

ОГЛАВЛЕНИЕ

1	Задание на курсовой проект	3
2	Алгоритмы и структуры данных	4
2.1	Структуры.....	4
2.1.1	Хеш-таблица.....	4
2.1.2	AVL-дерево	6
2.1.3	Список.....	7
2.2	Алгоритмы.....	9
2.2.1	Алгоритм поиска слова в тексте Боуера и Мура (БМ)	9
3	Описание программы.....	11
4	Тестирование программы.....	13
	Заключение	20
	ПРИЛОЖЕНИЕ А	21

Введение

В настоящее время электронным вычислительным машинам (ЭВМ) приходится не только считывать и выполнять определенные алгоритмы, но и хранить значительные объемы информации. При этом к хранимой информации нужно иметь быстрый доступ. Она в некотором смысле представляет собой абстракцию того или иного фрагмента реального мира и состоит из определенного множества данных, относящихся к какому-либо объекту или предметной области.

В контексте гостиничного бизнеса, электронные вычислительные машины (ЭВМ) играют ключевую роль, так как не только выполняют алгоритмы и хранят информацию, но и обеспечивают эффективность и быстродействие работы с данными. Важность этих аспектов резко возрастает при управлении гостиничным бизнесом, где необходимо оперировать большими объемами информации: данные о бронировании, информация о гостях, данные об услугах и так далее.

Абстрактные структуры данных, такие как списки, деревья и хеш-таблицы, в этом смысле становятся неотъемлемой частью программного обеспечения для гостиничного бизнеса. Они обеспечивают эффективное представление и обработку данных, таких как записи о клиентах, доступные номера, историю бронирований и так далее.

Именно о таких абстрактных структурах данных и алгоритмах работы с ними пойдет речь в этой работе.

1 Задание на курсовой проект

Задачей курсовой работы в соответствии с вариантом является разработка программы для заданной предметной области «Регистрация постояльцев в гостинице» с использованием заданных структур данных.

Расчет варианта производился по формуле:

$$N_{\text{вар}} = nnn \bmod K,$$

где $N_{\text{вар}}$ – номер варианта; nnn – три последние цифры студенческого билета; K – количество вариантов заданий.

В соответствие с этим получаем:

- предметная область – $812 \bmod 6 = 2$ регистрация постояльцев в гостинице;
- метод хеширования – $812 \bmod 4 = 0$ открытое хеширование;
- метод сортировки – $812 \bmod 7 = 0$ подсчётом;
- вид списка – $812 \bmod 5 = 2$ циклический однонаправленный;
- метод обхода дерева – $812 \bmod 3 = 2$ прямой;
- алгоритм поиска слова в тексте – $812 \bmod 2 = 0$ Боуера и Мура

2 Алгоритмы и структуры данных

2.1 Структуры

2.1.1 Хеш-таблица

Алгоритм работы с хеш-функцией с открытым хешированием - это метод, используемый для эффективного хранения и поиска элементов данных (например, массив, состоящий из записей о постояльцах в гостинице) в хеш-таблице. Хеш-функция преобразует ключ, связанный с каждым элементом данных (например, паспортные данные постояльца), в индекс массива, где этот элемент данных предполагается хранить. Открытое хеширование, также известное как хеширование с разрешением коллизий методом разрешения конфликтов, представляет собой один из способов управления коллизиями, то есть ситуациями, когда разные ключи хешируют в один и тот же индекс.

Для разрешения коллизий я использовала связанные списки (цепочки): хеш-таблица состоит из ячеек, каждая из которых указывает на голову связанного списка элементов, имеющих один и тот же хеш-индекс. Если происходит коллизия, элемент добавляется в список соответствующей ячейки.

Хеш-таблица и её функции в данной программе реализуют базовую систему управления данными гостей в контексте гостиничного бизнеса. Основой системы является структура `Guest`, представляющая собой запись о постояльце гостиницы. Структура `hashMap` представляет собой хеш-таблицу с открытым хешированием для эффективного хранения и поиска данных.

Структура `Guest` содержит данные о постояльце, включая его паспортные данные, ФИО, год рождения, адрес и цель прибытия. В структуре также есть указатель `next` для связи с другим `Guest` в случае возникновения коллизии в хеш-таблице.

Использование хеш-таблиц с открытым хешированием позволяет обеспечить быстрый доступ к данным при оптимально подобранной хеш-функции и эффективном методе разрешения коллизий, что особенно критично для систем, оперирующих большим объемом информации.

Давайте рассмотрим описание функций с точки зрения их логической роли в программе:

Работа с однонаправленным списком `struct Guest`:

`push_guest` - Добавляет нового постояльца в конец однонаправленного списка. Если список пуст, добавляемый элемент становится головой списка.

`pop_guest` - Удаляет постояльца из списка по номеру паспорта. Если удаляемый постоялец является головой списка, заменяет голову следующим элементом.

`print_guests` - Печатает информацию о всех постояльцах в списке, перебирая элементы от головы к хвосту.

`size_guests_list` - Возвращает количество постояльцев в списке, пересчитывая элементы от головы к хвосту.

`free_list` - Освобождает память, занимаемую списком постояльцев, полностью очищая его.

Работа с хеш-таблицей `struct hashMap`:

`hashFunction` - Вычисляет хеш-код на основе номера паспорта, чтобы определить индекс для хранения элемента в хеш-таблице.

`insert` - Добавляет нового постояльца в хеш-таблицу, используя хеш-функцию для определения индекса и добавляя запись в соответствующий список.

`erase` - Удаляет постояльца из хеш-таблицы по номеру паспорта, находя нужный список через хеш-функцию и затем выполняя удаление из списка.

`inHashTable` - Проверяет наличие постояльца в хеш-таблице по номеру паспорта, возвращая `'true'`, если постоялец найден.

`searchByPassport` - Ищет постояльца в хеш-таблице по номеру паспорта, возвращая указатель на найденную запись.

`searchByName` - Ищет всех постояльцев с заданным ФИО в хеш-таблице, возвращая список найденных записей.

`printTable` - Печатает содержимое хеш-таблицы, выводя информацию о постояльцах в каждом из списков.

Эти функции реализуют основную логику программы, позволяющую пользователю эффективно управлять списком постояльцев гостиницы.

2.1.2 AVL-дерево

AVL-дерево – это самобалансирующееся бинарное дерево поиска, в котором для каждого узла разность высот левого и правого поддеревьев, называемая балансирующим фактором, не превышает 1. Это означает, что AVL-дерево всегда остается относительно сбалансированным, и, как результат, операции поиска, вставки и удаления элементов могут быть выполнены с высокой производительностью. При вставке или удалении узлов AVL-дерево выполняет ряд поворотов, чтобы восстановить утраченное свойство балансировки.

Прямой (или префиксный) обход дерева означает обход дерева в порядке: корень, левое поддерево, правое поддерево. Этот метод обхода позволяет выполнять операции над узлами дерева (например, их вывод на экран или обработку) в потоке от корня к листьям.

Прямой обход AVL-дерева позволяет эффективно проходить все узлы дерева, используя систематический подход, и обеспечивает быстрый и организованный доступ к данным, сохраняя при этом приемлемую скорость выполнения операций вблизи логарифмической сложности благодаря свойствам балансировки AVL-деревьев.

Структура `HotelRoom` хранит основной узел AVL-дерева, хранящий информацию о номерах гостиницы, включая номер комнаты, количество мест, количество комнат в номере, наличие санузла и оборудование. Также содержит указатели на левое и правое поддерева и значение высоты для текущего узла.

Работа со структурой данных для управления номерами `struct HotelRoom`:

`height` и `getBalance` - Расчёт высоты узла и его баланс-фактора (разность высот правого и левого поддеревьев).

`updateHeight` - Обновление высоты узла после изменений в дереве.

`rotateRight` и `rotateLeft` - Выполнение правого и левого поворотов для балансировки дерева.

`balance` - Балансировка поддерева с корректировкой его структуры для поддержания свойств AVL-дерева.

`insert` - Вставка нового элемента с сохранением баланса дерева.

`inHotelRoom` - Проверка наличия номера в дереве.

`findMin` и `removeMin` - Нахождение минимального элемента в поддереве и его удаление.

`remove` - Удаление номера из дерева с восстановлением баланса.

`clearHotelRoom` - Очистка всего дерева (удаление всех номеров).

