**PARKING MANAGEMENT SYSTEM**

**A PROJECT REPORT**

**Submitted by :** - vishwas

**Date :-**30-8-2024

**TABLE OF CONTENT**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **TOPIC** | **PAGE NO** |
| 1 | Introduction | **3-4** |
| 2 | Objective | **5-6** |
| 3 | Tools/Environment | **9** |
| 4 | Program code | **10-17** |
| 5 | Input and output screen | **18-21** |
| 6 | Limitations of the project | **22-23** |
| 7 | Future applications of this project | **24-25** |
| 8 | biblography | **26** |

**INTRODUCTION**

The **Simplified Parking Management System** is a basic console-based application developed in C++ to manage a small parking lot of up to 50 vehicles. This system provides essential functionalities that allow users to park vehicles, view parking slot availability, calculate charges based on parking duration, and remove vehicles after payment. The goal of this project is to demonstrate a fundamental understanding of object-oriented programming concepts and basic file handling in C++, making it ideal for beginners who are learning to develop simple management systems.

****

**Key Features:**

1. **Parking Slot Management**:
   * The system keeps track of 50 parking slots, indicating whether each slot is available or occupied.
2. **Vehicle Parking**:
   * Users can park a vehicle by entering details such as the slot number, license plate, vehicle type, and estimated parking duration. The system checks if the selected slot is available and stores the vehicle's details if the slot is free.
3. **Vehicle Removal and Payment**:
   * Users can remove a parked vehicle from a slot by entering the slot number and making the required payment based on the parking duration. The system ensures payment is sufficient before removing the vehicle.
4. **Charge Calculation**:
   * The system calculates parking charges based on a fixed hourly rate. This simple approach ensures easy understanding and modification for different scenarios.
5. **File Handling**:
   * To ensure data persistence, the system uses basic file handling to save and load parking data (license plates, vehicle types, and parking charges) to and from a text file. This allows the system to retain information even after a restart, providing continuity in parking management.
6. **User-Friendly Interface**:
   * A menu-driven interface guides the user through various options, making the system straightforward to use.

**OBJECTIVE**

1. **Understand Object-Oriented Programming (OOP) Concepts:**

Develop a clear understanding of how to use classes and objects in C++.

Implement encapsulation by defining a class (ParkingSystem) to manage parking slots and vehicle details.

1. **Learn Basic File Handling:**

Gain experience in reading from and writing to files in C++.

Use file handling to save parking data persistently, allowing the system to retain information between sessions.

1. **Create a User-Friendly Console Interface:**

Design a simple, intuitive console-based menu that guides users through various options such as parking a vehicle, removing a vehicle, and viewing available slots.

Ensure the interface is straightforward and requires minimal input from the user, making the application accessible to those with basic computer skills.

1. **Implement Basic Parking Lot Management Features:**

Allow users to park vehicles by selecting an available slot and entering vehicle details.

Enable users to remove vehicles and calculate parking charges based on the duration of the stay.

Provide real-time updates on the status of parking slots, indicating whether they are occupied or available.

1. **Apply Simple Algorithms for Data Management:**

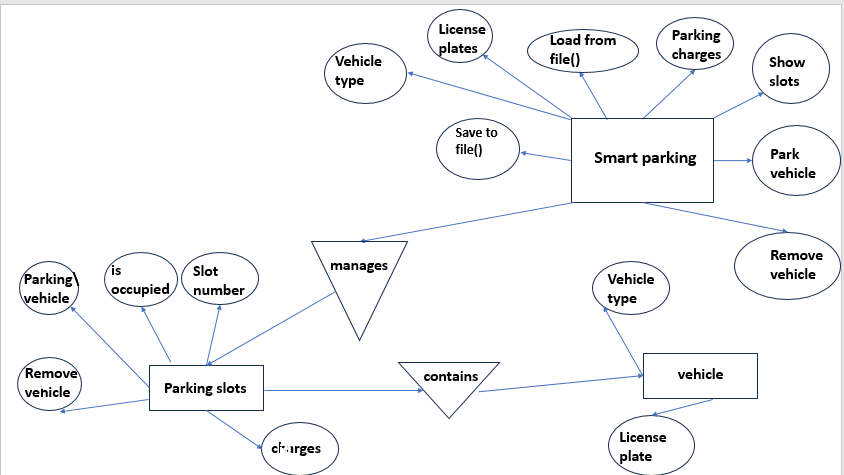
Use arrays to store vehicle information, demonstrating basic data structure management.

