EPEECT: Enhancing Parking-Lot Experience by Efficient Tidying
A Parking Management System

Santos, Adrian Walter D.
Quiming, Kadaphy James A.
Garcia, Dennis James T.
Omaña, Amiel Klarrence M.

Technological Institute of the Philippines
Quezon City

November 2025

Table of Contents

Introduction

Discovering that there are no more vacant spots left in a parking lot after minutes of trying to find one can be very frustrating that any driver can experience. What should be quick and easy often turns into a time-consuming and stressful process. Many minutes of a driver's valuable time are wasted driving around the parking lot searching for an available slot only to no avail. This can lead to unnecessary stress among drivers which should be easily alleviated with proper management and organization. In the survey conducted by Rappler "34% said that they would rather spend time with friends and family if they can reduce the time they spend on the road or lined up in parking areas (Mejia, 2017). Precious time that can be used on more productive and meaningful things are wasted because of this issue, "more than a third of respondents (37%) claimed that they spend an average of 20 minutes waiting for parking slots to free up" (Mejia, 2017) so this project aims to fix this problem by providing an efficient and organized system using the C++ language.

There are multiple factors that contribute to this issue, which all combine to create the issue of a disorganized and confusing parking lot. The lack of a UI leads drivers wandering the parking lot searching for an available slot without any form of guidance that could cost them minutes of their time. This could also result in congestion of the driveway leading to further problems. Another problem is the inefficient calculation of parking fees leading to a slow exit of drivers.

This project is done to create a solution to these problems by using the C++ programming language to create a parking management system. *Bajçinovci and Bajçinovci (2019)* argue that the lack of sustainable urban mobility planning has contributed to parking-related stress thus through project Enhancing Parking-Lot Experience by Efficient Tidying (EPEECT) a positive parking lot experience could be achieved hence preventing the stress of drivers resulting in the improvement of their overall quality of life.

The Project

This project is designed to significantly improve the efficiency of the parking lot by providing an organized system in order to avoid any delay. The project starts by displaying the number of cars inside the parking lot and displaying if there are any available spots left through this feature; drivers are informed in advance if the parking is full or not. Then proceeds to display four options "Vehicle Entry", "Vehicle Exit", "View Parking Logs & Invoice", and "Exit" for drivers to choose from.

When the driver chooses Vehicle Entry, the program will ask for the car's license plate number and entry time. It will then store these values and display the remaining parking slots left after accounting for the new entry. The program then updates the number of available slots left, minusing one to the total count.

When the driver chooses Vehicle Exit, the program will display a menu showing all the cars parked which then the program will ask the driver for the number corresponding to their car. The program would ask for the exit time in 24 hour format, then it would ask if the car was left overnight and if the driver had lost their parking card which would add 200 each to the total parking fee. The inputted entry time and exit time would be counted by the program in calculating the parking fee with a rate of 20 Pesos per hour for the first 3 hours; followed by 30 Pesos per hour when the time exceeds 3 hours. Finally the program would print a summary which includes the plate number, entry time, exit time, and total parking fee. Then it updates the number of available slots left, adding one count to the total count.

When the driver chooses View Parking Log & Invoices, the program displays the license plates of every car that has entered and exited the parking lot. It also includes the time each car has entry time and exit time of each car. Lastly it also displays the invoice of each car, showing the fees that each driver had to pay. It also includes the cars that have yet to exit the parking lot labeling them “exit time” as “still parked.

Lastly when the driver chooses Exit, the program ends leaving a thank you message.

Objectives

The goal of this project is to aid in creating a better parking experience and to remove any possible delay that may occur when a car enters and exits the parking lot.

1. To be able to show drivers if there are available spots left in the parking lot.
2. To be able to record whenever a car exits or enters the parking lot.
3. To be able to automatically calculate the parking fee when the car exits the parking lot.

Flowchart of the System

The following flowcharts (Figure 1 - 1.8) show the flow of the system guided by the main code and its functions.

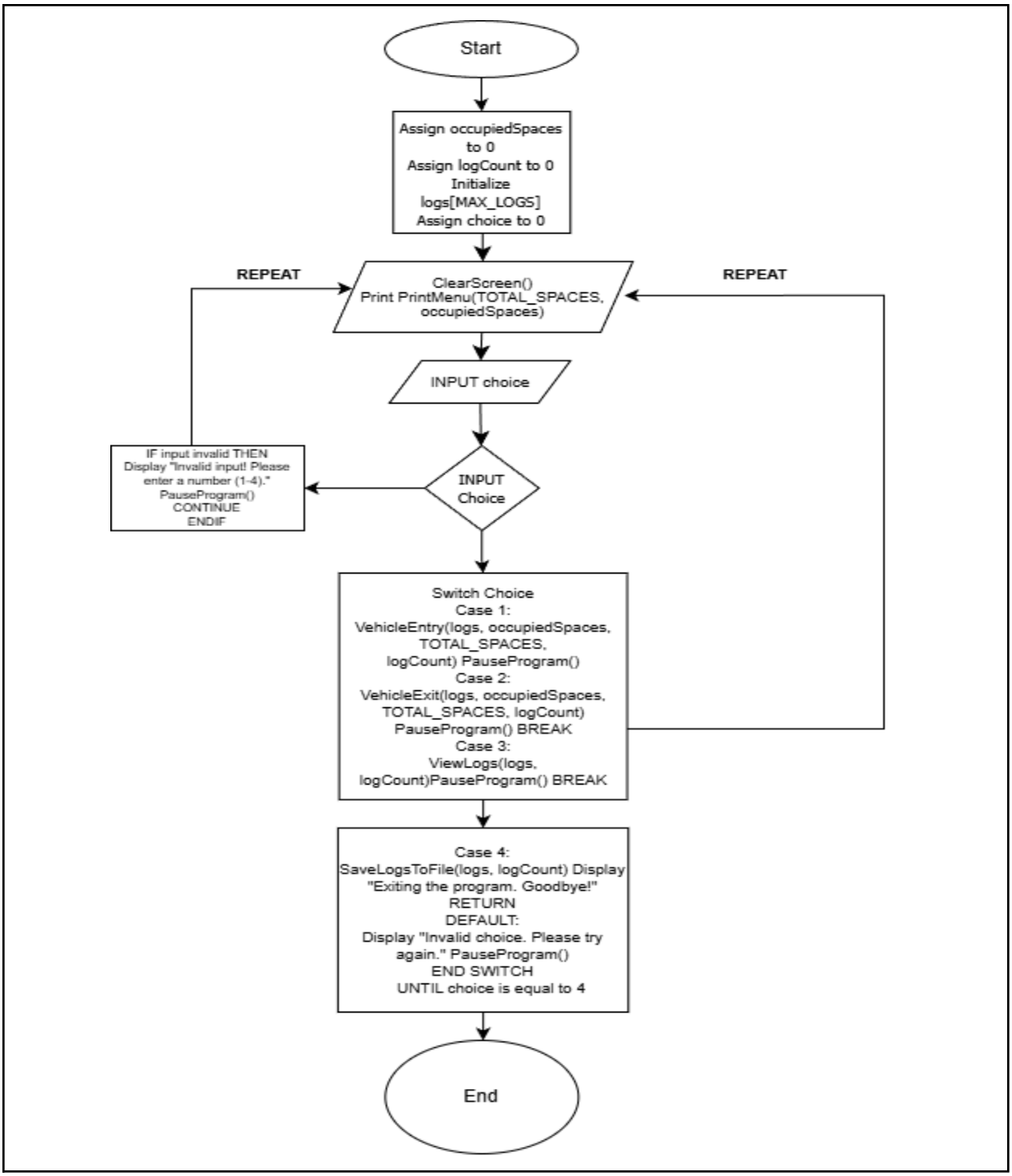


Figure 1: Main Code

This shows how the system operates. The program first initializes the dependencies or items for the program. Within the switch cases are options from the menu that provide you capabilities to manage the system.