`printHotelRoom` - Печать информации о номерах, используя прямой обход дерева.

`searchNumber` - Поиск и возврат информации о номере по его идентификатору.

`searchFacilityInHotelRooms` - Поиск номеров с определённым оборудованием через AVL-дерево.

Эти функции обеспечивают эффективное взаимодействие пользователя с данными о номерах гостиницы (выполняют добавление, удаление, поиск или вывод всех номеров, а также специализированный поиск по оборудованию). AVL-дерево выбрано в качестве структуры данных для обеспечения быстрых операций вставки, удаления и поиска, что важно для поддержания эффективности работы программы при управлении большим количеством номеров.

2.1.3 Список

Однонаправленный циклический список — это разновидность связанного списка, где каждый узел содержит ссылку на следующий узел в

последовательности, а последний узел списка связан с первым, образуя цикл. Это структура данных, которая поддерживает такую же последовательность элементов, как обычный связный список, но делает самый последний и первый элементы смежными друг другу. Алгоритмы для работы с такой структурой данных включают операции добавления, удаления, поиска и другие, необходимые для управления списком.

Данная структура программы предназначена для управления регистрационными данными постояльцев в рамках системы учета гостиницы. Она использует односвязный циклический список `TrafficData` для отслеживания заселений и выселений клиентов, а также AVL-дерево `HotelRoom` и хеш-таблицу `hashMap` для управления информацией о номерах и постояльцах.

Работа со структурой данных `struct TrafficData`:

`add` - Добавляет новую запись о вселении в циклический список. Если список пуст, созданный узел становится первым и последним элементом, указывая сам на себя. В противном случае узел добавляется в конец списка.

`del` - удаляет запись из списка по паспорту и номеру комнаты. Учитывает несколько случаев: удаление единственного элемента, удаление первого элемента (при этом изменяется указатель головы списка) и удаление элемента в середине или конце списка.

`inTrafficData` - Проверяет наличие записи в списке путём перебора всех элементов до возврата в начало списка.

`insertSorted` и `insertionSort` - Функции для сортировки списка вставками по номеру паспорта. `insertSorted` вставляет узел в уже отсортированную часть списка, а `insertionSort` полностью сортирует весь список, используя `insertSorted`.

`clearTrafficData` - Очищает список, последовательно удаляя все его элементы и освобождая занятую ими память.

Эта программа демонстрирует эффективное использование однонаправленного циклического списка для управления динамическим

набором данных в контексте регистрации вселений и выселений в гостинице. Сочетание процедур валидации данных, управления памятью и алгоритмов сортировки обеспечивает надёжный и удобный способ обработки информации о постояльцах.

2.2 Алгоритмы

2.2.1 Алгоритм поиска слова в тексте Бойера и Мура (БМ)

Алгоритм поиска Бойера-Мура — это эффективный алгоритм поиска строки, который работает, сравнивая символы с конца подстроки, а не с начала, как многие другие алгоритмы поиска. Такой подход часто приводит к меньшему количеству сравнений символов и к более быстрому поиску в небольших текстах, а также в больших текстах с малым количеством возвратов назад.

В алгоритме используются две эвристики:

Эвристика плохого символа: Если символ строки не совпадает с соответствующим символом подстроки, мы можем быть уверены, что начало подстроки не может находиться между несовпадающим символом и позицией, в которой сдвиг начался. Поэтому сдвиг может быть сделан дальше. Алгоритм создает таблицу сдвигов для всех возможных символов (в нашем случае общее количество символов 256). Сначала инициализирует все значения таблицы как длину подстроки, затем перебирает символы подстроки (кроме последнего) и записывает для каждого символа расстояние от его последнего вхождения до конца подстроки.

Эвристика хорошего суффикса: Если суффикс подстроки совпадает, но символ перед суффиксом не совпадает, мы сдвигаем подстроку так, чтобы её самый правый появившийся ранее суффикс совпадал с её самым левым суффиксом.

В своей программе я реализовала эвристику плохого символа.

Описание функций с точки зрения логики программы:

BMsearch - Эта функция реализует алгоритм поиска Боуера-Мура. Основная идея алгоритма заключается в том, что сравнения между подстрокой и отрезком строки проводятся справа налево. Если символ не совпадает, алгоритм пытается "пропустить" часть символов согласно заранее подготовленной таблице сдвигов, что позволяет уменьшить общее количество сравнений.

searchFacilityInHotelRooms - Эта функция осуществляет поиск номеров гостиницы, в описании которых содержится заданное оборудование. Поиск ведётся по всему AVL-дереву, представляющему собой структуру данных, используемую для хранения номеров. Реализуется рекурсивный обход по всем узлам дерева (левое поддерево, текущий узел, правое поддерево).

searchByEquipment - Эта функция предоставляет интерфейс для пользователя для ввода названия искомого оборудования и выводит список номеров, в которых данное оборудование присутствует.

Эти функции обеспечивают быстрый и эффективный поиск по данным, хранящимся в специализированных структурах, что позволяет пользователю легко находить номера с нужными характеристиками в базе данных гостиницы.

3 Описание программы

Программа "Система регистрации постояльцев в гостинице" представляет собой комплексное программное обеспечение для управления данными о постояльцах и гостиничных номерах, а также для отслеживания процессов вселения и выселения клиентов в отеле.

Функциональное назначение:

Программа позволяет выполнять следующие функции:

- Регистрация нового постояльца: Ввод personal данных, включая номер паспорта, ФИО, год рождения, адрес и цель прибытия.
- Удаление данных о постояльцах: С удалением связанных записей о заселении, при условии, что постоялец выселен.
- Просмотр зарегистрированных постояльцев: Возможность просмотра всех зарегистрированных клиентов и их информации.
- Очистка данных о постояльцах: Удаление всех записей о клиентах из системы.
- Поиск постояльца: По номеру паспорта или ФИО, с отображением всех данных о найденном постояльце, включая информацию о заселении.
- Добавление нового гостиничного номера: Ввод данных о номере, включая его номер, тип, количество мест и комнат, наличие санузла и оборудование.
- Удаление данных о гостиничном номере: Удаление информации о номере, при условии, что он свободен.
- Просмотр всех имеющихся номеров: Визуализация данных обо всех доступных гостиничных номерах.
- Очистка данных о гостиничных номерах: Полное удаление информации о номерах из системы.

- Поиск гостиничного номера: По уникальному номеру или по оборудованию, с отображением полной информации о номере и его постояльцах.
- Регистрация вселения: Заселение клиента в номер, с проверкой наличия свободных мест.
- Регистрация выселения: Выселение клиента, освобождение номера.

Особенности работы с данными:

Хранение данных о постояльцах: В виде хеш-таблицы с номером паспорта в качестве первичного ключа.

Хранение данных о номерах: В виде AVL-дерева, упорядоченного по номерам гостиничных номеров.

Хранение данных о вселении и выселении: В виде списка, упорядоченного по номеру номера, с учётом возможности проживания нескольких постояльцев в одном многоместном номере.

Интерфейс и управление:

Программа интуитивно понятна и легка в использовании, с четко организованным пользовательским интерфейсом, который включает:

- Формы для ввода и обработки информации о постояльцах и номерах.
- Функции поиска и фильтрации для быстрого доступа к требуемым данным.
- Отчеты и индикаторы состояния для регистрации вселения и выселения.

4 Тестирование программы

На рисунке 1 представлено основное меню программы.

```
Выберите, с какими данными нужно осуществить работу:
+-----+
| 1. | Постоялец |
+-----+
| 2. | Гостиничный номер |
+-----+
| 3. | Выселение и поселение постояльцев |
+-----+
| 0. | Завершение работы |
+-----+
Ваш выбор: |
```

Рисунок 1 – основное меню программы

На рисунке 2 представлено меню программы для работы с постояльцами.

```
Выберите действие:
+-----+
| 1. | Регистрация нового постояльца |
+-----+
| 2. | Удаление данных о постояльце |
+-----+
| 3. | Просмотр всех зарегистрированных |
+-----+
| 4. | Очистка данных о постояльцах |
+-----+
| 5. | Поиск по номеру паспорта |
+-----+
| 6. | Поиск по ФИО |
+-----+
| 0. | Вернуться в главное меню |
+-----+
Ваш выбор: |
```

Рисунок 2 – меню работы с постояльцами

На рисунке 3 представлена регистрация нового постояльца.

```
Заполните данные о постояльце:
Номер паспорта (строка формата NNNN–NNNNNN, где N – цифры): 1234–123456
ФИО: Иванов В.А.
Год рождения: 2005
Адрес: Москва
Цель прибытия: отдых
Для продолжения нажмите любую клавишу . . . |
```

Рисунок 3 – регистрация нового постояльца

На рисунке 4 показана попытка добавления уже зарегистрированного постояльца.

```
Заполните данные о постояльце:  
Номер паспорта (строка формата NNNN-NNNNNN, где N – цифры): 1234-123456  
Этот человек уже есть в базе!  
Для продолжения нажмите любую клавишу . . . |
```

Рисунок 4 – регистрация уже зарегистрированного постояльца

Далее я зарегистрировала ещё четырёх человек. На рисунке 5 представлен вывод всех постояльцев на экран.