Implement simple logic for checking slot availability, calculating charges, and updating the status of parking slots.

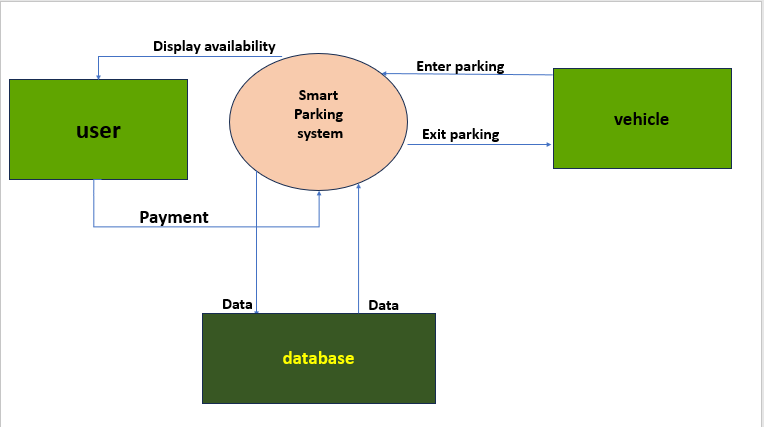
1. **Encourage Problem-Solving and Logical Thinking:**

Foster the development of problem-solving skills by requiring users to think logically about how to manage the parking slots and handle various scenarios such as full occupancy or insufficient payment.

**ER DIAGRAM**

****

**DATA FLOW DIAGRAM**



**TOOLS AND ENVIORNMENT**

HARDWARE REQUIREMENTS

Processor: Minimum Pentium IV 2.4 GHZ

RAM: At Least 100 MB

Disk Space: At Least 500 MB

SOFTWARE REQUIREMENTS

Operating System: Windows,IOS,LINUX,Etc.

Code Compiler : Visual Code Studio / Dev C++/ Turbo C++/Etc.

**PROGRAM CODE**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class ParkingSystem {

private:

string licensePlates[50]; // Stores the license plates of parked vehicles

string vehicleTypes[50]; // Stores the types of vehicles parked

double parkingCharges[50]; // Stores the parking charges for each slot

public:

// Constructor to initialize all parking slots to empty

ParkingSystem() {

for (int i = 0; i < 50; i++) {

licensePlates[i] = ""; // Empty string indicates the slot is available

vehicleTypes[i] = ""; // Empty string indicates no vehicle type is set

parkingCharges[i] = 0.0; // Zero charge means no vehicle is parked

}

}

// Function to display all parking slots and their status (available or occupied)

void showSlots() {

cout << "Parking Slots Status:\n";

for (int i = 0; i < 50; i++) {

if (licensePlates[i] == "") { // Check if the slot is available

cout << "Slot " << (i + 1) << ": Available\n";

} else {

cout << "Slot " << (i + 1) << ": Occupied by " << licensePlates[i] << "\n";

}

}

}

// Function to park a vehicle in a slot

void parkVehicle() {

int slotNumber;

string plate, type;

double hours;

cout << "Enter slot number to park the vehicle: ";

cin >> slotNumber;

// Check if the slot number is valid and available

if (slotNumber < 1 || slotNumber > 50 || licensePlates[slotNumber - 1] != "") {

cout << "Invalid slot number or slot already occupied!\n";

return;

}

// Get vehicle details from the user

cout << "Enter vehicle license plate: ";

cin >> plate;

cout << "Enter vehicle type: ";

cin >> type;

cout << "Enter number of hours to park: ";

cin >> hours;

// Store vehicle details and calculate parking charge

licensePlates[slotNumber - 1] = plate;

vehicleTypes[slotNumber - 1] = type;

parkingCharges[slotNumber - 1] = calculateCharge(hours);

cout << "Vehicle parked in slot " << slotNumber << ".\n";

cout << "Charge: Rs" << parkingCharges[slotNumber - 1] << "\n";

saveData(); // Save the updated parking data to the file

}

// Function to remove a vehicle from a slot

void removeVehicle() {

int slotNumber;

double payment;

cout << "Enter slot number to remove the vehicle from: ";

cin >> slotNumber;

// Check if the slot number is valid and occupied

if (slotNumber < 1 || slotNumber > 50 || licensePlates[slotNumber - 1] == "") {

cout << "Invalid slot number or slot is empty!\n";

return;

}

cout << "Parking charge for slot " << slotNumber << ": Rs" << parkingCharges[slotNumber - 1] << "\n";

