string purpose;

*//rooms*

int num;

string technic;

char let;

int capacity;

bool bathroom;

*//booking*

LinkedList booking;

string room, goesin, goesout;

HashTable visitors;

*//*

booking.add("1234-567890", "123G", "we", "sd");

booking.add("3232232", "1223e", "df", "dfff");

booking.add("3232232", "1223e", "df", "dfff");

visitors.put("1234-567890", "\_\_", 1980, ", . , .1", "");

visitors.put("2345-678901", "\_\_", 1990, "-, . , .2", "");

visitors.put("3456-789012", "\_\_", 1975, ", . , .3", "");

**while** (1)

{

std::cout << "------------------------- -------------------------**\n**";

std::cout << "1) |**\n**";

std::cout << "2) |**\n**";

std::cout << "3) |**\n**";

std::cout << "4) |**\n**";

std::cout << "5) |**\n**";

std::cout << "6) |**\n**";

std::cout << "7) |**\n**";

std::cout << "8) |**\n**";

std::cout << "9) |**\n**";

std::cout << "10) |**\n**";

std::cout << "11) |**\n**";

std::cout << "12) |**\n**";

std::cout << "13) |**\n**";

std::cout << "14) |**\n**";

std::cout << "------------------------------------------------------------------**\n**";

int var;

std::cin >> var;

**switch** (var)

{

**case** 1:

std::cout << " : ";

std::cin >> passport;

std::cout << " : ";

std::cin >> FIO;

std::cout << " : ";

std::cin >> year;

std::cout << " : ";

std::cin >> city;

std::cout << " : ";

std::cin >> purpose;

visitors.put(passport, FIO, year, city, purpose);

std::cout << ".**\n**";

**break**;

**case** 2:

std::cout << " : ";

std::cin >> passport;

visitors.remove(passport);

std::cout << ".**\n**";

**break**;

**case** 3:

visitors.show\_all();

**break**;

**case** 4:

std::cout << " : ";

std::cin >> passport;

visitors.remove(passport);

std::cout << ".**\n**";

**break**;

**case** 5:

std::cout << " : ";

std::cin >> passport;

visitors.show(passport);

**break**;

**case** 6:

std::cout << " : ";

std::cin>>FIO;

visitors.show\_by\_name(FIO);

std::cout << ".**\n**";

**break**;

**case** 7:

std::cout << " : ";

*//int k, unsigned char t = 'A', int cap = 2, string te = "", bool b = 1*

std::cin >> num;

std::cout << " : ";

std::cin >> let;

std::cout << " : ";

std::cin >> capacity;

std::cout << " \_: ";

std::cin >> technic;

std::cout << " 1 0 : ";

std::cin >> bathroom;

insert(rooms, num, let, capacity, technic, bathroom);

std::cout << ".**\n**";

**break**;

**case** 8:

std::cout << " : ";

std::cin >> num;

remove(rooms, num);

std::cout << ".**\n** ";

**break**;

**case** 9:

walk(rooms);

std::cout << "**\n**.**\n** ";

**break**;

**case** 10:

rooms = **new** node(5, 'A');

**break**;

**case** 11:

std::cout << " : ";

std::cin >> num;

serch(rooms, num);

**break**;

**case** 12:

std::cout << " : ";

std::cin >> technic;

find\_by\_technic(rooms, technic);

**break**;

**case** 13:

std::cout << " : ";

std::cin >> room;

std::cout << " : ";

std::cin >> passport;

std::cout << " : ";

std::cin >> goesin;

std::cout << " : ";

std::cin >> goesout;

booking.add(passport, room, goesin, goesout);

**break**;

**case** 14:

std::cout << " : ";

std::cin >> room;

booking.remove\_by\_room(room);

**break**;

**case** 15:

booking.print();

**break**;

**default**:

**break**;

}

}

**return** 0;

}

**myr.cpp**

#include *"myr.h"*

int maxx(int a, int b)

{

**return** (a > b) ? a : b;

}

bool search\_myr(std::string txt, std::string pat)

{

int m = pat.length();

int n = txt.length();

int badchar[NO\_OF\_CHARS];

int s = 0; *// s is shift of the pattern with respect to text*

**while** (s <= (n - m))

{

int j = m - 1;

**while** (j >= 0 && pat[j] == txt[s + j])

j--;

**if** (j < 0)

{

**return** 1;

}

**else**

s += maxx(1, j - badchar[txt[s + j]]);

}

**return** 0;

}

**myr.h**

#pragma once

#ifndef myr\_h

#define myr\_h

#include *<string>*

#include *<vector>*

# include *<limits.h>*

# define NO\_OF\_CHARS 256

**using** **namespace** **std**;

int maxx(int a, int b);

bool search\_myr(std::string txt, std::string pat);

#endif

**rooms.cpp**

#include *"rooms.h"*

#include *"myr.h"*

node::node(int k, unsigned char t , int cap, string te , bool b )

{

key = k; type = t;

capacity = cap; technic = te;

bathroom = b; left = right = 0;

height = 1; room = to\_string(key)+ to\_string(type);

}

node\* find(node\* tree, int v)

{

node\* cur = tree;

**while** (cur)

{

**if** (cur->key < v)

{

cur = cur->right;