Номер паспорта	ФИО	Год рождения	Адрес	Цель прибытия
1234-123456	Иванов В.А.	2005	Москва	отдых
1234-234567	Сидоров П.Е.	1990	Вологда	отдых
1234-345678	Петров М.Е.	1980	Иваново	отдых
1234-567890	Сидоренко П.И.	2000	Уфа	отдых
1234-456789	Иванов В.А.	1995	Киров	отдых

Для продолжения нажмите любую клавишу . . . |

Рисунок 5 – вывод всех постояльцев

На рисунке 6 представлено удаление постояльца по номеру паспорта.

```
Введите номер паспорта постояльца: 1234-234567  
  
Данные о постояльце удалены!  
Для продолжения нажмите любую клавишу . . . |
```

Рисунок 6 – удаление человека из базы постояльцев

На рисунке 7 представлен результат удаления.

Номер паспорта	ФИО	Год рождения	Адрес	Цель прибытия
1234-123456	Иванов В.А.	2005	Москва	отдых
1234-345678	Петров М.Е.	1980	Иваново	отдых
1234-567890	Сидоренко П.И.	2000	Уфа	отдых
1234-456789	Иванов В.А.	1995	Киров	отдых

Для продолжения нажмите любую клавишу . . . |

Рисунок 7 – результат удаления

На рисунке 8 представлен поиск по номеру паспорта.

```
Введите номер паспорта, по которому хотите посмотреть информацию: 1234-123456  
Номер паспорта  ФИО  Год рождения  Адрес  Цель прибытия  
1234-123456  Иванов В.А.  2005  Москва  отдых  
Для продолжения нажмите любую клавишу . . . |
```

Рисунок 8 – поиск по номеру паспорта

На рисунке 9 представлен поиск по ФИО.

```

Введите ФИО постояльца, информацию о котором хотите посмотреть: Иванов В.А.
Номер паспорта  ФИО          Год рождения      Адрес          Цель прибытия
1234-123456      Иванов В.А.          2005            Москва        отдых
1234-456789      Иванов В.А.          1995            Киров         отдых
Для продолжения нажмите любую клавишу . . . |

```

Рисунок 9 – поиск по ФИО

На рисунке 10 представлена очистка всех данных о постояльцах.

```

Данные о постояльцах очищены!
Для продолжения нажмите любую клавишу . . . |

```

Рисунок 10 – очистка всех данных о постояльцах

На рисунке 11 представлено меню работы с номерами.

```

Выберите действие:
+---+-----+
| 1. | Добавление нового номера |
+---+-----+
| 2. | Удаление сведений о номере  |
+---+-----+
| 3. | Просмотр всех номеров       |
+---+-----+
| 4. | Очистка данных о номерах    |
+---+-----+
| 5. | Поиск по номеру             |
+---+-----+
| 6. | Поиск по оборудованию       |
+---+-----+
| 0. | Вернуться в главное меню    |
+---+-----+
Ваш выбор: |

```

Рисунок 11 – меню работы с номерами

На рисунке 12 представлено добавление нового номера.

```

Заполните данные о номере:
Номер (формат "ANNN"): 0005
Количество мест: 3
Количество комнат: 1
Наличие санузла (y/n): n
Оборудование: Телевизор, Сейф
Для продолжения нажмите любую клавишу . . . |

```

Рисунок 12 – добавление нового номера

На рисунке 13 представлена попытка добавить уже имеющийся в базе номер.

```
Заполните данные о номере:
Номер (формат "ANNN"): 0005
Этот номер уже есть в базе!
Для продолжения нажмите любую клавишу . . . |
```

Рисунок 13 – добавление существующего номера

Далее я добавила информацию о ещё трёх номерах. Результат вывода всех номеров представлен на рисунке 14.

Номер	Количество мест	Количество комнат	Санузел	Оборудование
Л123	3	2	true	Холодильник, Wi-Fi
Л122	2	1	false	Рабочий стол
0003	5	1	true	Холодильник, Мини-бар
0005	3	1	false	Телевизор, Сейф

Рисунок 14 – вывод всех номеров

На рисунке 15 представлен поиск по номеру номера.

```
Введите номер, информацию о котором хотите получить: 0003
Номер    Количество мест  Количество комнат    Санузел  Оборудование
0003     5                 1                   true     Холодильник, Мини-бар
Для продолжения нажмите любую клавишу . . . |
```

Рисунок 15 – поиск по номеру номера

На рисунке 16 представлен поиск по оборудованию.

```
Введите название оборудования: Wi-Fi
Номер    Количество мест  Количество комнат    Санузел  Оборудование
Л122     2                 1                   false    Телевизор, Wi-Fi
Л123     3                 2                   true     Wi-Fi, Холодильник
Для продолжения нажмите любую клавишу . . . |
```

Рисунок 16 – поиск по оборудованию

На рисунке 17 представлено удаление информации о номере.

```
Введите номер, который нужно удалить: Л123
Удаление выполнено!
Для продолжения нажмите любую клавишу . . . |
```

Рисунок 17 – удаление номера

На рисунке 18 показан результат удаления информации о номере из рисунка 17.

Номер	Количество мест	Количество комнат	Санузел	Оборудование
Л122	2	1	false	Телевизор, Wi-Fi

Для продолжения нажмите любую клавишу . . . |

Рисунок 18 – результат удаления

На рисунке 19 показано очищение информации обо всех номерах.


```
Данные о постояльцах очищены!  
Для продолжения нажмите любую клавишу . . . |
```

Рисунок 19 – очищение информации обо всех номерах

На рисунке 20 представлено меню регистрации вселения и выселения.

```
Выберите действие:  
+-----+  
| 1. | Регистрация вселения |  
+-----+  
| 2. | Регистрация выселения |  
+-----+  
| 0. | Вернуться в главное меню |  
+-----+  
Ваш выбор: |
```

Рисунок 20 – меню регистрации вселения и выселения

На рисунке 21 представлена регистрация заселения постояльца в номер (предварительно постоялец и номер должны быть зарегистрированы в соответствующих базах).

```
Введите следующие данные для регистрации вселения:  
Номер паспорта (строка формата NNNN-NNNNNN, где N – цифры): 1234-123456  
Номер гостиничного номера (строка формата "ANNN"): Л123  
Дата заселения: 05.09.2024  
Дата выселения: 10.09.2024  
  
Заселение зарегистрировано!  
Для продолжения нажмите любую клавишу . . . |
```

Рисунок 21 – регистрация заселения

Регистрация заселения влечёт за собой изменения, показанные на рисунках 22 – 25.

На рисунке 22 показана попытка удалить данные о постояльце, который ещё зарегистрирован в номере.

```
Введите номер паспорта постояльца: 1234-123456  
  
Удаление невозможно! Сначала надо зарегистрировать выселение постояльца!  
Для продолжения нажмите любую клавишу . . . |
```

*Рисунок 22 – попытка удалить информацию о постояльце,
зарегистрированном в номер*

На рисунке 23 показана попытка очистить информацию о постояльцах, когда среди них есть зарегистрированные в номера.

A screenshot of a terminal window with a black background and white text. The text reads: "Невозможно очистить данные о всех постояльцах, так как есть зарегистрированные! Для продолжения нажмите любую клавишу . . . |".

Рисунок 23 – попытка очистить информацию о постояльцах

На рисунке 24 показана попытка удалить информацию о номере, в котором кто-то зарегистрирован.

