****

**project name- SMART PARKING system**

**A PROJECT REPORT**

**Submitted By :- SURAJ KUMAR AGRAHARI**

**Date :- 10-08-2024**

**TABLE OF CONTENT**

|  |  |  |
| --- | --- | --- |
| **S.No** | **TOPIC** | **PAGE NO** |
| 1 | Introduction | 1-2 |
| 2 | Objective | 3-4 |
| 3 | Tools/Environment | 5 |
| 4 | Program code | 6-17 |
| 5 | Limitations of the Project | 18-19 |
| 6 | Future Applications of the Project | 20-21 |
| 7 | Bibliography |  |

**INTRODUCTION**

Welcome to the Smart Parking System!

In an era where efficient parking management is crucial for urban planning and everyday convenience, our Smart Parking System provides a streamlined and user-friendly solution for managing parking slots. This C++ program simulates a parking management system that allows users to park and retrieve vehicles, track parking charges, and maintain data persistence through file operations.

#### Key Features:

1. **Dynamic Slot Management**:
   * Users can view available parking slots and choose where to park their vehicles. The system visually represents slot availability, making it easy to see which slots are free.
2. **Vehicle Entry and Exit**:
   * The program handles vehicle entry by recording the license plate, vehicle type, and duration of parking. On exit, it calculates the total charges and processes the payment.
3. **Charge Calculation**:
   * The system calculates parking charges based on the duration of parking using a fixed rate per hour, ensuring transparent and consistent billing.
4. **Data Persistence**:
   * All parking data, including slot information and charges, is saved to a file. This ensures that data is preserved between program runs and can be retrieved when the system is restarted.
5. **User-Friendly Interface**:
   * The console-based interface provides clear prompts and messages to guide users through the process of parking and exiting. Input validation helps prevent errors and improves the overall user experience.

#### How It Works:

1. **Enter Vehicle**:
   * Choose an available slot, enter the vehicle's license plate, select the vehicle type, and specify the parking duration. The system calculates and displays the parking charges.
2. **Exit Vehicle**:
   * When exiting, specify the slot number, and the system calculates the total charge. Users can then provide payment, and the system will issue change if necessary.
3. **Display Parked Vehicles**:
   * View a list of currently parked vehicles along with their slot numbers and charges, providing an overview of the parking lot's current status.
4. **Data Management**:
   * The system reads from and writes to a file named parking\_data.txt, ensuring data is preserved across sessions.

This parking management system demonstrates fundamental concepts of object-oriented programming, file handling, and user interaction in C++. Whether you're managing a small parking lot or simply interested in understanding how such systems work, this program offers practical insights into effective parking management.

Top of Form

Bottom of Form

**OBJECTIVES**

The Smart Parking System program is designed to achieve several key objectives, focusing on both practical functionality and effective learning outcomes. Below are the primary objectives:

1. **Efficient Slot Management**:
   * **Objective**: Provide a clear and effective mechanism for managing parking slots, including viewing available slots and handling vehicle entries and exits.
   * **Outcome**: Users can easily identify free slots and park their vehicles accordingly, ensuring optimal use of available parking space.
2. **Accurate Charge Calculation**:
   * **Objective**: Implement a reliable method for calculating parking charges based on the duration of stay.
   * **Outcome**: The system calculates parking fees accurately using a predefined rate per hour, providing transparency and consistency in billing.
3. **User Input Validation**:
   * **Objective**: Ensure that user inputs are validated to prevent errors and ensure the integrity of the parking management process.
   * **Outcome**: The program handles invalid inputs gracefully, guiding users to correct their entries and preventing system errors.
4. **Data Persistence**:
   * **Objective**: Maintain data consistency across multiple sessions by saving and retrieving parking data to and from a file.
   * **Outcome**: The system persists parking data, including slot availability and charges, allowing for continued use without data loss between sessions.
5. **User-Friendly Interface**:
   * **Objective**: Design a clear and intuitive console-based interface for interacting with the parking system.
   * **Outcome**: Users can easily navigate through options, enter data, and view information without confusion, enhancing the overall user experience.
6. **Practical Application of Object-Oriented Programming**:
   * **Objective**: Demonstrate the use of object-oriented programming principles, such as encapsulation, data management, and method implementation, in a real-world scenario.
   * **Outcome**: The program showcases practical use cases of classes and methods, reinforcing fundamental concepts of object-oriented design.
7. **File Handling Proficiency**:
   * **Objective**: Implement file handling operations to save and load parking data, showcasing skills in managing persistent storage.
   * **Outcome**: The system effectively reads from and writes to a file, ensuring that parking data is accurately maintained across program executions.
8. **Error Handling and Robustness**:
   * **Objective**: Develop a robust system capable of handling various error scenarios, including invalid user inputs and file operation failures.
   * **Outcome**: The program includes error-handling mechanisms to manage and report issues, ensuring stability and reliability in different situations.