}

**else** **if** (cur->key > v) {

cur = cur->left;

}

**if** (cur and cur->key == v)

{

**return** cur;

}

}

**return** 0;

}

unsigned char height(node\* p)

{

**return** p ? p->height : 0;

}

int bfactor(node\* p)

{

**return** height(p->right) - height(p->left);

}

void fixheight(node\* p)

{

unsigned char hl = height(p->left);

unsigned char hr = height(p->right);

p->height = (hl > hr ? hl : hr) + 1;

}

node\* rotateright(node\* p)

{

node\* q = p->left;

p->left = q->right;

q->right = p;

fixheight(p);

fixheight(q);

**return** q;

}

node\* rotateleft(node\* q)

{

node\* p = q->right;

q->right = p->left;

p->left = q;

fixheight(q);

fixheight(p);

**return** p;

}

node\* balance(node\* p)

{

fixheight(p);

**if** (bfactor(p) == 2)

{

**if** (bfactor(p->right) < 0)

p->right = rotateright(p->right);

**return** rotateleft(p);

}

**if** (bfactor(p) == -2)

{

**if** (bfactor(p->left) > 0)

p->left = rotateleft(p->left);

**return** rotateright(p);

}

**return** p; *//*

}

node\* insert(node\* p, int k, unsigned char t , int cap , string te, bool b )*//////////////////////////////////*

{

**if** (!p) **return** **new** node(k, t, cap, te, b);

**if** (k < p->key)

p->left = insert(p->left, k, t, cap, te, b);

**else**

p->right = insert(p->right, k, t, cap, te, b);

**return** balance(p);

}

node\* findmin(node\* p)

{

**return** p->left ? findmin(p->left) : p;

}

node\* findmax(node\* p)

{

**return** p->right ? findmax(p->right) : p;

}

node\* removemin(node\* p)

{

**if** (p->left == 0)

**return** p->right;

p->left = removemin(p->left);

**return** balance(p);

}

node\* removemax(node\* p)

{

**if** (p->right == 0)

**return** p->left;

p->right = removemax(p->right);

**return** balance(p);

}

node\* remove(node\* p, int k)

{

**if** (!p) **return** 0;

**if** (k < p->key)

p->left = remove(p->left, k);

**else** **if** (k > p->key)

p->right = remove(p->right, k);

**else** *// k == p->key*

{

node\* q = p->left;

node\* r = p->right;

**delete** p;

**if** (!r) **return** q;

node\* min = findmin(r);

min->right = removemin(r);

min->left = q;

**return** balance(min);

}

**return** balance(p);

}

void show(node\* root, int space)

{

**if** (!root)

**return**;

**enum** { COUNT = 2 };

space += COUNT;

show(root->right, space);

**for** (int i = COUNT; i < space; ++i)

std::cout << " ";

std::cout << root->key<<root->type << std::endl;

show(root->left, space);

}

void find\_by\_technic(node\* tree, string technic)

{

**if** (tree == NULL) **return**;

find\_by\_technic(tree->left, technic);

**if** (search\_myr(tree->technic, technic))

{

cout << "Node content: " << tree->key << " " << tree->type << " " << tree->capacity << " " << tree->technic << " " << tree->bathroom << endl;

}

find\_by\_technic(tree->right, technic);

}

void serch(node\* tree, int key)

{

node\* current = tree;

int i = 0;

**while** (i <= tree->height)

{

i++;

**if** (current == **nullptr**)

{

**return**;

}

cout << current->key << current->type<<"**\n**: "<<current->capacity<<"**\n**: "<<current->technic<<" : "<<current->bathroom<<"**\n**";

**if** (current->key != key)

{

**if** (key > current->key)

{

current = current->right;

}

**else** **if** (key < current->key)

{

current = current->left;

}

}

**else**

{

**break**;

}

}

}

void walk(node\* tree)

{

**if** (tree)

{

cout << tree->room<<tree->type << " ";

walk(tree->left);

walk(tree->right);

}

}

**rooms.h**

#pragma once

#ifndef rooms\_h

#define rooms\_h

#include *<iostream>*

#include *<string>*

#include *<vector>*

# include *<limits.h>*

# define NO\_OF\_CHARS 256

#include *"myr.h"*

**using** **namespace** **std**;

**struct** **node** *//*

{

int key, capacity;

string room, technic;

bool bathroom;

unsigned char height, type;

node\* left = **nullptr**;

node\* right = **nullptr**;

node\* top = **this**;

node(int k, unsigned char t = 'A', int cap = 2, string te = "", bool b = 1);

};

node\* find(node\* tree, int v);

unsigned char height(node\* p);

void fixheight(node\* p);

int bfactor(node\* p);

node\* rotateright(node\* p);

node\* rotateleft(node\* q);

node\* balance(node\* p);

node\* insert(node\* p, int k, unsigned char t = 'A', int cap = 2, string te = "-", bool b = 1);

node\* find(node\* tree, int v);

node\* findmin(node\* p);

node\* findmax(node\* p);

node\* removemin(node\* p);

node\* removemax(node\* p); *// p*

node\* remove(node\* p, int k); *// k p*

void show(node\* root, int space = 0);

void find\_by\_technic(node\* tree, string technic);

void serch(node\* tree, int key);

void walk(node\* tree);

#endif