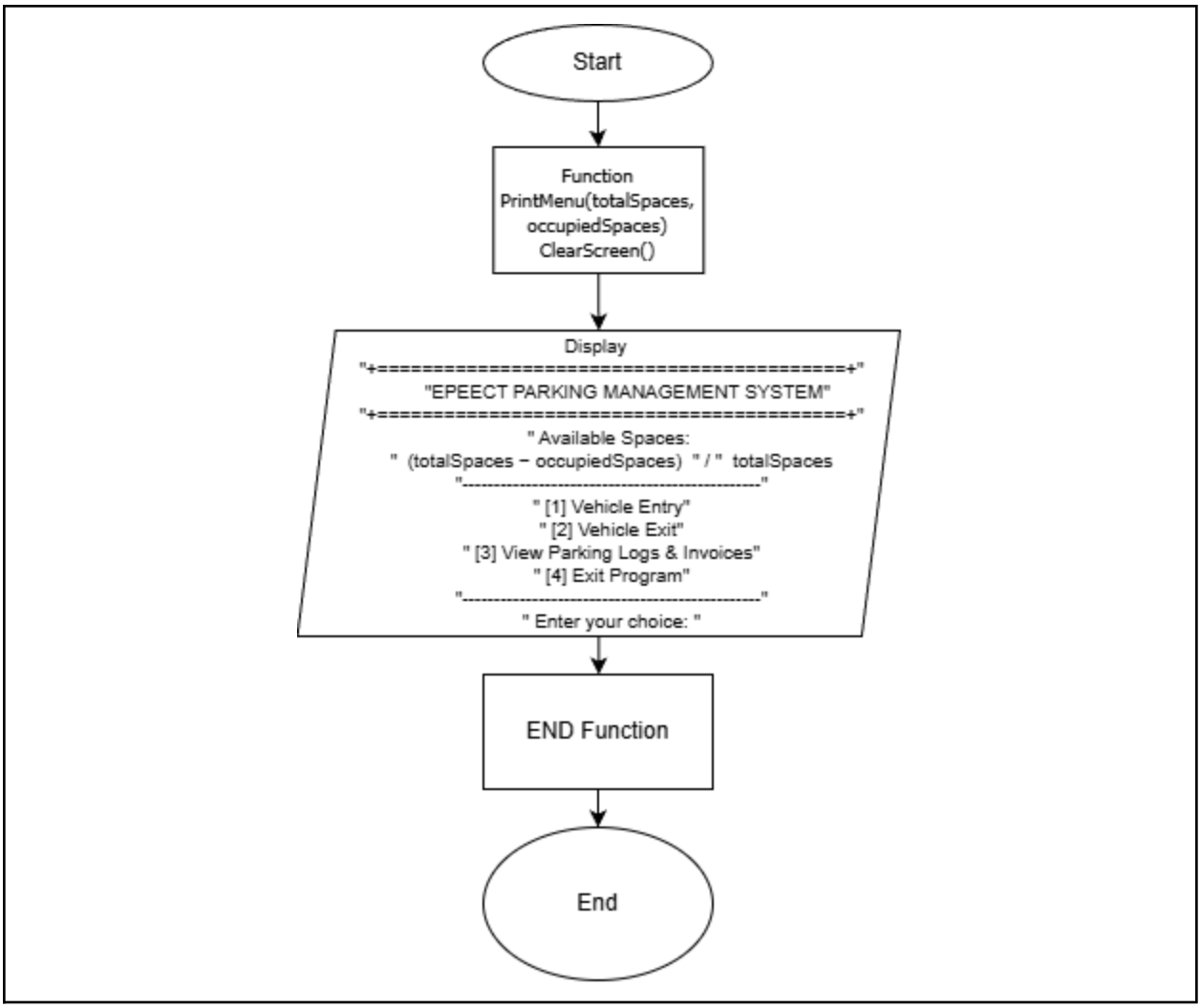


Figure 1.1: PrintMenu Function

This figure shows the function of displaying the menu, it is the user interface of the system and serves as indicator for information regarding the vehicles and availability of spaces. ClearScreen() is used to refresh the program's screen without re-printing the same menu below another.

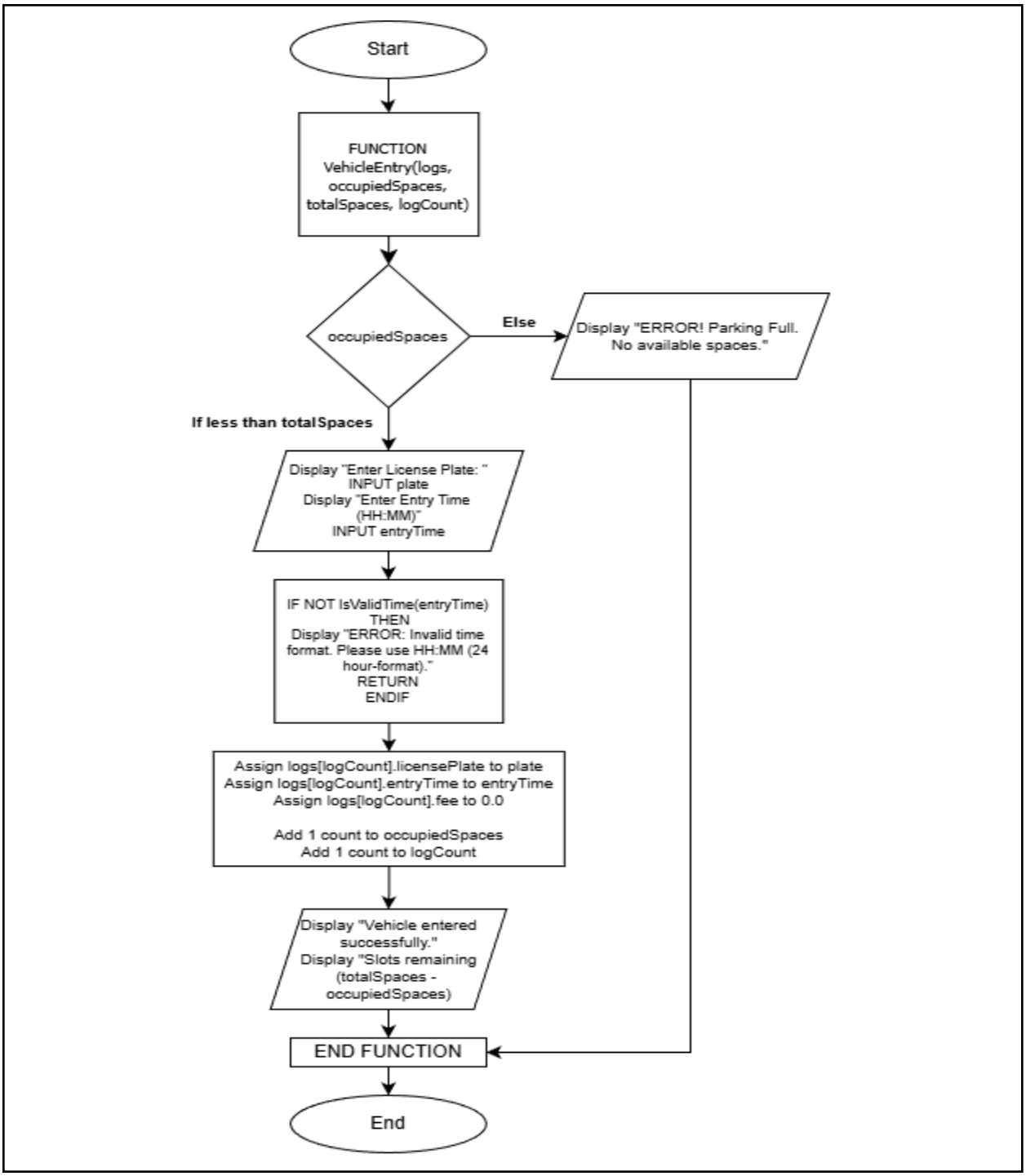


Figure 1.2: VehicleEntry Function

This function presents the process of assigning the vehicle's information to the system's logs. The logs is used to store information or data to be used in other functions. Afterwards, the current state of the parking slots are displayed.

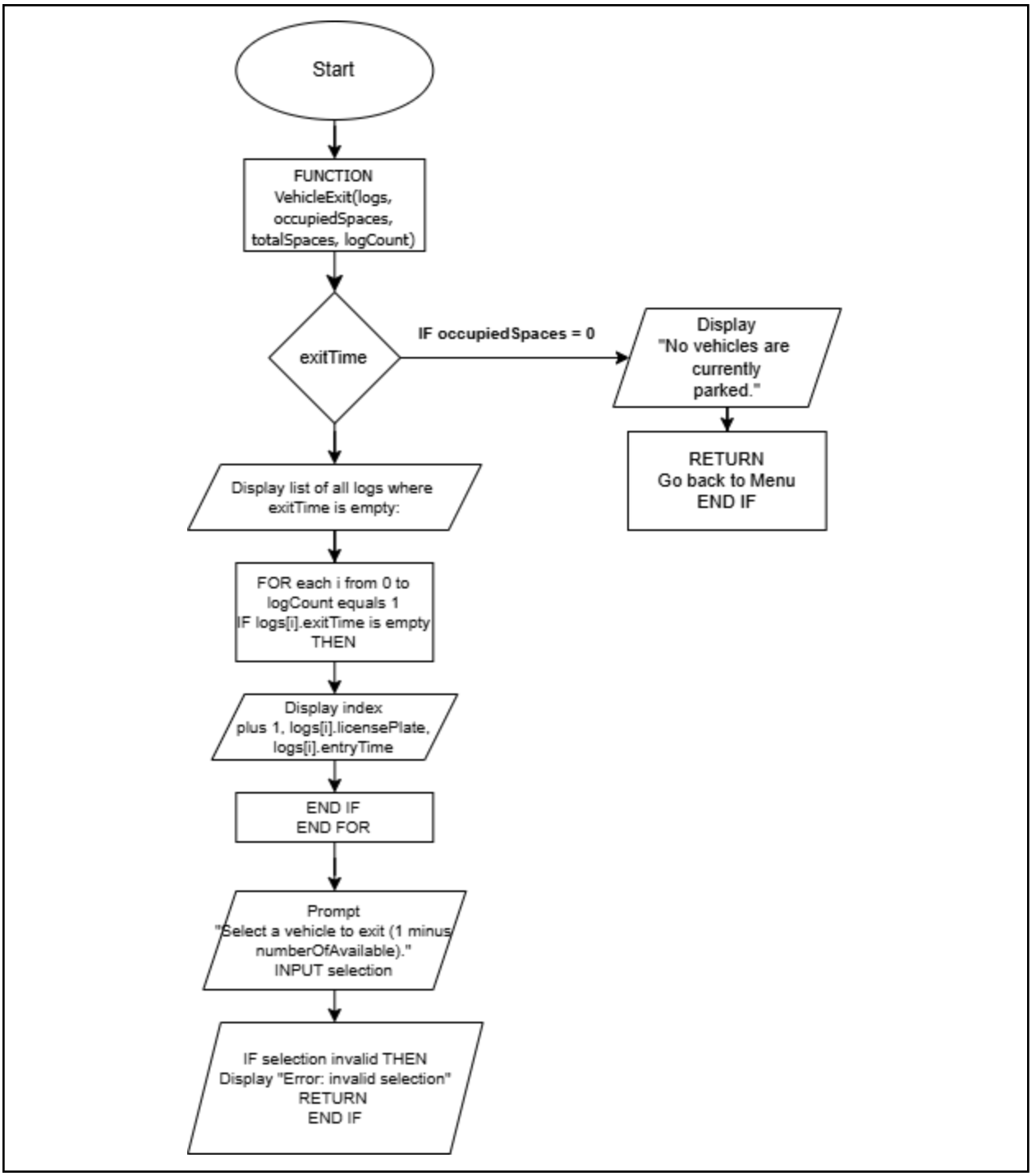


Figure 1.3: VehicleExit Function (Part 1)

This flowchart shows the process of removing vehicles. You are prompted to select a vehicle from the list of all logs where the exitTime is empty. If there's no occupied slots then the function should end.

