Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра электронных вычислительных машин
Дисциплина: Основы алгоритмизации и программирования

Отчёт
по учебной практике(ознакомительной)
на тему

Реализация сервиса бронирования различных мест

Студент М.В. Козлов

Руководитель И.В. Лукьянова

МИНСК 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Система бронирования мест - это программное обеспечение, которое упрощает процесс резервирования мест в различных контекстах, таких как авиарейсы, железнодорожные путешествия, кинотеатры и мероприятия. Система автоматизирует задачи, связанные с управлением доступными местами, обработкой бронирований и подтверждением резерваций, что позволяет значительно повысить эффективность и точность.

Для этого проекта я разработал и реализовал систему бронирования мест, которая позволяет пользователям легко и быстро резервировать места по выбранным ими критериям. Система включает в себя функционал для просмотра доступных мест, выбора предпочтительных мест, обработки платежей и отправки подтверждений пользователям.

Основные преимущества этой системы бронирования мест включают:

1. Удобство для пользователей: отправка ID для входа в завдение и манипуляциями со своей бронью прямо в мессенджер
2. Оптимизация управления местами: автоматизация процессов позволяет администраторам эффективно управлять доступными и забронированными местами.
3. Снижение вероятности ошибок: система минимизирует человеческий фактор, что уменьшает вероятность ошибок при бронировании.
4. Ускорение процесса бронирования: пользователи могут быстро завершить процесс бронирования, что экономит их время и усилия.
5. Повышение удовлетворенности клиентов: оперативное подтверждение бронирований и возможность выбора предпочтительных мест улучшают общий опыт пользователей.

Эта система бронирования мест была разработана с учетом современных требований к безопасности и надежности данных, что делает её надежным инструментом для различных сценариев бронирования**.**

ОПИСАНИЕ ФУНКЦИЙ

questionData() - выводит меню выбора функционала и выполняет соответствующую функцию на основе выбора пользователя.

main() - основной цикл программы, который вызывает questionData() и предлагает пользователю продолжить или завершить программу.

addData() - очищает консоль и вызывает функцию UserGeneration() для добавления новой записи.

removeData() - очищает консоль и вызывает функцию removeUser() для удаления записи.

checkData() - очищает консоль и вызывает функцию startAdminPanel() для проверки записей (доступно только для администратора).

sortingData() - вызывает функцию sorting() для сортировки данных.

changeData() - очищает консоль и вызывает функцию changeUsers() для изменения записи.Взаимосвязь функций

nameInfo() Запрашивает у пользователя фамилию и сохраняет её в файл admin.txt.

timeInfo() Позволяет пользователю выбрать время и сохраняет выбранное время в файл admin.txt.

DataComplictation() Предлагает пользователю выбрать место из списка и записывает выбранное место в файл admin.txt.

delimeter() Добавляет разделитель (новую строку) в файл admin.txt

ReadID() Читает последний ID из файла admin.txt, запрашивает у пользователя его tgID и отправляет уникальный код через API.

UserGeneration() Выполняет последовательность действий для генерации пользователя, включая вызовы функций DataComplictation(), timeInfo(), nameInfo(), IdGenerate(), ReadID(), и delimeter().

IdGenerate() Генерирует уникальный ID на основе текущего времени и записывает его в файл admin.txt. [Блок-схема приведена в приложении Е]

GetNextId() Подсчитывает количество строк в файле admin.txt и возвращает следующий номер ID.

void removeUser() Эта функция отвечает за отмену бронирования для пользователя. После подтверждения пользователем своего намерения отменить бронь и ввода идентификационного номера (ID), функция открывает файл с данными администраторов и ищет строку, содержащую указанный ID. Если ID найден, функция удаляет соответствующую строку и сохраняет обновленные данные в новый файл, который затем заменяет старый. В случае успеха пользователю выводится сообщение об успешной отмене брони, иначе сообщается, что ID не найден.

void loginCheck(const char *login, const char *password) Эта функция проверяет логин и пароль администратора. Открывается файл с данными об администраторах, и для каждого логина в файле считается хэш введенного пароля с использованием алгоритма SHA-256. Если хэш совпадает с сохраненным хэшем, вход считается успешным, и вызывается функция GenerateReport(). В противном случае пользователю предлагается повторно ввести логин и пароль через функцию loginAdmin().