A screenshot of a terminal window with a black background and white text. The text reads: "Введите номер, который нужно удалить: Л123. Удаление невозможно! Сначала надо зарегистрировать выселение постояльца! Для продолжения нажмите любую клавишу . . . |".

*Рисунок 24 - удаление информации о номере, где кто-то
зарегистрирован*

На рисунке 25 показана попытка очистить информацию о номерах, среди которых есть номера с зарегистрированными постояльцами.

A screenshot of a terminal window with a black background and white text. The text reads: "Невозможно очистить данные о всех номерах, так как есть зарегистрированные! Для продолжения нажмите любую клавишу . . . |".

*Рисунок 25 – очищение информации о номерах с зарегистрированными
постояльцами*

На рисунке 26 показана регистрация выселения.

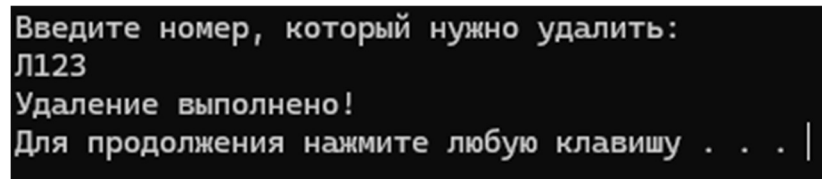
A screenshot of a terminal window with a black background and white text. The text reads: "Введите следующие данные для регистрации выселения: Номер паспорта (строка формата NNNN-NNNNNN, где N – цифры): 1234-123456. Номер гостиничного номера (строка формата "ANNN"): Л123. Регистрация выселения прошла! Для продолжения нажмите любую клавишу . . . |".

Рисунок 26 – регистрация выселения

После регистрации выселения функции представленные на рисунках 22–25 становятся доступными снова. Это показано на рисунках 27 и 28.

A screenshot of a terminal window with a black background and white text. The text reads: "Введите номер паспорта постояльца: 1234-123456. Данные о постояльце удалены! Для продолжения нажмите любую клавишу . . . |".

*Рисунок 27 – удаление данных о постояльце после регистрации
выселения*



```
Введите номер, который нужно удалить:  
Л123  
Удаление выполнено!  
Для продолжения нажмите любую клавишу . . . |
```

Рисунок 28 – удаление данных о номере после регистрации выселения

На этом тестирование можно закончить, видно, что все функции работают верно.

Заключение

Настоящая работа демонстрирует значительный вклад современных технологий обработки данных в сфере гостиничного бизнеса. Электронные вычислительные машины (ЭВМ) не просто предоставляют мощные вычислительные ресурсы для выполнения сложных задач, но и являются ключевым инструментом в организации и обслуживании потоков информации. Как было продемонстрировано в курсовой работе, применение абстрактных структур данных и эффективных алгоритмов позволило создать систему регистрации постояльцев в гостинице, основной задачей которой является обработка больших объемов информации.

Разработанная программа включает в себя реализацию хеш-таблицы с открытым хешированием, AVL-дерева и циклического однонаправленного списка, предоставляющих быстрый поиск, добавление и удаление данных, необходимых для оптимизации работы гостиничного бизнеса. Алгоритм Боуера-Мура был внедрен как часть системы, обеспечивая эффективный поиск данных о комнатах и удобствах, что выделяет систему с точки зрения производительности и удобства использования.

Таким образом, с использованием современных подходов к программированию и обработке данных, представленная система демонстрирует, как технологические решения могут служить основой для повышения эффективности бизнес-процессов в гостиничной отрасли. Высокая скорость работы программы, её надежность и гибкость делают ее не только важным инструментом в работе современной гостиницы, но и образцом для последующих разработок в данной области.

ПРИЛОЖЕНИЕ А

/*

Предметная область: регистрация постояльцев в гостинице

Метод хеширования: открытое хеширование

Метод сортировки: подсчётом

Вид списка: циклический однонаправленный

Метод обхода дерева: прямой

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

*/

```
#include <iostream>
```

```
#include <string>
```

```
#include <windows.h>
```

```
#include <limits.h>
```

```
#include <ios>
```

```
#include <vector>
```

```
//#define NOMINMAX
```

```
#ifdef max
```

```
#undef max
```

```
#endif
```

```
using namespace std;
```

```
//-----список постояльцев-----//
```

```
//структура, хранящая данные о постояльцах (однонаправленный список)
```

```
struct Guest {
```

```
    string passport; // первичный ключ, "NNNN - NNNNNN", где N – цифры
```

```
    string full_name; // ФИО
```

```
    int year_of_birth; // год рождения
```

```

string address; // адрес
string purpose; // цель прибытия
Guest* next;

Guest() {
    next = NULL;
}

Guest(string pass, string name, int y, string addr, string purp) {
    passport = pass;
    full_name = name;
    year_of_birth = y;
    address = addr;
    purpose = purp;
    next = NULL;
}
};

```

//структура, хранящая данные о зарегистрированных вселениях
(она направленный циклический список)

```

struct TrafficData {
    string passport; //"NNNN - NNNNNN", где N – цифры
    string number; //"ANNN"
    string checkin; //дата заселения
    string checkout; //дата выселения
    TrafficData* next;

    TrafficData();

    TrafficData(string p, string num, string in, string out) {

```

```

        passport = p;
        number = num;
        checkin = in;
        checkout = out;
        next = NULL;
    }
};

```

//функция добавления постояльца в список гостей

```

void push_guest(Guest** head, Guest* g) {
    Guest* cur = *head;
    Guest* p = g;
    if (*head == NULL) {
        *head = p;
    }
    else {
        while (cur->next != NULL) {
            cur = cur->next;
        }
        cur->next = p;
    }
    p->next = NULL;
}

```

//функция удаления постояльца из списка гостей

```

void pop_guest(Guest** head, string pass) {
    Guest* cur = *head;
    if (cur->passport == pass) {
        *head = cur->next;
        delete cur;
    }
}

```

```

        return;
    }
    while (cur->next != NULL) {
        if (cur->next->passport == pass) {
            Guest* p = cur->next;
            cur->next = (cur->next)->next;
            delete p;
            return;
        }
        cur = cur->next;
    }
}

```

//функция вывода переданного списка постояльцев

```

void print_guests(Guest* head) {
    Guest* p = head;
    if (p == NULL) {
        cout << "Список постояльцев пуст!\n";
        return;
    }
    while (p->next != NULL) {
        cout << p->passport << '\t' << p->full_name << "\t\t" << p->
        year_of_birth << "\t\t" << p->address << "\t\t" << p->purpose << "\n";
        p = p->next;
    }
    cout << p->passport << '\t' << p->full_name << "\t\t" << p->year_of_birth
    << "\t\t" << p->address << "\t\t" << p->purpose << "\n";
}

```

//функция вычисления размера списка постояльцев


```

int size_guests_list(Guest* head) {
    Guest* p = head;
    int s = 0;
    if (p == NULL) return 0;
    while (p->next != NULL) {
        s++;
        p = p->next;
    }
    return s + 1;
}

```

//функция для очистки переданного списка постояльцев

```

void free_list(Guest** head) {
    Guest* cur = *head;
    if (cur == NULL) return;
    if (cur->next == NULL) {
        delete cur;
        *head = NULL;
        return;
    }
    while (cur != nullptr) {
        Guest* next = cur->next;
        delete cur;
        cur = next;
    }
}

```

//функция проверки корректности введённого номера паспорта

```

void checkPassport(string& passport) {
    cin >> passport;
}

```

```

bool f;
while (true) {
    f = true;
    if (passport.size() != 11) f = false;
    else {
        for (int i = 0; i < 4; ++i) {
            if (passport[i] < '0' || '9' < passport[i]) {
                f = false;
                break;
            }
        }
        if (passport[4] != '-') f = false;
        for (int i = 5; i < 11; ++i) {
            if (passport[i] < '0' || '9' < passport[i]) {
                f = false;
                break;
            }
        }
    }
}

if (f) return;
cin.clear();
cin.ignore(numeric_limits<streamsize>::max(), '\n');
cerr << "Введено некорректное значение! ";
cout << "Повторите ввод: ";
cin >> passport;
}
}

```

//функция проверки корректности введённого года рождения

```

void checkYear(int& n) {
    cin >> n;
    while (!cin.good() || n < 1902 || 2024 < n || cin.get() != '\n') {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cerr << "Введено некорректное значение! ";
        cout << "Повторите ввод: ";
        cin >> n;
    }
}