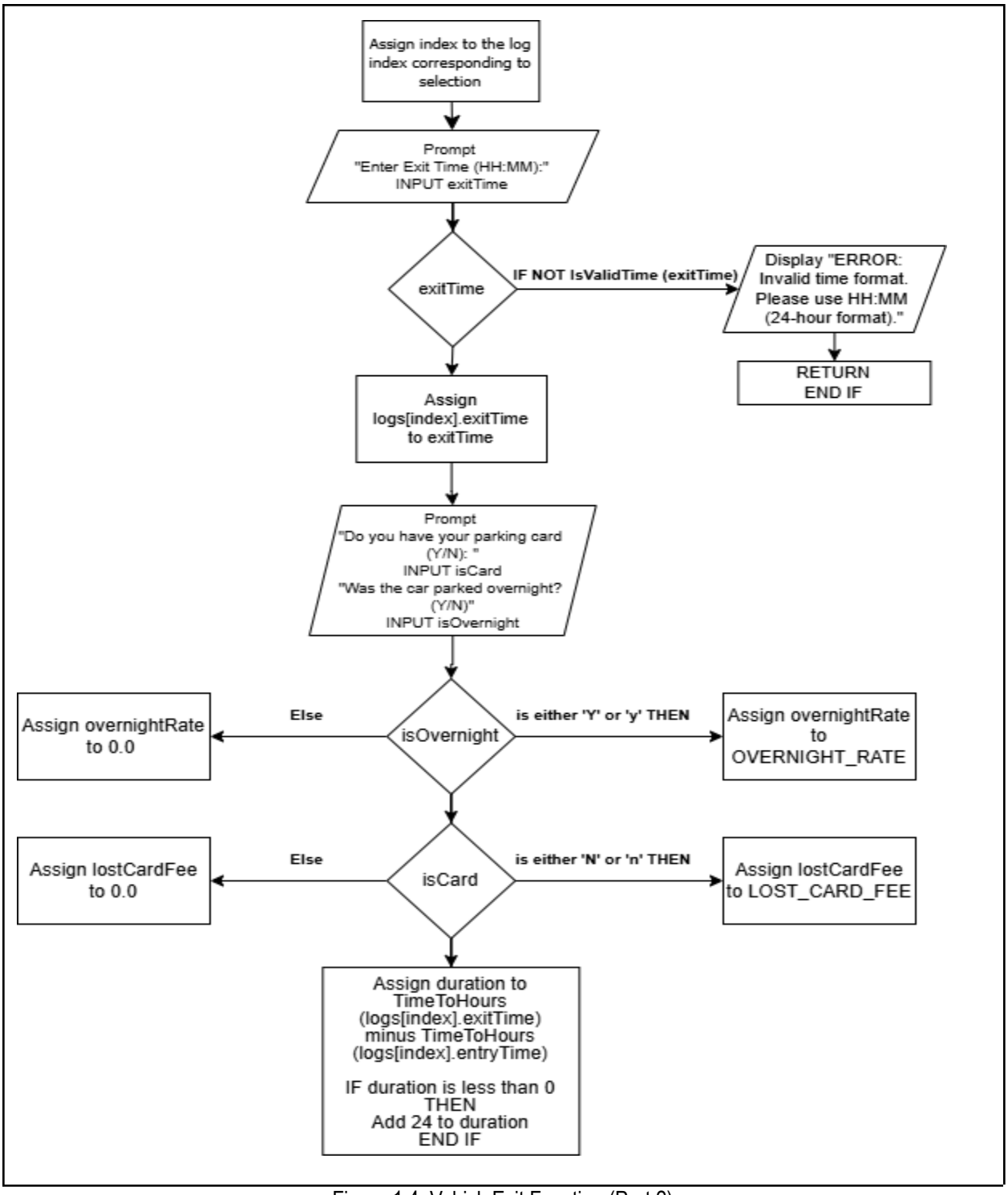


Figure 1.4: VehicleExit Function (Part 2)

Every parking ticket has information about how long the vehicle has been parked. This is used to calculate the fee and depending whether the driver has spent overnight parking or lost his ticket/card then there would be additional fees on the selected vehicle for exiting.

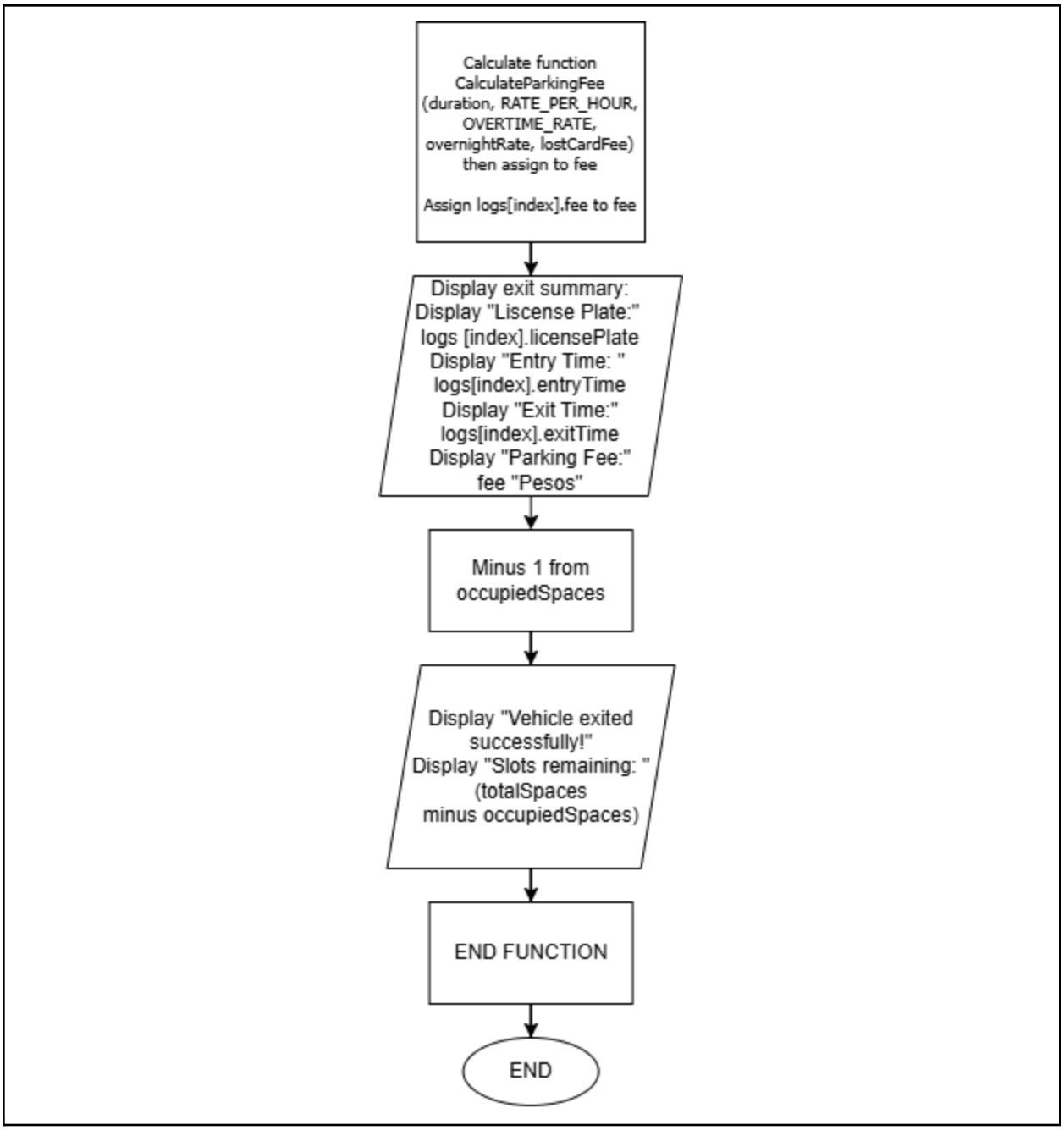


Figure 1.5: VehicleExit Function (Part 3)

This graphic shows the process or calculations to be made for the exiting vehicle. Once the fee has been calculated, the program removes one slot from the occupied spaces then displays that the vehicle has exited the parking lot.