// Get payment from the user and check if it's sufficient

cout << "Enter payment amount: ";

cin >> payment;

if (payment >= parkingCharges[slotNumber - 1]) {

cout << "Payment successful! Vehicle removed from slot " << slotNumber << ".\n";

// Clear the slot data

licensePlates[slotNumber - 1] = "";

vehicleTypes[slotNumber - 1] = "";

parkingCharges[slotNumber - 1] = 0.0;

} else {

cout << "Insufficient payment. Vehicle not removed.\n";

}

saveData(); // Save the updated parking data to the file

}

// Function to calculate parking charges based on hours

double calculateCharge(double hours) {

double ratePerHour = 10.0; // Simple fixed rate per hour

return hours \* ratePerHour;

}

// Function to save parking data to a file

void saveData() {

ofstream file("parking\_data.txt");

for (int i = 0; i < 50; i++) {

file << licensePlates[i] << endl;

file << vehicleTypes[i] << endl;

file << parkingCharges[i] << endl;

}

file.close();

}

// Function to load parking data from a file

void loadData() {

ifstream file("parking\_data.txt");

for (int i = 0; i < 50; i++) {

getline(file, licensePlates[i]);

getline(file, vehicleTypes[i]);

file >> parkingCharges[i];

file.ignore(); // Ignore the newline character after the charge

}

file.close();

}

};

int main() {

ParkingSystem parking;

parking.loadData(); // Load existing parking data from the file

int choice;

do {

cout << "\nSmart Parking System Menu:\n";

cout << "1. Show Parking Slots\n";

cout << "2. Park a Vehicle\n";

cout << "3. Remove a Vehicle\n";

cout << "4. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

parking.showSlots();

break;

case 2:

parking.parkVehicle();

break;

case 3:

parking.removeVehicle();

break;

case 4:

cout << "Exiting the system. Goodbye!\n";

parking.saveData(); // Save data before exiting

break;

default:

cout << "Invalid choice! Please try again.\n";

break;

}

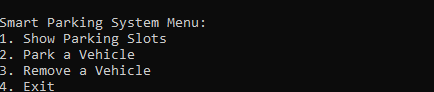
} while (choice != 4);

return 0;

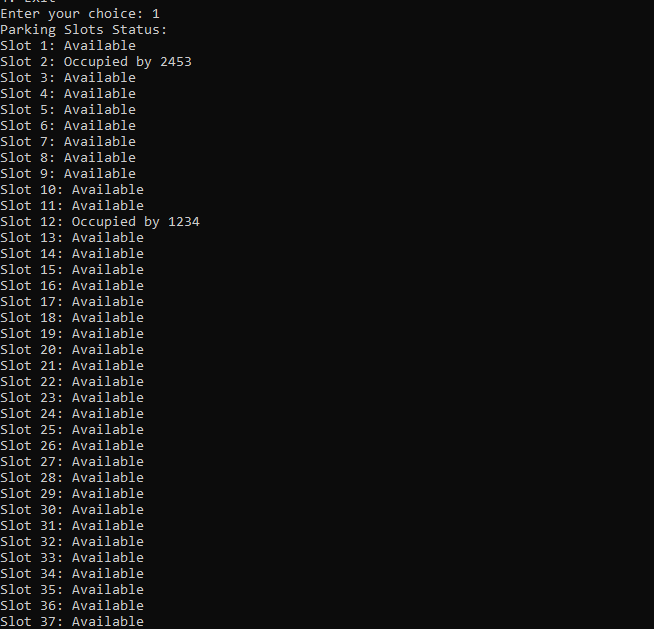
}

**INPUT/OUTPUT SCREEN**

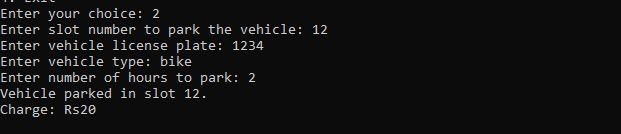
**1** Main menu



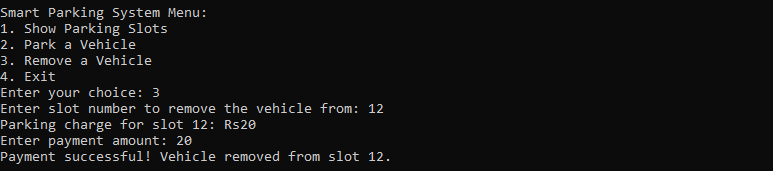
**2** show parking slots



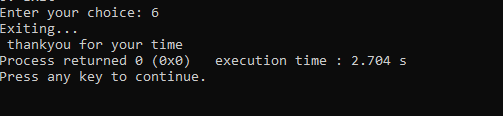
**3** park a vehicle



4 remove vehicle



5 exit



**Limitations of this project**

While the Simplified Parking Management System provides a basic framework for managing a small parking lot, it has several limitations due to its simplicity and the educational focus of the project. These limitations are inherent to its design and the constraints set to keep the system understandable for beginners.