```

//функция проверки наличия записи в списке зарегистрированных вселений

```

bool inTrafficData(TrafficData* head, string s) {
    if ('0' <= s[0] && s[0] <= '9') {
        if (head == nullptr) {
            return false;
        }
        TrafficData* cur = head;
        do {
            if (cur->passport == s) {
                return true;
            }
            cur = cur->next;
        } while (cur != head);
        return false;
    }
    else {
        if (head == nullptr) {
            return false;
        }
    }
}

```

```

        TrafficData* cur = head;
        do {
            if (cur->number == s) {
                return true;
            }
            cur = cur->next;
        } while (cur != head);
        return false;
    }
}

//-----хеш-таблица с данными о постояльцах-----
-----//

//структура для работы с данными о постояльцах (реализация хеш-таблицы)
struct hashMap {
    int capacity;
    int size;
    Guest* arr[100];

    hashMap() {
        capacity = 100;
        size = 0;
        for (int i = 0; i < 100; ++i)
            arr[i] = nullptr;
    }

    ~hashMap() {
        for (int i = 0; i < 100; ++i) {
            free_list(&arr[i]);
        }
    }
}

```

```

    }

    int hashFunction(string);
    void insert(Guest*);
    void erase(string);
    bool inHashTable(string);
    Guest* searchByPassport(string);
    Guest* searchByName(string);
    void printTable();
};

//метод вычисления хеша
int hashMap::hashFunction(string key) {
    int hash = 0;
    for (int i = 0; i < 4; ++i) {
        hash += int(key[i]);
    }
    for (int i = 5; i < 11; ++i) {
        hash += int(key[i]);
    }
    return hash % capacity;
}

//метод добавления нового элемента в хеш-таблицу
void hashMap::insert(Guest* g) {
    int index = hashFunction(g->passport);
    push_guest(&arr[index], g);
    size++;
}

```

//метод удаления элемента из хеш-таблицы

```
void hashMap::erase(string passport) {  
    int index = hashFunction(passport);  
    pop_guest(&arr[index], passport);  
    size--;  
}
```

//метод проверки наличия элемента в хеш-таблице

```
bool hashMap::inHashTable(string passport) {  
    int index = hashFunction(passport);  
    Guest* ptr = arr[index];  
    while (ptr != NULL) {  
        if (ptr->passport == passport) return true;  
        ptr = ptr->next;  
    }  
    return false;  
}
```

//метод поиска в хеш-таблице по номеру паспорта постояльца

```
Guest* hashMap::searchByPassport(string passport) {  
    Guest* ans = new Guest;  
    for (int i = 0; i < capacity; ++i) {  
        Guest* p = arr[i];  
        while (p != NULL) {  
            if (arr[i]->passport == passport) {  
                ans = p;  
                return ans;  
            }  
            p = p->next;  
        }  
    }  
}
```

```

    }
}

```

//метод поиска в хеш-таблице по ФИО постояльца

```

Guest* hashMap::searchByName(string name) {
    Guest* ans = NULL;
    for (int i = 0; i < capacity; ++i) {
        Guest* p = arr[i];
        while (p != NULL) {
            if (p->full_name == name) {
                push_guest(&ans, p);
            }
            p = p->next;
        }
    }
    return ans;
}

```

//метод вывода хеш-таблицы на экран

```

void hashMap::printTable() {
    if (size == 0) {
        cout << "Список зарегистрированных пуст!\n";
        return;
    }
    cout << "Номер паспорта" << '\t' << "ФИО" << "\t\t" << "Год рождения"
    << '\t' << "Адрес" << "\t\t" << "Цель прибытия" << '\n';
    for (int i = 0; i < capacity; ++i) {
        if (size_guests_list(arr[i]) != 0) {
            print_guests(arr[i]);
        }
    }
}

```

```

    }
}

//-----действия с постояльцами-----
//

//функция для регистрации нового постояльца
void registration(hashMap* table) {
    cout << "Заполните данные о постояльце: \n";
    string passport, name, address, purpose;
    int year;
    cout << "Номер паспорта (строка формата NNNN-NNNNNN, где N -
цифры): ";
    checkPassport(passport);
    if (table->inHashTable(passport)) {
        cout << "Этот человек уже есть в базе!\n";
        return;
    }
    cout << "ФИО: ";
    cin.ignore();
    getline(cin, name);
    cout << "Год рождения: ";
    checkYear(year);
    cout << "Адрес: ";
    //cin.ignore();
    getline(cin, address);
    cout << "Цель прибытия: ";
    //cin.ignore();
    getline(cin, purpose);
}

```



```

    Guest* g = new Guest(passport, name, year, address, purpose);
    table->insert(g);
}

//функция для удаления постояльца
void deliting(hashMap* table, TrafficData* data) {
    if (table->size == 0) {
        cout << "Список зарегистрированных пуст!\n";
        return;
    }
    string passport;
    cout << "Введите номер паспорта постояльца: ";
    checkPassport(passport);
    cin.ignore();

    if (!table->inHashTable(passport)) {
        cout << "\nПостояльца с таким номером паспорта нет в
списке!\n";
        return;
    }

    if (inTrafficData(data, passport)) {
        cout << "\nУдаление невозможно! Сначала надо зарегистрировать
выселение постояльца!\n";
        return;
    }
    table->erase(passport);
    cout << "\nДанные о постояльце удалены!\n";
}

```

//функция поиска постояльца по номеру паспорта

```
void searchPassport(hashMap* table) {  
    cout << "Введите номер паспорта, по которому хотите посмотреть  
информацию: ";  
    string passport;  
    checkPassport(passport);  
  
    if (table->inHashTable(passport)) {  
        Guest* ans = table->searchByPassport(passport);  
        cout << "Номер паспорта" << '\t' << "ФИО" << "\t\t" << "Год  
рождения" << '\t' << "Адрес" << "\t\t" << "Цель прибытия" << '\n';  
        cout << ans->passport << '\t' << ans->full_name << "\t\t" << ans->  
>year_of_birth << "\t\t" << ans->address << "\t\t" << ans->purpose << "\n";  
    }  
    else {  
        cout << "В списке постояльцев нет человека с таким номером  
паспорта!\n";  
    }  
}
```

//функция поиска постояльца по ФИО

```
void searchFIO(hashMap* table) {  
    cout << "Введите ФИО постояльца, информацию о котором хотите  
посмотреть: ";  
    //cin.ignore();  
    string name;  
    getline(cin, name);  
  
    Guest* ans = table->searchByName(name);
```

```

        if (ans == NULL) cout << "В списке постояльцев нет человека с таким
        ФИО!\n";
        else {
            cout << "Номер паспорта" << '\t' << "ФИО" << "\t\t" << "Год
            рождения" << '\t' << "Адрес" << "\t\t" << "Цель прибытия" << '\n';
            print_guests(ans);
        }
        free_list(&ans);
    }
}

```

//функция вывода меню для работы с постояльцами

```

void printGuestMenu() {
    system("cls");
    cout << "Выберите действие: \n";
    cout << "+----+-----+\n";
    cout << "| 1. | Регистрация нового постояльца   |\n";
    cout << "+----+-----+\n";
    cout << "| 2. | Удаление данных о постояльце   |\n";
    cout << "+----+-----+\n";
    cout << "| 3. | Просмотр всех зарегистрированных |\n";
    cout << "+----+-----+\n";
    cout << "| 4. | Очистка данных о постояльцах   |\n";
    cout << "+----+-----+\n";
    cout << "| 5. | Поиск по номеру паспорта       |\n";
    cout << "+----+-----+\n";
    cout << "| 6. | Поиск по ФИО                   |\n";
    cout << "+----+-----+\n";
    cout << "| 0. | Вернуться в главное меню       |\n";
    cout << "+----+-----+\n";
}