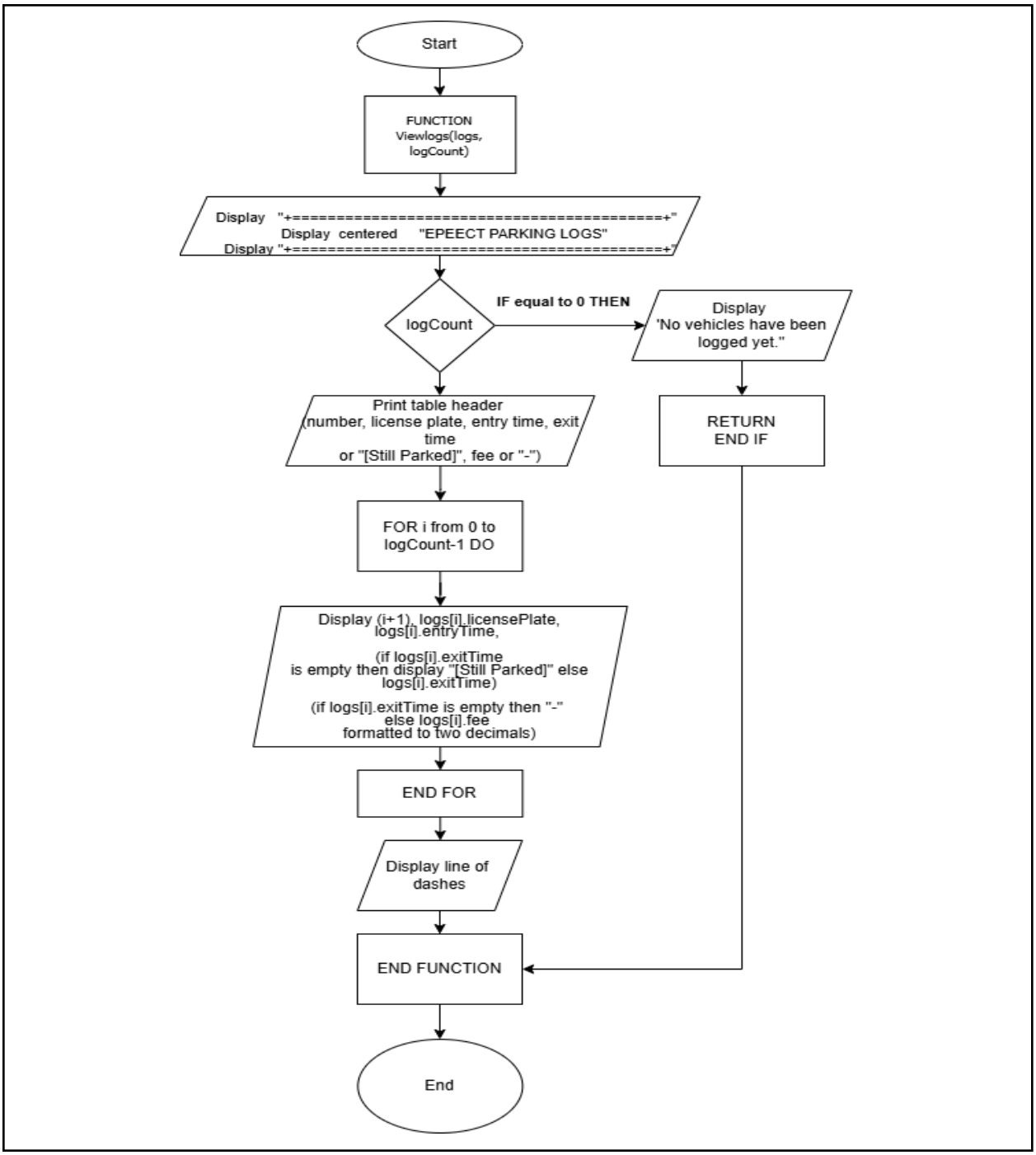


Figure 1.6: ViewLogs Function

This flowchart presents what's happening in the code when using the ViewLogs function to view the system's logs. It displays the current status of the parking lot and prints a table to show the records. If there are no records via logCount then it displays that there are no vehicles that have logged.

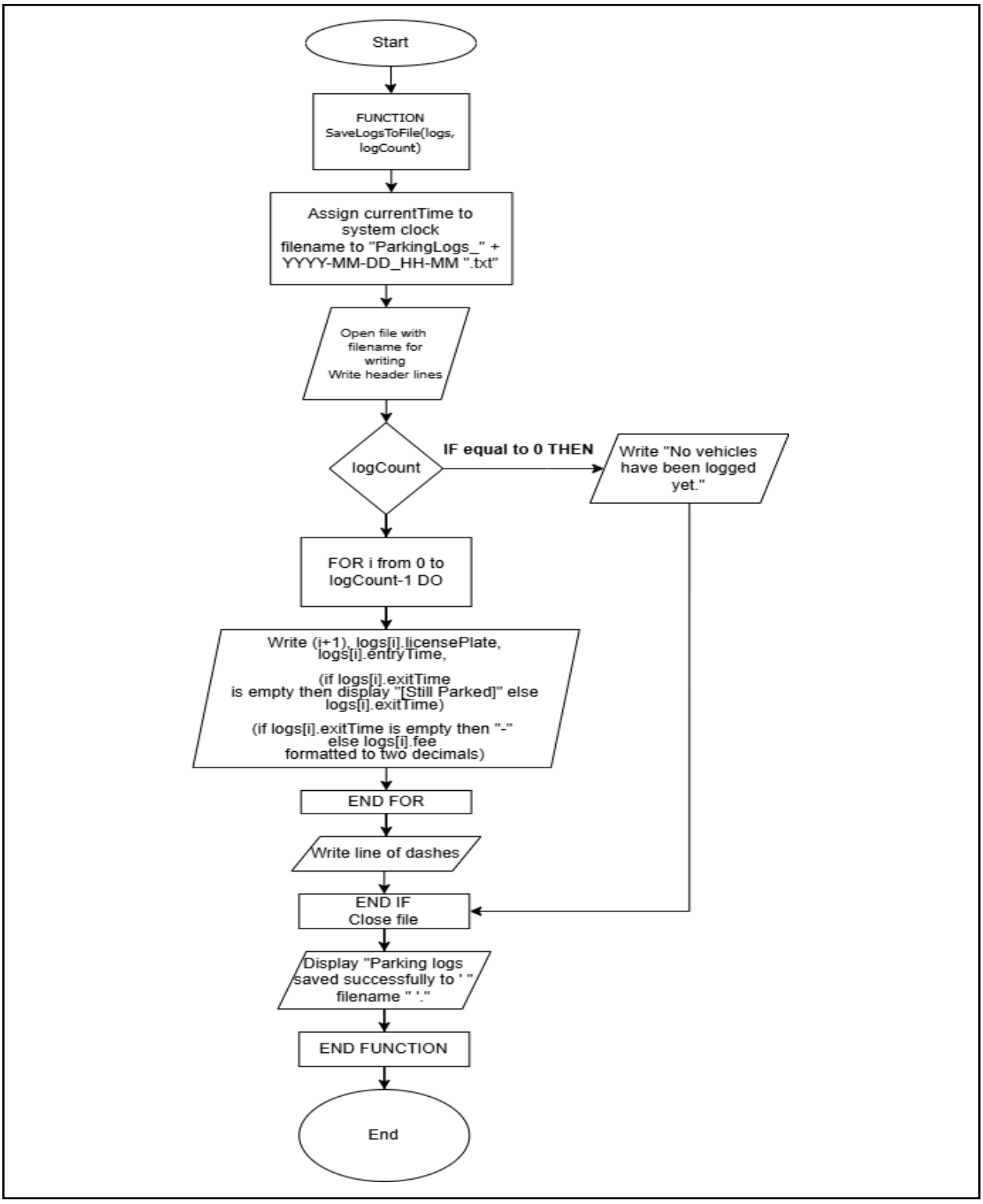


Figure 1.7: SaveToLogsFile Function

This figure reveals how saving the information to the logs work. Assigns the current time for the filename format, write's data based on the logCount's records then saves it in a created file. If LogCount has no records then it writes that no vehicles have been logged in the file.