1. **Fixed Parking Slot Limit:**
   * The system is designed to handle only 50 parking slots. It lacks scalability and cannot easily accommodate a parking lot with more slots without significant modifications to the code.
2. **Basic Data Structures:**
   * The use of fixed-size arrays for storing vehicle information means the system is not dynamic. This makes it less flexible and more challenging to manage when dealing with more complex or larger datasets.
3. **Simple File Handling:**
   * The file handling implementation is basic and does not handle errors robustly. For example, if the data file is missing or corrupted, the system simply starts with empty parking slots, which could lead to data loss.
4. **Lack of Concurrency Handling:**
   * The system does not support concurrent access, meaning it cannot handle multiple users interacting with the system at the same time. This limitation makes the system unsuitable for larger or more complex environments where multiple administrators might need access simultaneously.
5. **No Advanced Error Handling:**
   * Error handling is minimal, focusing primarily on preventing invalid slot selections or insufficient payments. It does not handle more complex scenarios such as partial data corruption, network issues (in case of future expansion), or unexpected user input.
6. **Limited Vehicle Information:**
   * The system stores only basic information about each vehicle (license plate, vehicle type, and parking charge). There is no capability to store additional details like owner information, vehicle color, or contact information, which could be useful in a real-world scenario.
7. **Simplistic Billing System:**
   * The billing system uses a flat hourly rate for calculating charges, without any consideration for different rates based on vehicle type, time of day, or membership discounts. This makes the system unsuitable for parking lots with more complex pricing structures.

**Future applications of this project**

The Simplified Parking Management System provides a foundational framework that can be expanded upon to create more advanced and versatile parking solutions. With enhancements and additional features, this system could be adapted for various real-world applications. Below are some potential future applications and improvements:

1. **Scalable Parking Management for Larger Facilities:**
   * Dynamic Slot Allocation: Expand the system to support dynamic slot management using data structures like vectors or linked lists, allowing it to handle an increased number of parking slots without a fixed limit.
   * Multiple Parking Zones: Add the ability to manage multiple zones or levels within a parking facility, making it suitable for larger parking lots or multi-level garages.
2. **Integration with Real-Time Sensor and IoT Devices:**
   * Automatic Slot Detection: Integrate the system with IoT devices and sensors that automatically detect when a vehicle enters or exits a parking slot, allowing for real-time updates without manual input.
   * Smart Parking Guidance: Implement features that provide real-time guidance to drivers, directing them to the nearest available parking spot using sensors and display boards.
3. **Enhanced User Experience with a Graphical User Interface (GUI):**
   * Web or Mobile Application: Develop a web-based or mobile app interface that allows users to view available parking slots, reserve slots, and pay parking fees, enhancing convenience and accessibility.
   * Interactive Kiosks: Deploy interactive kiosks at parking lot entrances for self-service parking management, reducing the need for on-site staff.
4. **Advanced Billing and Payment Systems:**
   * Flexible Billing Options: Introduce variable pricing models based on time of day, vehicle type, or user membership status, allowing for more sophisticated billing strategies.
   * Digital Payment Integration: Integrate with digital payment gateways and mobile payment solutions, enabling users to pay parking fees using credit cards, mobile wallets, or contactless payments.
5. **Enhanced Security and User Authentication:**
   * User Accounts and Authentication: Implement user accounts with authentication features, ensuring that only authorized personnel can access the system and manage parking slots.
   * License Plate Recognition (LPR): Integrate license plate recognition technology to automatically identify and authenticate vehicles entering and exiting the parking lot.

**BIBLOGRAPHY**

Books

Let Us C by Yashavant Kanetkar.

Let us C++ by Yashavant Kanetkar.

C in Depth by S.K Srivastava.

The C++ Programming Language By Bjarne Stroustrup.

Websites

[www.google.com](http://www.google.com)

[www.youtube.com](http://www.youtube.com)

[www.w3schools.com](http://www.w3schools.com)

[www.geeksforgeeks.com](http://www.geeksforgeeks.com)