void loginAdmin() Эта функция отвечает за процесс входа администратора. Пользователь вводит свой логин и пароль, которые затем передаются в функцию loginCheck() для проверки.

void registrationInfo() Эта функция выполняет регистрацию нового администратора. Пользователь вводит логин, пароль и уникальный код. Логин и пароль должны быть длиннее 8 символов. Уникальный код проверяется на соответствие кодам в файле с кодами. Если введенный код действителен, данные передаются в функцию registrationAdmin() для завершения регистрации. В противном случае пользователю предлагается повторить попытку регистрации.

void registrationAdmin(const char *password, const char* login, const char* code) Эта функция завершает процесс регистрации администратора, сохраняя хэш пароля, логин и код в файл с данными администраторов. Для хэширования пароля используется алгоритм SHA-256[Блок-схема привидена в приложении Б]. После успешной записи данных в файл вызывается функция loginAdmin() для входа в систему.

void swapPlaceCount(PlaceCount *a, PlaceCount *b) и void swapTimeCount(TimeCount *a, TimeCount *b)

Эти вспомогательные функции меняют местами два элемента структур PlaceCount и TimeCount, соответственно. Используются в процессе сортировки.

int partitionPlace(PlaceCount arr[], int low, int high) и int partitionTime(TimeCount arr[], int low, int high)

Эти функции выполняют разбиение массива структур PlaceCount и TimeCount на две части для алгоритма быстрой сортировки. Разбиение происходит по значению count.

void quickSortPlace(PlaceCount arr[], int low, int high)[приложение B] и void quickSortTime(TimeCount arr[], int low, int high)

Эти функции реализуют алгоритм быстрой сортировки для массивов структур PlaceCount и TimeCount, соответственно. Сортировка производится по полю count в порядке убывания.

void changeUsers()

Эта функция позволяет пользователю изменить бронирование. Пользователю предлагается подтвердить свое намерение, выбрав один из вариантов: "Да" или "Нет". Если пользователь выбирает "Да", его просят ввести свой идентификационный номер (ID). Затем открывается файл с данными администраторов и создается временный файл для записи обновленных данных. Если введенный ID найден, строка с этим ID удаляется, а остальные строки записываются в временный файл. После этого оригинальный файл заменяется временным файлом. Если ID найден и удален, вызывается функция UserGeneration(), иначе пользователю сообщается, что ID не найден.

void changeConsoleTextColor(int color)

Эта функция изменяет цвет текста в консоли. Используется функция SetConsoleTextAttribute из Windows API для установки цвета текста. Параметр color определяет цвет, который будет установлен.

void ClearConsole()

Эта функция очищает консоль. Используется системная команда cls для очистки консоли.

void Exit()

Эта функция завершает выполнение программы. Используется функция exit(1) из стандартной библиотеки C для завершения программы.

void Music()

Эта функция воспроизводит аудиофайл в фоне консоли. Для воспроизведения звука используется функция PlaySound из библиотеки winmm.lib. Путь к файлу аудио указывается в параметре функции.

PlaceNode* createPlaceNode(int place)

Эта функция создает новый узел дерева для хранения информации о месте. Память для нового узла выделяется динамически, и инициализируются его поля. Узел возвращается как результат функции.

PlaceNode* insertPlaceNode(PlaceNode *root, int place)

Эта функция вставляет новый узел в дерево мест. Если корень дерева пуст, создается новый узел. В противном случае функция рекурсивно вставляет узел в левое или правое поддерево в зависимости от значения места. Если место уже существует, увеличивается счетчик бронирований для этого места.

TimeNode* createTimeNode(char *time)

Эта функция создает новый узел дерева для хранения информации о времени. Память для нового узла выделяется динамически, и инициализируются его поля. Узел возвращается как результат функции.

TimeNode* insertTimeNode(TimeNode *root, char *time)

Эта функция вставляет новый узел в дерево времени. Если корень дерева пуст, создается новый узел. В противном случае функция рекурсивно вставляет узел в левое или правое поддерево в зависимости от значения времени. Если время уже существует, увеличивается счетчик бронирований для этого времени.

void findMaxPlace(PlaceNode *root, int *maxCount, int *mostVisitedPlace)

Эта функция рекурсивно обходит дерево мест и находит место с наибольшим количеством бронирований. Результаты сохраняются в переменные maxCount и mostVisitedPlace.

void findMaxTime(TimeNode *root, int *maxCount, char *mostPopularTime)