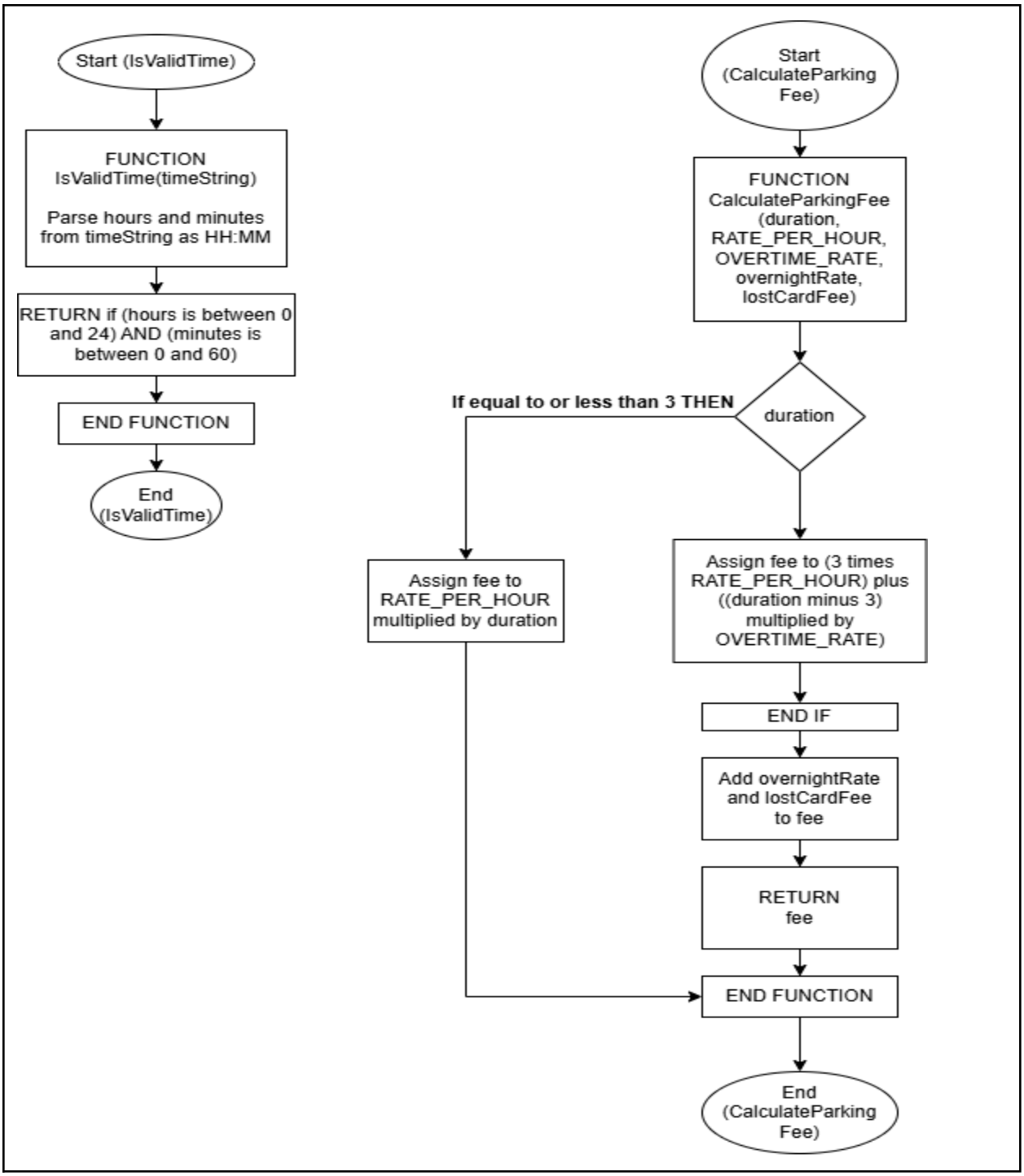


Figure 1.8: IsValidTime and CalculateParkingFee Functions

This figures shows two different flowcharts, one for showing the process of the IsValidTime function and the CalculateParkingFee function. On the left, it presents the format for inputting time and minutes. While on the right, it shows the formula used to calculate the fee for parking.

Pseudocode

The Enhancing Parking-Lot Experience by Efficient Tidying (EPEECT) is a Parking Management System. It tracks available parking slots, records the time of entry and exit, and calculates the proper parking fees based on hours parked. It also generates logs that track all the transactions that take place for easy monitoring of parking activities. The main goal of this system is to make managing parking-lots easier, reduce human errors, and provide organized records of all the activities in the parking area.

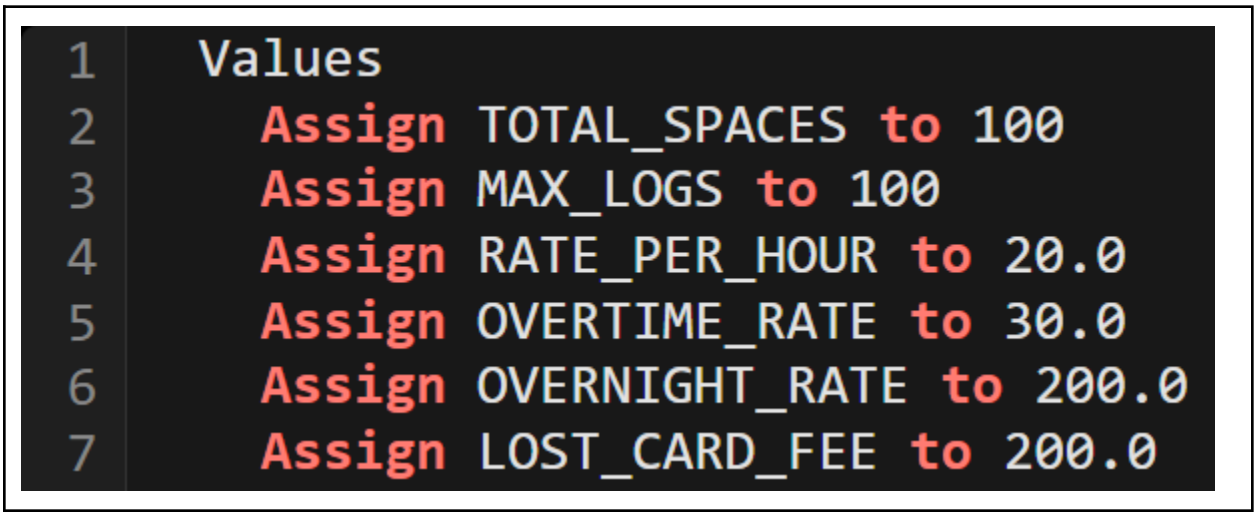


Figure 2.0: Pseudo code of the main code

This part of the pseudo code includes the assignment of the variables with their repsective values. The variables used available spaces, the logs and the values involved in calculating for the total parking fee.

```
Main Code
Start
    Assign occupiedSpaces to 0
    Assign logCount to 0
    logs[0…MAX_LOGS-1]  // array of ParkingLog
    Assign choice to 0

    REPEAT
        ClearScreen()
        PrintMenu(TOTAL_SPACES, occupiedSpaces)
        INPUT choice
        IF input invalid THEN
            Display "Invalid input! Please enter a number (1-4)."
            PauseProgram()
            CONTINUE
        END IF

        SWITCH choice
            CASE 1:
                VehicleEntry(logs, occupiedSpaces, TOTAL_SPACES, logCount)
                PauseProgram()
                BREAK
            CASE 2:
                VehicleExit(logs, occupiedSpaces, TOTAL_SPACES, logCount)
                PauseProgram()
                BREAK
            CASE 3:
                ViewLogs(logs, logCount)
                PauseProgram()
                BREAK
            CASE 4:
                SaveLogsToFile(logs, logCount)
                Display "Exiting the program. Goodbye!"
                RETURN
            DEFAULT:
                Display "Invalid choice. Please try again."
                PauseProgram()
        END SWITCH
    UNTIL choice is equal to 4
end
```

Figure 2.1: Pseudo code of the main code

This pseudo code shows the main code of the project where you can see how the functions are used in the code.. The program starts with an If statement to make sure that the inputted choice is a number where if it is not it will print invalid. Then it will use a switch with 4 cases representing the Vehicle entry, Vehicle exit, View logs and exit with a default of an invalid choice. This menu loops until the inputted choice is equal to 4.

```
FUNCTION PrintMenu(totalSpaces, occupiedSpaces)
    ClearScreen()
    Display "+=========================================+"
    Display centered "EPEECT PARKING MANAGEMENT SYSTEM"
    Display "+=========================================+"
    Display " Available Spaces: "  (totalSpaces - occupiedSpaces)  " / "  totalSpaces
    Display "-------------------------------------------"
    Display " [1] Vehicle Entry"
    Display " [2] Vehicle Exit"
    Display " [3] View Parking Logs & Invoices"
    Display " [4] Exit Program"
    Display "-------------------------------------------"
    Display " Enter your choice: "
END FUNCTION
```

Figure 2.2: Pseudo code of PrintMenu function

This part of the pseudo code prints the menu of the project including all of the options that are involved in the project. It also displays the remaining space that is left on the parking lot which could be seen at the top. At the end of the function is where the inputted choice is located which the driver would input on the main code.

```
FUNCTION VehicleEntry(logs, occupiedSpaces, totalSpaces, logCount)
    IF occupiedSpaces is less than totalSpaces THEN
        Display "Enter License Plate: "
        INPUT plate
        Display "Enter Entry Time (HH:MM): "
        INPUT entryTime

        IF NOT IsValidTime(entryTime) THEN
            Display "ERROR: Invalid time format. Please use HH:MM (24-hour format)."
            RETURN
        END IF

        Assign logs[logCount].licensePlate to plate
        Assign logs[logCount].entryTime to entryTime
        Assign logs[logCount].fee to 0.0

        Add 1 count to occupiedSpaces
        Add 1 count to logCount

        Display "Vehicle entered successfully."
        Display "Slots remaining: " (totalSpaces - occupiedSpaces)
    ELSE
        Display "ERROR! Parking Full. No available spaces."
    END IF
END FUNCTION
```

Figure 2.3: Pseudo code of VehicleEntry function

This part of the pseudo code involves the Vehicle entry function. It starts with an if statement where if the occupied spaces are less than the total space then asks for an input of the license plate and the entry time of the car. If the entry time is invalid using the isValidTime function then an error saying that the inputted time will be displayed. If the occupied space is not less than the total space then the program is going to display invalid saying that parking is full. This pseudo code also includes the assigning of the plate and entrytime to the logs array.

```
FUNCTION VehicleExit(logs, occupiedSpaces, totalSpaces, logCount)
    IF occupiedSpaces equals to 0 THEN
        Display "No vehicles are currently parked."
        RETURN
        Go go back to Menu
    END IF

    Display list of all logs where exitTime is empty:
        FOR each i from 0 to logCount-1
            IF logs[i].exitTime is empty THEN
                Display index + 1, logs[i].licensePlate, logs[i].entryTime
            END IF
        END FOR

    Prompt "Select a vehicle to exit (1 - numberOfAvailable): "
    INPUT selection
    IF selection invalid THEN
        Display "ERROR: Invalid selection."
        RETURN
    END IF

    Assign index to the log index corresponding to selection
    Prompt "Enter Exit Time (HH:MM): "
    INPUT exitTime
    IF NOT IsValidTime(exitTime) THEN
        Display "ERROR: Invalid time format. Please use HH:MM (24-hour format)."
        RETURN
    END IF
    Assign logs[index].exitTime to exitTime

    Prompt "Do you have your parking card? (Y/N): "
    INPUT isCard
    Prompt "Was the car parked overnight? (Y/N): "
    INPUT isOvernight

    IF inputted isOvernight is either 'Y' or 'y' THEN
        Assign overnightRate to OVERNIGHT_RATE
    ELSE
        Assign overnightRate to 0.0
    END IF

    IF inputted isCard is either 'N' or 'n' THEN
        Assign lostCardFee to LOST_CARD_FEE
    ELSE
        Assign lostCardFee to 0.0
    END IF

    Assign duration to TimeToHours(logs[index].exitTime) - TimeToHours(logs[index].entryTime)
    IF duration is less than 0 THEN
        Add 24 to duration
    END IF

    Calculate function CalculateParkingFee(duration, RATE_PER_HOUR, OVERTIME_RATE, overnightRate, lostCardFee) then assign to fee
    Assign logs[index].fee to fee

    Display exit summary:
        Display "License Plate: "  logs[index].licensePlate
        Display "Entry Time:    "  logs[index].entryTime
        Display "Exit Time:     "  logs[index].exitTime
        Display "Parking Fee:   "  fee  " Pesos"
    Minus 1 from occupiedSpaces <≠≤≥ANDORNOTMOD
    Display "Vehicle exited successfully!"
    Display "Slots remaining: "  (totalSpaces - occupiedSpaces)
END FUNCTION
```