Top of Form

Bottom of Form

**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.

**library MANAGEMENT SYSTEM C++ CODE**

#include <iostream>

#include <string>

#include <iomanip>

#include <fstream>

#include <limits>

#include <cstdlib> // For system()

using namespace std;

const int MAX\_SLOTS = 50;

const int COLUMNS = 10;

const double RATE\_PER\_HOUR = 10.00;

class ParkingSystem {

private:

const string VEHICLE\_TYPES[3] = {"Car", "Motorbike", "Bus"};

string licensePlates[MAX\_SLOTS];

string vehicleTypes[MAX\_SLOTS];

double parkingCharges[MAX\_SLOTS];

bool isValidVehicleType(const string& type) {

for (const auto& vt : VEHICLE\_TYPES) {

if (type == vt) {

return true;

}

}

return false;

}

void clrscr() {

#ifdef \_WIN32

system("cls");

#else

system("clear");

#endif

}

public:

ParkingSystem() {

for (int i = 0; i < MAX\_SLOTS; ++i) {

licensePlates[i] = "";

vehicleTypes[i] = "";

parkingCharges[i] = 0.0;

}

}

void displayAvailableSlots() {

clrscr();

cout << "Available Slots:\n";

cout << "----------------------------\n";

for (int i = 0; i < MAX\_SLOTS; ++i) {

if (licensePlates[i].empty()) {

cout << setw(3) << i + 1 << " ";

} else {

cout << setw(3) << "X" << " ";

}

if ((i + 1) % COLUMNS == 0) {

cout << endl;

}

}

cout << "\n----------------------------\n";

}

void displayVehicleTypeOptions() {

clrscr();

cout << "Available Vehicle Types:\n";

for (int i = 0; i < 3; ++i) {

cout << i + 1 << ". " << VEHICLE\_TYPES[i] << endl;

}

}

void EnterVehicle() {

int slotNumber;

string plate, type;

double hours;

displayAvailableSlots();

cout << "Enter slot number to enter: ";

cin >> slotNumber;

// Input validation

if (cin.fail() || slotNumber < 1 || slotNumber > MAX\_SLOTS || !licensePlates[slotNumber - 1].empty()) {

cin.clear(); // Clear error flag

cin.ignore(numeric\_limits<streamsize>::max(), '\n'); // Discard invalid input

cout << "Invalid slot number!" << endl;

return;

}

cout << "Enter license plate: ";

cin >> plate;

displayVehicleTypeOptions();

cout << "Enter vehicle type (number): ";

int typeChoice;

cin >> typeChoice;

if (cin.fail() || typeChoice < 1 || typeChoice > 3) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Invalid vehicle type choice!" << endl;

return;

}

type = VEHICLE\_TYPES[typeChoice - 1];

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

cin >> hours;

if (cin.fail() || hours < 0) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Invalid input for hours!" << endl;

return;

}

licensePlates[slotNumber - 1] = plate;

vehicleTypes[slotNumber - 1] = type;

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

cout << "Vehicle parked in slot " << slotNumber << endl;

cout << "Charge for " << hours << " hours: Rs" << parkingCharges[slotNumber - 1] << endl;

SaveToFile();

}

void ExitVehicle() {

int slotNumber;

double payment;

clrscr();

cout << "Enter slot number to exit: ";

cin >> slotNumber;

// Input validation

if (cin.fail() || slotNumber < 1 || slotNumber > MAX\_SLOTS || licensePlates[slotNumber - 1].empty()) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Invalid slot number!" << endl;

return;

}

cout << "Your parking charge is: Rs" << parkingCharges[slotNumber - 1] << endl;

do {

cout << "Please enter payment amount: ";

cin >> payment;

if (cin.fail() || payment < 0) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Invalid payment amount!" << endl;