```

//функция для реализации работы меню постояльцев

```
void guestMenu(hashMap* table, TrafficData* data) {  
    char menu_item;  
    while (true) {  
        printGuestMenu();  
        cout << "Ваш выбор: ";  
        cin >> menu_item;  
        cin.ignore(numeric_limits<streamsize>::max(), '\n');  
        system("cls");  
        switch (menu_item) {  
            case '1':  
                registration(table);  
                system("pause");  
                break;  
            case '2':  
                deliting(table, data);  
                system("pause");  
                break;  
            case '3':  
                table->printTable();  
                system("pause");  
                break;  
            case '4':  
                if (data == nullptr) {  
                    table->~hashMap();  
                    table->size = 0;  
                    cout << "Данные о постояльцах очищены!\n";  
                }  
                else {
```

```

        cout << "\nНевозможно очистить данные о всех
постояльцах, так как есть зарегистрированные!\n";
    }
    system("pause");
    break;
case '5':
    searchPassport(table);
    system("pause");
    break;
case '6':
    searchFIO(table);
    system("pause");
    break;
case '0':
    return;
default:
    cout << "Некорректный ввод!\n";
    system("pause");
}
}
}

//-----действия с гостиничными номерами-----
-----//

//структура для работы с номерами гостиницы (реализация AVL-дерева)
struct HotelRoom {
    string number; //строка формата "ANNN", где А - буква, обозначающая
тип
    //номера (Л - люкс, П - полулюкс, О - одноместный, М -
многоместный);

```

```

//NNN - порядковый номер (цифры)
int count_of_seats; //количество мест
int count_of_rooms; //количество комнат
bool bathroom; //наличие санузла
string facilities; //оборудование
int height; //высота дерева
HotelRoom* left;
HotelRoom* right;
int free_seats;

```

```

HotelRoom(string n, int s, int r, bool b, string f) {
    number = n;
    count_of_seats = s;
    count_of_rooms = r;
    bathroom = b;
    facilities = f;
    left = right = 0;
    height = 1;
    if (n[0] == 'M') free_seats = s;
    else free_seats = 1;
};

```

```
};
```

//функция проверки корректности введённого номера

```

void checkNumberRoom(string& num) {
    bool f;
    cin >> num;
    while (true) {
        f = true;

```

```

        if (num.size() != 4) {
            f = false;
        }
        else {
            if (!(num[0] == 'J' || num[0] == 'I' || num[0] == 'O' || num[0] ==
'M')) f = false;

            for (int i = 1; i < 4; ++i) {
                if (num[i] < '0' || '9' < num[i]) {
                    f = false;
                    break;
                }
            }
        }
    }

    if (f) return;
    cin.clear();
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    cerr << "Введено некорректное значение! ";
    cout << "Повторите ввод: ";
    cin >> num;
}
}

```

//функция проверки корректности введённого целого числа

```

void checkInt(int& n) {
    cin >> n;
    while (!cin.good() || cin.get() != '\n') {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cerr << "Введено некорректное значение! ";
    }
}

```

```

        cout << "Повторите ввод: ";
        cin >> n;
    }
}

//функция проверки корректности наличия ванной в номере
void checkBathroom(char& b) {
    cin >> b;
    while (!cin.good() || !(b == 'y' || b == 'n') || cin.get() != '\n') {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cerr << "Введено некорректное значение! ";
        cout << "Повторите ввод: ";
        cin >> b;
    }
}

```

```

//функция вычисления высоты AVL-дерева
int height(HotelRoom* room) {
    if (room != NULL) return room->height;
    else return 0;
}

```

```

//функция вычисления баланс-фактора AVL-дерева
int getBalance(HotelRoom* room) {
    if (room != NULL) return height(room->right) - height(room->left);
    else return 0;
}

```

```

//функция обновления высоты AVL-дерева

```



```

void updateHeight(HotelRoom* room) {
    if (room != NULL) room->height = 1 + max(height(room->left),
height(room->right));
}

```

//функция правого поворота AVL-дерева

```

HotelRoom* rotateRight(HotelRoom* room) {
    HotelRoom* room_new = room->left;
    room->left = room_new->right;
    room_new->right = room;
    updateHeight(room);
    updateHeight(room_new);
    return room_new;
}

```

//функция левого поворота AVL-дерева

```

HotelRoom* rotateLeft(HotelRoom* room) {
    HotelRoom* room_new = room->right;
    room->right = room_new->left;
    room_new->left = room;
    updateHeight(room);
    updateHeight(room_new);
    return room_new;
}

```

//функция балансировки AVL-дерева

```

HotelRoom* balance(HotelRoom* root)
{
    updateHeight(root);
    if (getBalance(root) == 2)

```

```

    {
        if (getBalance(root->right) < 0)
            root->right = rotateRight(root->right);
        return rotateLeft(root);
    }
    if (getBalance(root) == -2)
    {
        if (getBalance(root->left) > 0)
            root->left = rotateLeft(root->left);
        return rotateRight(root);
    }
    return root; // балансировка не нужна
}

//функция добавления нового элемента в AVL-дерево
HotelRoom* insert(HotelRoom* root, HotelRoom* room) {
    if (!root) return room;
    if (room->number < root->number) {
        root->left = insert(root->left, room);
    }
    else {
        root->right = insert(root->right, room);
    }
    return balance(root);
}

```

```

//функция проверки наличия номера в текущей базе
bool inHotelRoom(HotelRoom* root, string number) {
    if (!root) return false;

```

```

    if (number == root->number) {
        return true;
    }
    else if (number < root->number) {
        return inHotelRoom(root->left, number);
    }
    else {
        return inHotelRoom(root->right, number);
    }
}

```

//функция поиска минимального левого значения в поддереве

```

HotelRoom* findMin(HotelRoom* root) {
    if (root->left) return findMin(root->left);
    else return root;
}

```

//функция поиска минимального правого значения в поддереве

```

HotelRoom* removeMin(HotelRoom* root) {
    if (root->left == 0) {
        return root->right;
    }
    root->left = removeMin(root->left);
    return balance(root);
}

```

//функция удаления элемента из AVL-дерева

```

HotelRoom* remove(HotelRoom* root, string number) {
    if (!root) {
        return nullptr;
    }
}

```

```

    }
    if (number < root->number) {
        root->left = remove(root->left, number);
    }
    else if (number > root->number) {
        root->right = remove(root->right, number);
    }
    else {
        HotelRoom* l = root->left;
        HotelRoom* r = root->right;

        if (!r) {
            root->left = nullptr;
            delete root;
            return l;
        }

        HotelRoom* mini = findMin(r);
        mini->right = removeMin(r);
        mini->left = l;
        if (r != mini) {
            delete r;
        }

        root = mini;
    }
    return balance(root);
}

```

//функция очищения данных о всех номерах

```

void clearHotelRoom(HotelRoom* root) {
    if (root != nullptr) {
        clearHotelRoom(root->left);
        clearHotelRoom(root->right);
        delete root;
    }
}

```

//функция вывода номеров в порядке прямого обхода AVL-дерева

```

void printHotelRoom(HotelRoom* root) {
    if (!root) return;

    cout << root->number << '\t' << root->count_of_seats << "\t\t" << root-
>count_of_rooms << "\t\t\t" << (root->bathroom ? "true" : "false") << '\t' << root-
>facilities << '\n';

    if (root->left) printHotelRoom(root->left);
    if (root->right) printHotelRoom(root->right);
}

```

//функция поиска по номеру номера в AVL-дерева

```

HotelRoom* searchNumber(HotelRoom* root, string number) {
    if (number == root->number) {
        HotelRoom* result = new HotelRoom(root->number, root-
>count_of_seats, root->count_of_rooms, root->bathroom, root->facilities);
        return result;
    }
    else if (number < root->number) {
        return searchNumber(root->left, number);
    }
}

```

```

    }
    else {
        return searchNumber(root->right, number);
    }
}

//функция доавления нового элемента в базу номеров
void addRoom(HotelRoom*& rooms) {
    string number, facilities;
    int count_of_seats, count_of_rooms;
    char b;
    bool bathroom;
    cout << "Заполните данные о номере: \n";
    cout << "Номер (формат \"ANNN\"): ";
    checkNumberRoom(number);
    if (inHotelRoom(rooms, number)) {
        cout << "Этот номер уже есть в базе!\n";
        return;
    }
    cout << "Количество мест: ";
    checkInt(count_of_seats);
    cout << "Количество комнат: ";
    checkInt(count_of_rooms);
    cout << "Наличие санузла (y/n): ";
    checkBathroom(b);
    if (b == 'y') bathroom = true;
    else bathroom = false;
    cout << "Оборудование: ";
    //cin.ignore();
    getline(cin, facilities);