Figure 2.4: Pseudo code of VehicleExit function

This part of the pseudo code is involved in the Vehicle exit. It starts with an if statement to see if the occupied space is equal to 0 if it is then the program will display that no vehicles are currently parked and it will loop back to the menu. If it is not then the program will display a table of all values from log which the exit time is empty. Then it would show a corresponding number to each car which the program will ask for the user input in which car they want to exit. If the input that the driver did is invalid then the program will print an error. If it is not then the program will ask for the Exit time, if the car was parked overnight, and if the driver has lost their parking card. If the driver chose n for the overnight parking then the program will assign 0 to the value of the overnight part of the calculator. If the driver chose n for the lost card then the program will assign 0 to the lost card value for the calculator. The program will then calculate the parking fee. Then it will show a summary including the Plate number, Entry Time, Exit time, and Parking fee. Lastly it would update the total number of parking spaces left by adding a new spot to the value.

```
FUNCTION ViewLogs(logs, logCount)
    Display "+========================================+"
    Display centered "EPEECT PARKING LOGS"
    Display "+========================================+"

    IF logCount is equal to 0 THEN
        Display "No vehicles have been logged yet."
        RETURN
    END IF

    Print table header (number, license plate, entry time, exit time or “[Still Parked]”, fee or “–”)
    FOR i from 0 to logCount-1 DO
        Display (i+1), logs[i].licensePlate, logs[i].entryTime,
                (if logs[i].exitTime is empty then display “[Still Parked]” else logs[i].exitTime),
                (if logs[i].exitTime is empty then “–” else logs[i].fee formatted to two decimals)
    END FOR
    Display line of dashes
END FUNCTION
```

Figure 2.5: Pseudo code of ViewLogs function

This part of the pseudo code is involved in the view logs option. Where it starts with displaying the title then proceeds to an if statement to check if the logcount is equal to 0 which if it is then the program will display no vehicles have been logged. If not then it would print a table with the plate number, entry time, exit time, and fee included. If a car is still inside the parking lot then it would be displayed as still parked.

```
FUNCTION SaveLogsToFile(logs, logCount)
    Assign currentTime to system clock
    filename ← "ParkingLogs_" + YYYY-MM-DD_HH-MM + ".txt"

    Open file with filename for writing
    Write header lines
    IF logCount is equal to 0 THEN
        Write "No vehicles have been logged yet."
    ELSE
        Write table header
        FOR i from 0 to logCount-1 DO
            Write (i+1), logs[i].licensePlate, logs[i].entryTime,
                  (if logs[i].exitTime is empty then display “[Still Parked]” else logs[i].exitTime),
                  (if logs[i].exitTime is empty then display “–” else logs[i].fee formatted to two decimals)
        END FOR
        Write line of dashes
    END IF
    Close file
    Display "Parking logs saved successfully to '"  filename  "'."
END FUNCTION
```

Figure 2.6: Pseudo code of SaveLogstoFile function

This pseudocode describes the SaveLogsToFile function, which saves all parking records into a timestamped text file. It first creates a filename based on the current date and time, then writes header lines to the file. If no logs exist,

it records a message indicating that no vehicles were logged. Otherwise, it lists each vehicle's license plate, entry time, exit time (or "[Still Parked]" if none), and computed fee (or "----" if not exited). Finally, it closes the file and confirms that the logs were successfully saved.

```
FUNCTION IsValidTime(timeString)
    Parse hours and minutes from timeString as HH:MM
    RETURN if (hours is between 0 and 24) AND (minutes is between 0 and 60)
END FUNCTION
```

Figure 2.7: Pseudo code of IsValidTime function

This part of the pseudo code involves making sure that the inputted time is valid within the 24 hour format. It uses a boolean in which it would register as true if the hours is between 0 and 24 and if the minutes are between 0 and 60.

```
FUNCTION CalculateParkingFee(duration, RATE_PER_HOUR, OVERTIME_RATE, overnightRate, lostCardFee)
    IF duration is equal or less than 3 THEN
        Assign fee to RATE_PER_HOUR × duration
    ELSE
        Assign fee to (3 × RATE_PER_HOUR) + ((duration – 3) × OVERTIME_RATE)
    END IF
    Add overnightRate and lostCardFee to fee
    RETURN fee
```

Figure 2.8: Pseudo code of CalculateParkingFee function

This part of the pseudo code shows the calculator used in the project. Where it starts with an if statement to see if the time duration is less than 3 hours where if it is then a rate of 20 pesos per hour will be used. If it is not then an overtime rate of 30 pesos will be used. It also accounts the overnight rate and lost card fee to the function by adding them.

This program first initializes parking slots and logs, then displays a menu for options to Park a Vehicle, Remove a Vehicle, View Logs, or Exit. In the case of parking, the system will ask the user to input the plate number of the vehicle and its type, and ask for the entry time if there is an available slot.

If not, it shows an error. When removing the vehicle, it will record the exit time and compute the total hours, charge 20 PHP for the first 3 hours, 30 PHP per extra hour, 200 PHP for overnight parking, or if the driver lost their parking card or not. It then frees a slot and updates the logs. The system can also display the available lots and sorted daily logs. After each transaction, the user can choose to either continue or exit the program.

Data Dictionary

The Data Dictionary provides a description of all the data elements used in the project. It defines the variables, their data types, and their specific purposes within the program to ensure clarity and consistency in data handling.

The table below summarizes all variables and data structures used in the system, including their size, type, and function. Each entry explains how the data contributes to the program's operation, such as tracking vehicles, calculating fees, and managing available parking slots.

Table 1: Data Dictionary

Data Name	Size	Data Type	Description
1. licensePlate	Varies (max 15)	string	To store the **license plate number** of the vehicle, which can be up to 15 alphanumeric characters long.
2. entryTime	Varies (max 5)	string	To store the **entry time** when the vehicle enters the parking lot, in "HH:MM" format
3. exitTime	Varies (max 5)	string	To store the **exit time** when the vehicle leaves the parking lot. This can be empty if the vehicle has not exited yet
4. fee	4 bytes	float	To store the **total parking fee** for the vehicles. Its calculated based on the parking duration, whethere the vehicle stayed overnight, or if the parking card was lost.
5. logs[ ]	100 * sizeof(ParkingLog)	ParkingLog[ ]	To store an array of **parking logs.** Each record holds information about a vehicle that has entered the parking lot.
6. occupiedSpaces	4 bytes	int	To store the **number of parking spots currently occupied**, indicating how many vehicles are parked at the moment.
7. logCount	4 bytes	int	To store the **total number of parking logs** or vehicles that have entered the parking lot, which tracks how many records have been created.
8. choice	4 bytes	int	To store the **user's menu**

			choice, which determines the next action the system should take (adding a new vehicle, viewing logs, or exiting).
9. plate	Varies (max 15)	string	To temporarily store the **license plate number** of a vehicle when it enters the parking lot.
10. entryTime	Varies (max 5)	string	To temporarily store the **entry time** of the vehicle at the time of its arrival.
11. exitTime	Varies (max 5)	string	To temporarily store the **exit time** when a vehicle leaves the parking lot.
12. availableIndices[]	100 * sizeof(int)	int[ ]	To store the **indices of vehicles** still parked in the lot. This helps the system track which vehicles have not exited yet.
13. count	4 bytes	int	To store a temporary count of how many vehicles are still parked, used to display the current list of vehicles in the parking lot.
14. exitVehicle	4 bytes	int	To store the **vehicle number** selected for exit from the list of parked vehicles.
15. isCard	1 bytes	char	To store whether the **parking card was lost** during exit. 'Y' indicates the card was lost, and 'N' indicates it was not.
16. isOvernight	1 bytes	char	To store whether the **vehicle was parked overnight**. 'Y' means yes, and 'N' means no.
17. duration	4 bytes	float	To store the **duration in**

			hours the vehicle was parked, calculated from the entry and exit times.
18..overnightRate	4 bytes	float	To store the **additional fee** for overnight parking, added if the vehicle stayed in the parking lot overnight.
19. lostCardFee	4 bytes	float	To store the **fee for a lost parking card**, charged if the card is not returned at the time of exit.
20. fee	4 bytes	float	To store the **total parking fee** for the vehicle, calculated based on the parking duration, overtime charges, overnight fees, and lost card fee.
21. filename	100 bytes	char[ ]	To store the **file name** used when saving parking logs. The file name includes a timestamp to ensure each file is unique, e.g. "ParkingLogs_2025-11-11 _15-30.txt".