Эта функция рекурсивно обходит дерево времени и находит время с наибольшим количеством бронирований. Результаты сохраняются в переменные maxCount и mostPopularTime.

void freePlaceTree(PlaceNode *root)

Эта функция освобождает память, выделенную для всех узлов дерева мест. Дерево обходится рекурсивно, и каждый узел освобождается.

void freeTimeTree(TimeNode *root)

Эта функция освобождает память, выделенную для всех узлов дерева времени. Дерево обходится рекурсивно, и каждый узел освобождается.

insertPlaceNode вставляет узел с указанным значением place в бинарное дерево поиска. Если узел с таким значением уже существует, увеличивает счетчик этого узла (count). Если узел не найден, создает новый узел с помощью функции createPlaceNode. [Блок-схема приведена в приложении Д]

Функции в этом коде тесно связаны между собой и работают вместе для достижения общей цели - сжатия и распаковки файлов.

main() является основной функцией, которая зацикливает работу программы и вызывает questionData() для выбора пользователем необходимого функционала.[Блок-схема приведена в приложении А]

questionData() обрабатывает выбор пользователя и вызывает соответствующую функцию (addData(), removeData(), changeData(), или checkData()) для выполнения указанного действия.

addData(), removeData(), changeData(), и checkData() выполняют конкретные задачи, такие как добавление, удаление, изменение или проверка записей соответственно.

void startAdminPanel() Эта функция запускает панель администратора, предлагая пользователю выбрать между входом в учетную запись и регистрацией нового администратора. В зависимости от выбора пользователя вызываются соответствующие функции: loginAdmin() или registrationInfo(). [Блок-схема приведена в приложении Г]

void sorting() Эта функция выполняет сортировку данных по количеству бронирований мест и времени. Читает данные из файла, подсчитывает количество бронирований для каждого места и времени, затем сортирует их по убыванию количества с использованием алгоритма быстрой сортировки. После сортировки данные выводятся на экран в виде таблицы.

void GenerateReport()

Эта функция генерирует отчет о самых популярных местах и времени бронирований. Открывается файл с данными администраторов, и данные считываются построчно. Для каждого места и времени создаются или обновляются соответствующие узлы дерева. После обработки всех данных, функция находит самое популярное место и время, выводит их в виде таблицы и предлагает пользователю отсортировать данные по популярности. Если пользователь выбирает сортировку, вызывается функция sortingData().

Общая логика программы заключается в следующем:

1. Основная программа запускается с отображения меню опций, из которых пользователь может выбрать. - Пользователь может выбрать один из следующих вариантов:

1: Добавить новую запись

2: Удалить запись

3: Изменить запись

4: Проверить запись (доступно только для Админа)

На основании ввода пользователя программа вызывает соответствующий подпрограмму для обработки выбранной операции.

2. Добавление новых пользователей начинается с addData, где вызывается UserGenerarion, который собирает все необходимые для записи в файл данные.

3. Процесс удаления и изменения практически идентичен и основан на создании временного файла, перезаписи всех нужных элементов, удаления исходного файла и переименования временного в исходный.

4.Процесс проверки информации начинается с проверки на уперпользователя, которая реализована через ввод логина и пароля, где пароль хэшируется алгоритмом шифрования sha256, после успешного входа через динамическую структуру данных бинарное дерево, находится самое посещаемое место и самое популярное время, дальше предлагается реализовать сортировку и вывод полной таблицы через алгоритм сортировки quickSort.

ЗАКЛЮЧЕНИЕ

Разработав это приложение, я приобрел ценный опыт реализации алгоритмов, ввода-вывода файлов, работы с API на C, опыт работы с динамическими структурами данных и передовых методов разработки программного обеспечения. [Исходный код в приложении Ё].

ПРИЛОЖЕНИЕ А

Блок схема алгоритма функции main().

ПРИЛОЖЕНИЕ Б

Блок схема алгоритма хэширования sha256.

ПРИЛОЖЕНИЕ В

Блок схема алгоритма функции quickSortPlace().

ПРИЛОЖЕНИЕ Г

Блок схема алгоритма функции startAdminPanel ().

ПРИЛОЖЕНИЕ Д

Блок схема алгоритма функции insertPlaceNode ().

ПРИЛОЖЕНИЕ Е

Блок схема алгоритма функции IdGenerate().

ПРИЛОЖЕНИЕ Ё

Исходный код приложения