} else if (payment < parkingCharges[slotNumber - 1]) {

cout << "Insufficient amount.\n";

}

} while (payment < parkingCharges[slotNumber - 1]);

double change = payment - parkingCharges[slotNumber - 1];

cout << "Payment successful! Your change is Rs" << change << endl;

licensePlates[slotNumber - 1] = "";

vehicleTypes[slotNumber - 1] = "";

parkingCharges[slotNumber - 1] = 0.0;

cout << "Vehicle removed from slot " << slotNumber << endl;

SaveToFile();

}

void DisplayParkedVehicles() {

clrscr();

cout << "Parked Vehicles:\n";

int count = 0;

for (int i = 0; i < MAX\_SLOTS; ++i) {

if (!licensePlates[i].empty()) {

cout << "Slot " << i + 1 << ": " << licensePlates[i] << " (" << vehicleTypes[i] << "), Charge: Rs"

<< parkingCharges[i] << endl;

count++;

}

}

if (count == 0) {

cout << "No vehicles parked." << endl;

}

}

double calculateCharge(double hours) {

return hours \* RATE\_PER\_HOUR;

}

void SaveToFile() {

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

if (outfile.is\_open()) {

for (int i = 0; i < MAX\_SLOTS; ++i) {

outfile << licensePlates[i] << endl;

outfile << vehicleTypes[i] << endl;

outfile << parkingCharges[i] << endl;

}

outfile.close();

} else {

cout << "Failed to open data file for writing.\n";

}

}

void FetchFromFile() {

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

if (infile.is\_open()) {

for (int i = 0; i < MAX\_SLOTS; ++i) {

getline(infile, licensePlates[i]);

getline(infile, vehicleTypes[i]);

infile >> parkingCharges[i];

infile.ignore();

}

infile.close();

} else {

cout << "Failed to open data file. Starting with empty parking slots.\n";

}

}

};

int main() {

ParkingSystem sy;

sy.FetchFromFile();

int choice;

do {

cout << "\n--------------> Smart Parking System <-----------------\n";

cout << "1. Enter Parking\n";

cout << "2. Exit Parking\n";

cout << "3. Display Parked Vehicles\n";

cout << "4. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

// Input validation

if (cin.fail()) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Invalid choice! Try again.\n";

continue;

}

cout << endl;

switch (choice) {

case 1:

sy.EnterVehicle();

break;

case 2:

sy.ExitVehicle();

break;

case 3:

sy.DisplayParkedVehicles();

break;

case 4:

cout << "Exiting...\n";

sy.SaveToFile();

break;

default:

cout << "Invalid choice! Try again.\n";

}

} while (choice != 4);

return 0;

}

**Limitations of the Project**

While the Smart Parking System offers a range of useful features, there are certain limitations to be aware of:

1. **Fixed Number of Slots**:
   * **Limitation**: The system supports a fixed number of parking slots (50 slots), which may not scale well for larger parking facilities.
   * **Impact**: This limitation restricts the system's applicability to larger or more dynamic parking environments where the number of slots may vary.
2. **Simple Charge Calculation**:
   * **Limitation**: The charge calculation is based on a fixed rate per hour, with no differentiation based on vehicle type or parking duration beyond the hourly rate.
   * **Impact**: This simplicity may not reflect real-world scenarios where different vehicle types or longer parking durations could influence pricing.
3. **Basic File Handling**:
   * **Limitation**: Data is stored in a plain text file without any structured format or encryption, which may not be secure or easily manageable for larger datasets.
   * **Impact**: This could lead to data corruption or security issues if the file is improperly handled, and it may not be suitable for high-security environments.
4. **Limited Vehicle Type Options**:
   * **Limitation**: The system supports only three types of vehicles (Car, Motorbike, Bus) and does not allow for customization or expansion of vehicle types.
   * **Impact**: This restricts the flexibility of the system to accommodate a wider range of vehicle types that might be present in different parking facilities.
5. **No Real-Time Updates**:
   * **Limitation**: The system does not handle real-time updates or provide mechanisms for tracking vehicle movements or slot changes dynamically.
   * **Impact**: Users need to manually refresh and interact with the system for updates, which may not be efficient for high-traffic scenarios.
6. **Error Handling Limitations**:
   * **Limitation**: While the system includes basic error handling, it may not cover all potential edge cases, such as handling file read/write errors comprehensively.
   * **Impact**: In some scenarios, the system might not recover gracefully from unforeseen errors, leading to potential issues or data loss.
7. **No GUI or Advanced Features**:
   * **Limitation**: The program uses a console-based interface with no graphical user interface (GUI) or advanced features such as automated reporting or notifications.
   * **Impact**: This limits the user experience to a text-based interface, which may not be as engaging or intuitive as modern graphical interfaces.
8. **Limited Payment Options**:
   * **Limitation**: The system handles payment only through a simple input mechanism without supporting various payment methods or integrating with external payment systems.
   * **Impact**: This could be a limitation in real-world applications where diverse payment options are required for user convenience.
9. **No Multi-user Support**:
   * **Limitation**: The system is designed for single-user interaction and does not support simultaneous access by multiple users.
   * **Impact**: This limits the system's application in environments where multiple users need to access or manage the parking system concurrently.