```

```

        HotelRoom* room = new HotelRoom(number, count_of_seats,
count_of_rooms, bathroom, facilities);
        rooms = insert(rooms, room);
    }

//функция удаления элемента из базы элементов
void delRoom(HotelRoom*& rooms, TrafficData*& data) {
    string number;
    cout << "Введите номер, который нужно удалить: ";
    checkNumberRoom(number);

    if (!inHotelRoom(rooms, number)) {
        cout << "Такого номера нет в базе!\n";
        return;
    }

    if (inTrafficData(data, number)) {
        cout << "\nУдаление невозможно! Сначала надо зарегистрировать
выселение постояльца!\n";
        return;
    }

    rooms = remove(rooms, number);
    cout << "Удаление выполнено!\n";
}

//функция вывода информации обо всех номерах
void printAllRooms(HotelRoom* rooms) {

```

```

    if (!rooms) {
        cout << "В базе нет номеров!\n";
        return;
    }

    cout << "Номер" << '\t' << "Количество мест" << '\t' << "Количество
комнат" << '\t' << "Санузел" << '\t' << "Оборудование\n";

    printHotelRoom(rooms);
    //printTree(rooms, rooms->height);
}

//функция поиска по номеру номера
void searchByNumber(HotelRoom* rooms) {
    string number;
    cout << "Введите номер, информацию о котором хотите получить: ";
    checkNumberRoom(number);

    HotelRoom* r = searchNumber(rooms, number);
    if (!inHotelRoom(rooms, number)) cout << "Такого номера нет в базе!\n";
    else {
        cout << "Номер" << '\t' << "Количество мест" << '\t' <<
"Количество комнат" << '\t' << "Санузел" << '\t' << "Оборудование\n";
        printHotelRoom(r);
    }
}

//функция реализующая алгоритм поиска Бойера-Мура
int BMsearch(string str, string substr) {
    int length_sub = substr.length();

```



```

int length_str = str.length();
if (length_str == 0 || length_sub == 0 || length_str < length_sub) {
    return -1; // Если строки пустые или подстрока длиннее строки
}

int arr[256];
for (int i = 0; i < 256; i++) {
    arr[i] = length_sub;
}

for (int i = 0; i < length_sub - 1; i++) { // Последний символ не
учитываем
    arr[(unsigned char)substr[i]] = length_sub - i - 1;
}

int position = length_sub - 1;
while (position < length_str) {
    int j = length_sub - 1;
    while (j >= 0 && substr[j] == str[position - length_sub + 1 + j]) {
        j--;
    }
    if (j < 0) { // Совпадение найдено
        return position - length_sub + 1; // Возвращаем индекс начала
совпадения
    }

    position += arr[(unsigned char)str[position]]; // Сдвигаем по
таблице
}

return -1; // Совпадение не найдено

```

```
}
```

```
//функция поиска оборудования в строке по AVL дереву
```

```
bool searchFacilityInHotelRooms(HotelRoom* root, const string& facility,
```

```
vector<HotelRoom*>& results) {
```

```
    if (root == nullptr) return false;
```

```
    bool found = false;
```

```
    // Поиск в левом поддереве
```

```
    found |= searchFacilityInHotelRooms(root->left, facility, results);
```

```
    // Поиск оборудования в текущем узле
```

```
    if (BMsearch(root->facilities, facility) != -1) {
```

```
        results.push_back(new HotelRoom(*root));
```

```
        found = true; // найдено совпадение
```

```
    }
```

```
    // Поиск в правом поддереве
```

```
    found |= searchFacilityInHotelRooms(root->right, facility, results);
```

```
    return found;
```

```
}
```

```
//функция поиска по оборудованию
```

```
void searchByEquipment(HotelRoom* rooms) {
```

```
    string equipment;
```

```
    cout << "Введите название оборудования: ";
```

```
    getline(cin, equipment);
```

```
    vector<HotelRoom*> results;
```

```

        if (searchFacilityInHotelRooms(rooms, equipment, results)) {
            cout << "Номер" << '\t' << "Количество мест" << '\t' <<
"Количество комнат" << '\t' << "Санузел" << '\t' << "Оборудование\n";
            for (int i = 0; i < results.size(); ++i) {
                cout << results[i]->number << '\t' << results[i]->count_of_seats
<< "\t\t" << results[i]->count_of_rooms << "\t\t\t" << (results[i]->bathroom ?
"true" : "false") << '\t' << results[i]->facilities << '\n';
            }
        }
        else {
            cout << "Нет номеров с таким оборудованием!\n";
        }
    }
}

```

//функция вывода меню раюоты с номерами

```

void printRoomMenu() {
    system("cls");
    cout << "Выберите действие: \n";
    cout << "+----+-----+\n";
    cout << "| 1. | Добавление нового номера      |\n";
    cout << "+----+-----+\n";
    cout << "| 2. | Удаление сведений о номере     |\n";
    cout << "+----+-----+\n";
    cout << "| 3. | Просмотр всех номеров          |\n";
    cout << "+----+-----+\n";
    cout << "| 4. | Очистка данных о номерах       |\n";
    cout << "+----+-----+\n";
    cout << "| 5. | Поиск по номеру                |\n";
    cout << "+----+-----+\n";
    cout << "| 6. | Поиск по оборудованию          |\n";
}

```

```

    cout << "+---+-----+\n";
    cout << "| 0. | Вернуться в главное меню      |\n";
    cout << "+---+-----+\n";
}

//функция для работы с номерами
void roomMenu(HotelRoom*& rooms, TrafficData*& data) {
    char menu_item;
    while (true) {
        printRoomMenu();
        cout << "Ваш выбор: ";
        cin >> menu_item;
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        system("cls");
        switch (menu_item) {
            case '1':
                addRoom(rooms);
                system("pause");
                break;
            case '2':
                delRoom(rooms, data);
                system("pause");
                break;
            case '3':
                printAllRooms(rooms);
                system("pause");
                break;
            case '4':
                if (data == nullptr) {
                    clearHotelRoom(rooms);

```

```

        cout << "Данные о постояльцах очищены!\n";
    }
    else {
        cout << "\nНевозможно очистить данные о всех
номерах, так как есть зарегистрированные!\n";
    }
    system("pause");
    break;
case '5':
    searchByNumber(rooms);
    system("pause");
    break;
case '6':
    searchByEquipment(rooms);
    system("pause");
    break;
case '0':
    return;
default:
    cout << "Некорректный ввод!\n";
    system("pause");
}
system("cls");
}
}

```

```

//-----действия с выселением и вселением-----
-----//

```

```

//функция проверки корректности введённой даты
void checkDate(string& date) {

```

```

bool f;
cin >> date;
while (true) {
    f = true;
    if (date.size() != 10) {
        f = false;
    }
    else {
        if (!(date[2] == '.' && date[5] == '.')) {
            f = false;
        }
        if (date[0] < '0' || '9' < date[0] || date[1] < '0' || '9' < date[1] ||
            date[3] < '0' || '9' < date[3] || date[4] < '0' || '9' < date[4] ||
            date[6] < '0' || '9' < date[6] || date[7] < '0' || '9' < date[7] ||
            date[8] < '0' || '9' < date[8] || date[9] < '0' || '9' < date[9]) {
            f = false;
        }
    }
    if (f) return;
    cin.clear();
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    cerr << "Введено некорректное значение! ";
    cout << "Повторите ввод: ";
    cin >> date;
}
}

```

//функция добавления нового вселения

```

void add(TrafficData** head, TrafficData* p) {
    TrafficData* cur = *head;

```

```

    if (cur == nullptr) {
        *head = p;
        p->next = p;
    }
    else {
        while (cur->next != *head)
            cur = cur->next;
        cur->next = p;
        p->next = *head;
    }
}

```

//функция удаления вселения (выселение)

```

void del(TrafficData** head, string passport, string number) {
    TrafficData* prev = nullptr;
    TrafficData* cur = *head;

    if (cur == nullptr) {
        // Список пустой, удалять нечего.
        return;
    }

    // Ищем удаляемый узел и его предыдущий элемент.
    do {
        if (cur->passport == passport && cur->number == number) {
            // Нашли удаляемый узел.
            break;
        }
        prev = cur;
    }
}