Code

This C++ program is a simple Parking Lot Management System that tracks the vehicles entering and exiting the parking lot, taking the number of hours or overnight parking, and computes the price depending on the time of stay in the parking lot.

```
//+========================================+
//     EPEECT PARKING MANAGEMENT SYSTEM
//+========================================+

#include <iostream>     // For input and output
#include <string>       // For string handling
#include <cstdlib>      // For system("cls")
#include <limits>       // For numeric limits
#include <iomanip>      // For output formatting
#include <fstream>      // For file handling
#include <chrono>       // For timestamping log files
#include <ctime>        // For time formatting
using namespace std;

//+========================================+
//      STRUCTURES AND CONSTANTS
//+========================================+

struct ParkingLog {
   string licensePlate;    // Vehicle's license plate
   string entryTime;       // Entry time of the vehicle
   string exitTime;        // Exit time of the vehicle
   float fee;              // Parking fee
};

constexpr int   TOTAL_SPACES    = 100;             // Total parking spaces
constexpr int   MAX_LOGS        = 100;             // Maximum number of parking logs
constexpr float RATE_PER_HOUR   = 20.0f;           // Standard rate per hour
constexpr float OVERTIME_RATE   = 30.0f;           // Overtime parking rate
constexpr float OVERNIGHT_RATE  = 200.0f;          // Overnight parking rate
constexpr float LOST_CARD_FEE   = 200.0f;          // Lost card compensation fee

//+========================================+
//          FUNCTION DECLARATIONS
//+========================================+
// HELPER FUNCTION DECLARATIONS
bool isValidTime(const string &time);                                          // Declares the function to validate time
format
float timeToHours(string time);                                                // Declares the function to convert time to
hours
float calculateParkingFee(float duration, float RATE_PER_HOUR, float OVERTIME_RATE, float overnightRate,
float lostCardFee); // Declares the function to calculate parking fee
int  findVehicle(ParkingLog logs[], int logCount, string plate);               // Declares the function to find
vehicle by license plate
void printLogHeader(ostream &out);                                             // Declares the function to print log
header to file
string formatExitTime(const ParkingLog &log);                                  // Declares the function to format exit
time
string formatFee(const ParkingLog &log);                                       // Declares the function to format fee
void clearScreen();        // Declares the function to clear the console screen
```

```
void pauseProgram();                                                    // Declares the function to pause the program
void printCentered(ostream &out, string text, int width = 45);                  // Declares the function to print
centered text
// MAIN FUNCTION DECLARATIONS
void printMenu(int TOTAL_SPACES, int occupiedSpaces);                           // Declares the function to
print the menu
void vehicleEntry(ParkingLog logs[], int &occupiedSpaces, int TOTAL_SPACES, int &logCount);    // Declares the
function for vehicle entry
void vehicleExit(ParkingLog logs[], int &occupiedSpaces, int TOTAL_SPACES, int &logCount);     // Declares the
function for vehicle exit
void viewLogs(ParkingLog logs[], int logCount);                                 // Declares the function to view
parking logs
void saveLogsToFile(ParkingLog logs[], int logCount);                           // Declares the function to save
logs to a file

//+========================================+
//             MAIN FUNCTION
//+========================================+

int main() {
   int occupiedSpaces = 0;     // Current occupied parking spaces
   ParkingLog logs[MAX_LOGS];  // Array to store parking logs
   int logCount = 0;           // Current number of logs
   int choice = 0;             // User menu choice

   // Main program loop
   while (choice != 4) {
      clearScreen();
      printMenu(TOTAL_SPACES, occupiedSpaces);
      cin >> choice;
      cin.ignore(numeric_limits<streamsize>::max(), '\n');

      // Validate input
      if (cin.fail()) {
         cin.clear();
         cin.ignore(numeric_limits<streamsize>::max(), '\n');
         cout << "\nInvalid input! Please enter a number (1-4).\n";
         pauseProgram();
         continue;
      }

      // Handle user choice
      switch (choice) {
         case 1: vehicleEntry(logs, occupiedSpaces, TOTAL_SPACES, logCount); pauseProgram(); break;  //
Vehicle Entry
         case 2: vehicleExit(logs, occupiedSpaces, TOTAL_SPACES, logCount); pauseProgram(); break;   //
Vehicle Exit
         case 3: viewLogs(logs, logCount); pauseProgram(); break;                       // View Parking Logs
         case 4: saveLogsToFile(logs, logCount); cout << "Exiting the program. Goodbye!\n"; return 0;// Exit
Program
```

```
            default: cout << "Invalid choice. Please try again.\n"; pauseProgram(); break;              // Invalid Choice
        }
    }

    return 0;
}
//+=========================================+
//       HELPER FUNCTIONS DEFINITIONS
//+=========================================+

// Validate time format HH:MM
bool isValidTime(const string& time) {
    int h, m;
    if (sscanf(time.c_str(), "%d:%d", &h, &m) != 2) return false;
    return h >= 0 && h < 24 && m >= 0 && m < 60;
}

// Convert "HH:MM" string to float hours
float timeToHours(string time) {
    int hours, minutes;
    sscanf(time.c_str(), "%d:%d", &hours, &minutes);
    return hours + minutes / 60.0;
}

// Calculate parking fee
float calculateParkingFee(float duration, float RATE_PER_HOUR, float OVERTIME_RATE, float overnightRate,
float lostCardFee) {
    float fee = (duration <= 3) ? RATE_PER_HOUR * duration : (3 * RATE_PER_HOUR + (duration - 3) *
OVERTIME_RATE);
    fee += overnightRate + lostCardFee;
    return fee;
}

// Find vehicle index by license plate
int findVehicle(ParkingLog logs[], int logCount, string plate) {
    for (int i = 0; i < logCount; ++i) {
        if (logs[i].licensePlate == plate) return i;
    }
    return -1; // Not found
}

// Print log table header
void printLogHeader(ostream &out) {
    out << left
        << setw(5)  << "#"
        << setw(15) << "License Plate"
        << setw(15) << "Entry Time"
        << setw(15) << "Exit Time"
        << setw(15) << "Fee (Pesos)"
        << endl;
```

```
    out << string(60, '-') << endl;
}

// Format exit time for display
string formatExitTime(const ParkingLog &log) {
    return log.exitTime.empty() ? "[Still Parked]" : log.exitTime;
}

// Format fee for display
string formatFee(const ParkingLog &log) {
    if (log.exitTime.empty()) return "----";
    char buffer[20];
    snprintf(buffer, sizeof(buffer), "%.2f", log.fee);
    return string(buffer);
}

// Clear console screen
void clearScreen() {
    system("cls");
}

// Pause program until Enter is pressed
void pauseProgram() {
    cout << "\nPress Enter to continue...";
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
}

// Center text
void printCentered(ostream &out, string text, int width) {
    int pad = max(0, (width - (int)text.length()) / 2);
    out << string(pad, ' ') << text << endl;
}

//+=========================================+
//        MAIN FUNCTIONS DECLARATIONS
//+=========================================+

// Print menu
void printMenu(int TOTAL_SPACES, int occupiedSpaces) {
    clearScreen();
    cout << "+=========================================+\n";
    printCentered(cout, "EPEECT PARKING MANAGEMENT SYSTEM", 45);
    cout << "+=========================================+\n";
    cout << " Available Spaces: " << setw(3) << (TOTAL_SPACES - occupiedSpaces)
         << " / " << TOTAL_SPACES << "\n";
    cout << "-------------------------------------------\n";
    cout << " [1] Vehicle Entry\n";
    cout << " [2] Vehicle Exit\n";
    cout << " [3] View Parking Logs & Invoices\n";
    cout << " [4] Exit Program\n";
```

```
    cout << "-----------------------------------------------\n";
    cout << " Enter your choice: ";
}

// Vehicle entry
void vehicleEntry(ParkingLog logs[], int &occupiedSpaces, int TOTAL_SPACES, int &logCount) {
    string plate, entryTime;
    if (occupiedSpaces < TOTAL_SPACES) {
        cout << "\nEnter License Plate: ";
        getline(cin, plate);
        logs[logCount].licensePlate = plate;

        cout << "Enter Entry Time (HH:MM): ";
        getline(cin, entryTime);
        if (!isValidTime(entryTime)) {
            cout << "ERROR: Invalid time format. Please use HH:MM (24-hour format).\n";
            return;
        }

        logs[logCount].entryTime = entryTime;
        logs[logCount].fee = 0;
        occupiedSpaces++;
        logCount++;
        cout << "Vehicle entered successfully.\n";
        cout << "Slots remaining: " << (TOTAL_SPACES - occupiedSpaces) << "\n";
    } else {
        cout << "ERROR! Parking Full. No available spaces.\n";
    }
}

// Vehicle exit
void vehicleExit(ParkingLog logs[], int &occupiedSpaces, int TOTAL_SPACES, int &logCount) {
    string exitTime;

    if (occupiedSpaces == 0) {
        cout << "\nNo vehicles are currently parked.\n";
        return;
    }

    cout << "+=========================================+\n";
    printCentered(cout, "CURRENTLY PARKED VEHICLES", 45);
    cout << "+=========================================+\n";
    cout << left << setw(5) << "#" << setw(15) << "License Plate" << setw(15) << "Entry Time\n";
    cout << string(35, '-') << endl;

    int availableIndices[MAX_LOGS];
    int count = 0;

    for (int i = 0; i < logCount; i++) {
        if (logs[i].exitTime.empty()) {
```

```
        cout << left << setw(5) << count + 1
             << setw(15) << logs[i].licensePlate
             << setw(15) << logs[i].entryTime << endl;
        availableIndices[count++] = i;
    }
}

if (count == 0) {
    cout << "\nAll vehicles have already exited.\n";
    return;
}

int exitVehicle;
cout << "\nSelect a vehicle to exit (1 - " << count << "): ";
cin >> exitVehicle;
cin.ignore(numeric_limits<streamsize>::max(), '\n');

if (cin.fail() || exitVehicle < 1 || exitVehicle > count) {
    cin.clear();
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    cout << "ERROR: Invalid selection.\n";
    return;
}

int index = availableIndices[exitVehicle - 1];
cout << "Enter Exit Time (HH:MM): ";
getline(cin, exitTime);
if (!isValidTime(exitTime)) {
    cout << "ERROR: Invalid time format. Please use HH:MM (24-hour format).\n";
    return;
}
logs[index].exitTime = exitTime;

char isCard, isOvernight;
cout << "Do you have your parking card? (Y/N): ";
cin >> isCard;
cout << "Was the car parked overnight? (Y/N): ";
cin >> isOvernight;
cin.ignore(numeric_limits<streamsize>::max(), '\n');

float overnightRate = (isOvernight == 'Y' || isOvernight == 'y') ? OVERNIGHT_RATE : 0.0f;
float lostCardFee   = (isCard == 'N' || isCard == 'n') ? LOST_CARD_FEE : 0.0f;

float duration = timeToHours(logs[index].exitTime) - timeToHours(logs[index].entryTime);
if (duration < 0) duration += 24;
float fee = calculateParkingFee(duration, RATE_PER_HOUR, OVERTIME_RATE, overnightRate, lostCardFee);
logs[index].fee = fee;

cout << "+=========================================+\n";
printCentered(cout, "EXIT SUMMARY", 45);
```

```
    cout << "+=========================================+\n";
    cout << " License Plate: " << logs[index].licensePlate << endl;
    cout << " Entry Time:    " << logs[index].entryTime << endl;
    cout << " Exit Time:     " << logs[index].exitTime << endl;
    cout << " Parking Fee:   " << fixed << setprecision(2) << fee << " Pesos" << endl;
    cout << "-------------------------------------------\n";
    cout << "Vehicle exited successfully!\n";
    occupiedSpaces--;
    cout << "Slots remaining: " << (TOTAL_SPACES - occupiedSpaces) << "\n";
}

// View parking logs
void viewLogs(ParkingLog logs[], int logCount) {
    cout << "+=========================================+\n";
    printCentered(cout, "EPEECT PARKING LOGS", 45);
    cout << "+=========================================+\n";

    if (logCount == 0) {
        cout << "No vehicles have been logged yet.\n";
        return;
    }
    printLogHeader(cout);
    for (int i = 0; i < logCount; ++i) {
        cout << left
             << setw(5)  << i + 1
             << setw(15) << logs[i].licensePlate
             << setw(15) << logs[i].entryTime
             << setw(15) << formatExitTime(logs[i])
             << setw(15) << formatFee(logs[i])
             << endl;
    }
    cout << string(60, '-') << endl;
}

// Save logs to a file
void saveLogsToFile(ParkingLog logs[], int logCount) {
    using namespace std::chrono;
    auto now = system_clock::now();
    std::time_t now_time = system_clock::to_time_t(now);
    std::tm localTime = *std::localtime(&now_time);

    // Create a timestamped filename like ParkingLogs_2025-11-11_15-30.txt
    char filename[100];
    std::strftime(filename, sizeof(filename), "ParkingLogs_%Y-%m-%d_%H-%M.txt", &localTime);

    ofstream file(filename, ios::out);
    if (!file) {
        cout << "Error: Could not create log file.\n";
        return;
    }
```

```
    file << "+========================================+\n";
    printCentered(file, "EPEECT PARKING LOGS", 45);
    file << "+========================================+\n";

    if (logCount == 0) {
        file << "No vehicles have been logged yet.\n";
    } else {
        printLogHeader(file);
        for (int i = 0; i < logCount; ++i) {
            file << left
                 << setw(5)  << i + 1
                 << setw(15) << logs[i].licensePlate
                 << setw(15) << logs[i].entryTime
                 << setw(15) << formatExitTime(logs[i])
                 << setw(15) << formatFee(logs[i])
                 << endl;
        }
        file << string(60, '-') << endl;
    }

    file.close();
    cout << "\nParking logs saved successfully to '" << filename << "'.\n";
}
```