**FUTURE APPLICATION OF THE PROJECT**

The Smart Parking System can be expanded and adapted for a variety of future applications, enhancing its functionality and integration with modern technologies. Here are some potential future applications:

1. **Dynamic Pricing Models**:
   * **Application**: Implement dynamic pricing based on factors such as demand, time of day, vehicle type, or special events.
   * **Impact**: This can optimize revenue and provide users with more flexible pricing options, reflecting real-time market conditions.
2. **Scalability for Larger Parking Facilities**:
   * **Application**: Extend the system to handle a larger number of parking slots and integrate with parking structures of varying sizes.
   * **Impact**: Adapt the system to manage extensive parking lots or multi-level garages, improving its applicability for urban and commercial environments.
3. **Integration with IoT Devices**:
   * **Application**: Integrate with Internet of Things (IoT) sensors and devices to monitor real-time slot availability, detect vehicle presence, and manage parking space efficiently.
   * **Impact**: Enhance the accuracy of slot availability information and automate parking management processes, improving user convenience and operational efficiency.
4. **Mobile and Web Applications**:
   * **Application**: Develop mobile and web applications for users to manage parking reservations, payments, and notifications from their smartphones or computers.
   * **Impact**: Provide users with a more accessible and convenient interface for interacting with the parking system, including features like real-time updates and reservations.
5. **Automated Payment and Ticketing Systems**:
   * **Application**: Incorporate automated payment kiosks, contactless payment options, and electronic ticketing systems.
   * **Impact**: Streamline the payment process, reduce queues, and enhance the overall user experience with modern payment technologies.
6. **Advanced Reporting and Analytics**:
   * **Application**: Implement advanced reporting tools and analytics to track parking usage patterns, revenue generation, and operational performance.
   * **Impact**: Provide valuable insights for parking facility managers to optimize operations, make data-driven decisions, and improve overall efficiency.
7. **Integration with Navigation Systems**:
   * **Application**: Integrate with GPS and navigation systems to guide drivers to available parking spaces based on real-time data.
   * **Impact**: Improve the user experience by providing directions to the nearest available slot, reducing time spent searching for parking.
8. **Support for Multiple User Roles**:
   * **Application**: Implement multi-user support with different access levels for administrators, staff, and regular users.
   * **Impact**: Allow for more sophisticated management and oversight of the parking system, including features like user permissions and administrative controls.
9. **Environmental and Energy Management**:
   * **Application**: Integrate with energy management systems to support electric vehicle (EV) charging stations and monitor energy usage.
   * **Impact**: Promote sustainability and accommodate the growing number of electric vehicles by providing charging infrastructure and managing energy consumption.
10. **Enhanced Security Features**:
    * **Application**: Incorporate security features such as surveillance cameras, license plate recognition, and access control mechanisms.
    * **Impact**: Increase the safety and security of the parking facility, reducing incidents of theft and vandalism.
11. **Localization and Multi-Language Support**:
    * **Application**: Provide localization and support for multiple languages to cater to diverse user bases in different regions.
    * **Impact**: Make the system more accessible and user-friendly for international and multilingual users.
12. **Integration with Smart City Initiatives**:
    * **Application**: Integrate with broader smart city initiatives to coordinate with other urban systems such as traffic management and public transportation.

**BIBLIOGRAPHY**

**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)