```

```

        cur = cur->next;
    } while (cur != *head);

    if (cur->passport != passport || cur->number != number) {
        // Удаляемый элемент не найден.
        return;
    }

    if (prev == nullptr) {
        // Удаляем единственный элемент
        delete* head;
        *head = nullptr;
    }
    else if (cur->next == cur) {
        // Удаляем последний элемент в циклическом списке
        prev->next = cur;
        delete cur;
        *head = nullptr;
    }
    else {
        // Удаляем узел из середины или начала
        prev->next = cur->next;
        if (cur == *head) {
            *head = cur->next; // Меняем head, если удаляем первый
элемент
        }
        delete cur;
    }
}

```


//функция проверки наличия зарегистрированного вселения

```
bool inTrafficData(TrafficData* head, string passport, string number) {  
    if (head == nullptr) {  
        return false;  
    }  
    TrafficData* cur = head;  
    do {  
        if (cur->passport == passport && cur->number == number) {  
            return true;  
        }  
        cur = cur->next;  
    } while (cur != head);  
    return false;  
}
```

//функция сортировки вставками

```
void insertSorted(TrafficData** headRef, TrafficData* newNode) {  
    // Специальный случай для головного узла  
    if (*headRef == nullptr || (*headRef)->passport >= newNode->passport) {  
        newNode->next = *headRef;  
        *headRef = newNode;  
    }  
    else {  
        // Найти узел, после которого необходимо осуществить вставку  
        TrafficData* current = *headRef;  
        while (current->next != nullptr && current->next->passport <  
newNode->passport) {  
            current = current->next;  
        }  
        newNode->next = current->next;
```

```

        current->next = newNode;
    }
}

//вспомогательная функция сортировки вставками
void insertionSort(TrafficData** headRef) {
    // Инициализируем отсортированный список
    TrafficData* sorted = nullptr;

    // Перебираем текущий список и вставляем каждый его узел в
    отсортированный список
    TrafficData* current = *headRef;
    while (current != nullptr) {
        // Запоминаем следующий для текущего, потому что мы меняем
        его next
        TrafficData* next = current->next;

        // Вставляем текущий в отсортированный список
        insertSorted(&sorted, current);

        // Перемещаемся на следующий узел
        current = next;
    }

    // Изменяем headRef, чтобы он указывал на отсортированный список
    *headRef = sorted;
}

//функция для очищения данных о зарегистрированных заселениях
void clearTrafficData(TrafficData*& head) {

```

```

while (head != nullptr) {
    TrafficData* temp = head; // Сохраняем текущий элемент.
    head = head->next; // Перемещаемся к следующему элементу.
    delete temp; // Удаляем текущий элемент.
}
}

//функция регистрации вселения
void registrationCheckIn(TrafficData*& data, hashMap* table, HotelRoom*
rooms) {
    cout << "Введите следующие данные для регистрации вселения: \n";
    string passport, number, checkin, checkout;
    cout << "Номер паспорта (строка формата NNNN-NNNNNN, где N -
цифры): ";
    checkPassport(passport);
    if (!table->inHashTable(passport)) {
        cout << "\nЭтот постоялец не живёт в гостинице!\n";
        return;
    }
    cout << "Номер гостиничного номера (строка формата \"ANNN\"): ";
    checkNumberRoom(number);
    if (!inHotelRoom(rooms, number)) {
        cout << "\nЭтого номера нет в базе номеров!\n";
        return;
    }
    if (inTrafficData(data, passport, number)) {
        cout << "\nЭтот постоялец уже зарегистрирован!\n";
        return;
    }
    if (rooms->count_of_seats == 0) {

```

```

        cout << "\nВ этой комнате нет свободных мест!\n";
        return;
    }
    cout << "Дата заселения: ";
    checkDate(checkin);
    cout << "Дата выселения: ";
    checkDate(checkout);

    bool f = true;
    int d1 = checkin[0] * 10 + checkin[1], d2 = checkout[0] * 10 + checkout[1];
    int m1 = checkin[3] * 10 + checkin[4], m2 = checkout[3] * 10 +
checkout[4];
    int y1 = checkin[6] * 1000 + checkin[7] * 100 + checkin[8] * 10 +
checkin[9],
        y2 = checkout[6] * 1000 + checkout[7] * 100 + checkout[8] * 10 +
checkout[9];
    if (y1 > y2) f = false;
    else if (y2 == y1 && m1 > m2) f = false;
    else if (y2 == y1 && m2 == m1 && d1 > d2) f = false;

    if (!f) {
        cout << "\nДата выселения не может быть меньше даты
заселения!\n";
        return;
    }

    TrafficData* d = new TrafficData(passport, number, checkin, checkout);
    add(&data, d);
    insertionSort(&data);
    rooms->free_seats--;

```

```

        cout << "\nЗаселение загеристировано!\n";
    }

//функция регистрации выселения
void registrationCheckOut(TrafficData*& data, hashMap* table, HotelRoom*
rooms) {
    cout << "Введите следующие данные для регистрации выселения: \n";
    string passport, number, checkin, checkout;
    cout << "Номер паспорта (строка формата NNNN-NNNNNN, где N -
цифры): ";
    checkPassport(passport);
    if (!table->inHashTable(passport)) {
        cout << "\nЭтот постоялец не живёт в гостинице!\n";
        return;
    }
    cout << "Номер гостиничного номера (строка формата \"ANNN\"): ";
    checkNumberRoom(number);
    if (!inHotelRoom(rooms, number)) {
        cout << "\nЭтого номера нет в базе номеров!\n";
        return;
    }

    if (!inTrafficData(data, passport, number)) {
        cout << "\nТакой регистрации нет в базе!\n";
        return;
    }

    del(&data, passport, number);
    rooms->free_seats++;
    cout << "\nРегистрация выселения прошла!\n";
}

```

```
}
```

```
//функция вывода меню регистрации вселения и выселения
```

```
void printDataMenu() {  
    system("cls");  
    cout << "Выберите действие: \n";  
    cout << "+----+-----+\n";  
    cout << "| 1. | Регистрация вселения      |\n";  
    cout << "+----+-----+\n";  
    cout << "| 2. | Регистрация выселения     |\n";  
    cout << "+----+-----+\n";  
    cout << "| 0. | Вернуться в главное меню  |\n";  
    cout << "+----+-----+\n";  
}
```

```
//функция реализации работы меню регистрации вселения и выселения
```

```
void dataMenu(TrafficData*& data, hashMap* table, HotelRoom* rooms) {  
    char menu_item;  
    while (true) {  
        printDataMenu();  
        cout << "Ваш выбор: ";  
        cin >> menu_item;  
        cin.ignore(numeric_limits<streamsize>::max(), '\n');  
  
        system("cls");  
        switch (menu_item) {  
            case '1':  
                registrationCheckIn(data, table, rooms);  
                system("pause");  
                break;
```

```

        case '2':
            registrationCheckOut(data, table, rooms);
            system("pause");
            break;
        case '0':
            return;
        default:
            cout << "Некорректный ввод!\n";

            system("pause");
        }
        system("cls");
    }
}

```

//функция вывода главного меню

```

void printMainMenu() {
    cout << "+----+-----+\n";
    cout << "| 1. | Постоялец          |\n";
    cout << "+----+-----+\n";
    cout << "| 2. | Гостиничный номер    |\n";
    cout << "+----+-----+\n";
    cout << "| 3. | Выселение и поселение постояльцев |\n";
    cout << "+----+-----+\n";
    cout << "| 0. | Завершение работы    |\n";
    cout << "+----+-----+\n";
}

```

//функция реализации работы главного меню

```

void mainMenu(){

```

```

hashMap table;
HotelRoom* rooms = NULL;
TrafficData* data = NULL;
char menu_item;
while (true) {
    cout << "Выберите, с какими данными нужно осуществить
работу: \n";
    printMainMenu();
    cout << "Ваш выбор: ";
    cin >> menu_item;

    system("cls");

    switch (menu_item) {
    case '1':
        guestMenu(&table, data);
        break;
    case '2':
        roomMenu(rooms, data);
        break;
    case '3':
        dataMenu(data, &table, rooms);
        break;
    case '0':
        cout << "Работа программы завершена!\n";
        table.~hashMap();
        clearHotelRoom(rooms);
        clearTrafficData(data);

        return;
    }
}

```



```

        }
        system("cls");
    }

}

int main() {
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    setlocale(LC_ALL, "rus");

    mainMenu();

    return 0;
}

```