Figure 3. Raw code of the Parking Management System>

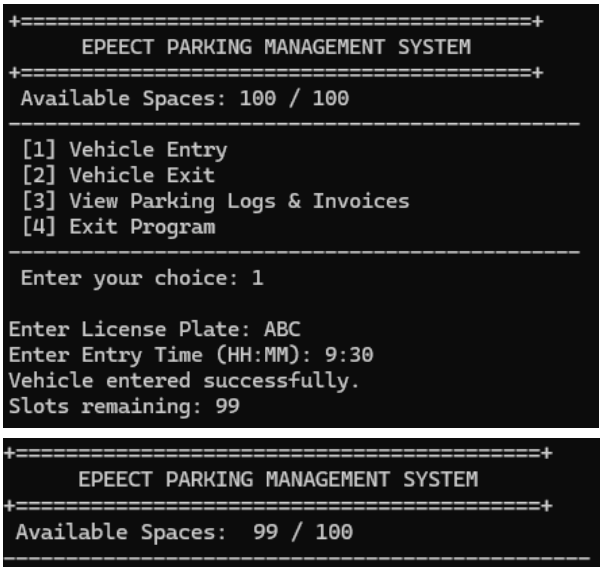
In this program, we keep track of vehicles entering and leaving the parking lot. It manages the total number of parking spaces available, records the vehicles inside the parking lot and calculates the parking fees based on how long the vehicles stay inside. Everything happens within a loop, allowing the user to perform multiple actions until the user decides to exit the program.

Results and Discussion

The EPEECT Parking Management System efficiently manages the entries, exits, and parking fees of vehicles. It automates fee computation and records the logs for organized parking operations.

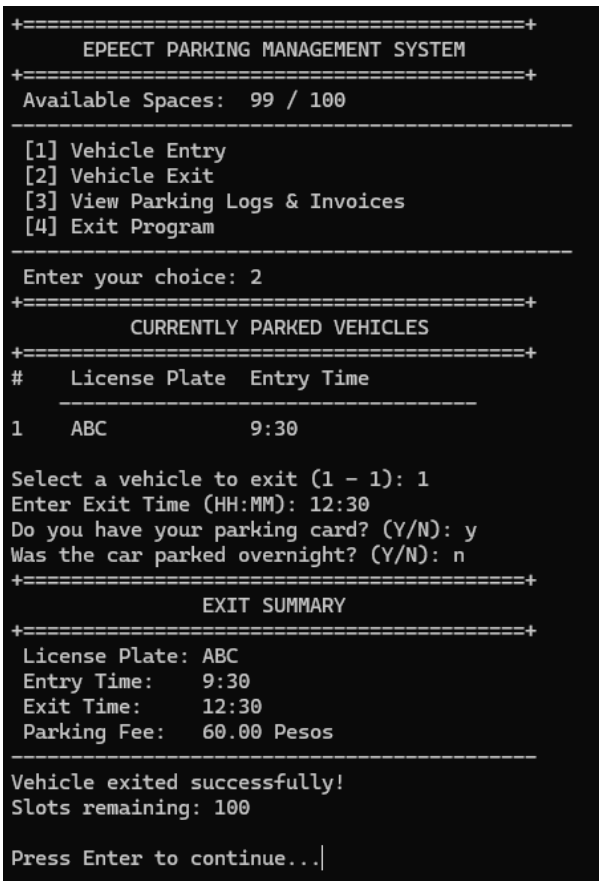
Menu Display

Upon executing the program, the system displays a main menu showing how many slots are available and the options: Vehicle Entry and Exit, View Parking Logs & Invoices, and Exit. It ensures easy navigation for users.



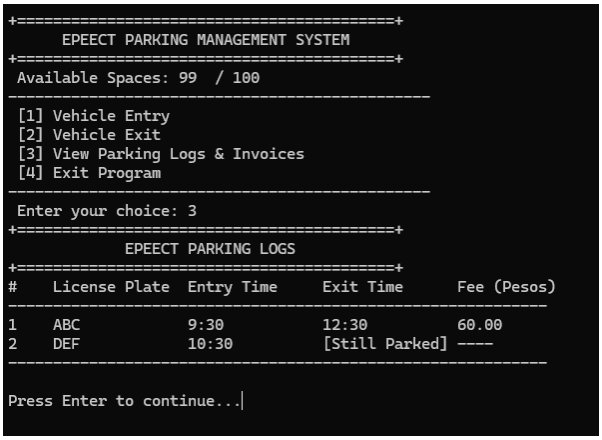
Vehicle Entry

When choosing the first option, the user is asked to input a license plate and entry time. The system validates the time format, records the data, and updates the remaining slots simulating a real-time parking entry



Vehicle Exit

When exiting, the system lists the parked vehicles, asks the user for the time of exit, and checks for overnight parking or lost card. It then computes the fee automatically based on duration. Then it shows an exit summary containing parking information and the fee, then updates the slots remaining.

	**Viewing Logs** Displays all parking records with license plats, entry and exit times, and total fees in a formatted table.
	**Exit** When exiting, the program automatically saves the logs then immediately closes.

Conclusion

The EPEECT Parking Management System successfully automates the tracking of parking spaces, vehicle entries, and exits while computing corresponding parking fees accurately. It simplifies record management through an efficient logging system and data storage feature, reducing manual work and human error. The program demonstrates how structured programming and file handling can be applied to solve real-world problems in parking management.

Recommendation

Enhance the system by integrating GUI for easier navigation and better user experience. Future versions could also include a database system to handle larger data storage and retrieval efficiently. Additionally, implementing time-based automation and license plate recognition could further improve accuracy and convenience for users.

References

Bajçinovci, B. Q., & Bajçinovci, M. (2019). Anxiety and Urban Stress for Parking Spots. Journal of Science, Humanities and Arts (JOSHA), 6(1).

https://doi.org/10.17160/josha.6.1.525

Mejia, A. (2017, December 5). *Rappler readers share how they really feel about Manila traffic and parking issues*. RAPPLER.

https://www.rappler.com/brandrap/data-stories/190406-rappler-traffic-parking-online-survey/