PROGRAMMING IN C
PROJECT

SUBMMITED BY- PARTH PANT
SAP ID – 590025382
BATCH – 18
COURSE – B.Tech CSE

AUTOMATED PARKING REVENUE SYSTEM

An advanced system for managing vehicle with parking revenue tracking

ABSTRACT

This program implements a comprehensive AUTOMATEDPARKING REVENUE SYSTEM in C that handles vehicle parking operations, revenue tracking, and slot management . The system allows users to park vehicles with different types (bike, car, truck) and VIP options , automatically generates unique ticket IDs for each vehicle , calculates parking bills based on duration and vehicle type with time-based discounts and coupon support, maintains a visual parking map showing slot occupancy, logs all transactions to a file for record keeping, and tracks total revenue generated. The system integrates advanced memory allocation, structures, file functions, and time operations. This project demonstrates application of multiple fundamental and advanced C concepts including memory management, time-based calculations, input validation, and real-world business logic implementation.

FEATURES -:

• **Dynamic Memory Allocation:** Flexible slot count at runtime

• **File-Based Logging:** All transactions recorded in parking_log.txt

• **Time-Based Billing:** 10-second units with configurable rates

• **VIP Support:** Double pricing for VIP parking slots

• **Discount System:** Long stay discount and coupon support

• **Input Validation:** Type checking and invalid input recovery

• **Case-Insensitive Input:** Flexible input handling

• **Visual Parking Map:** Grid-based slot display

• **Status Dashboard:** Real-time occupancy and revenue tracking

• **Robust Error Handling:** Graceful handling of edge cases

The Code-

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <ctype.h>

typedef struct {
    int slot;
    char vehicleNo[20];
    char ticketID[20];
    char type[10];
    int isVIP;
    time_t entryTime;
    int occupied;
} Parking;

double totalRevenue = 0;

int isAlreadyParked(Parking *p, int n, char *num) {
    for (int i = 0; i < n; i++) {
        if (p[i].occupied && strcmp(p[i].vehicleNo, num) == 0)
            return 1;
    }
    return 0;
}
```

```
void generateTicketID(char *id, int slot) {
    sprintf(id, "TKT%04d%02d", rand() % 9000 + 1000, slot);
}

void logToFile(Parking *p, double stay, double bill) {
    FILE *f = fopen("parking_log.txt", "a");
    if (!f) {
        printf("Error! Unable to open parking_log.txt for writing.\n");
        return;
    }

    time_t entry = p->entryTime;
    char timeIn[30], timeOut[30];
    time_t exitTime = time(NULL);
    struct tm *in = localtime(&entry);
    struct tm *out = localtime(&exitTime);

    strftime(timeIn, sizeof(timeIn), "%Y-%m-%d %H:%M:%S", in);
    strftime(timeOut, sizeof(timeOut), "%Y-%m-%d %H:%M:%S", out);

    fprintf(f, "Vehicle: %s | Ticket: %s | Type: %s | VIP: %d\n",
            p->vehicleNo, p->ticketID, p->type, p->isVIP);
    fprintf(f, "Entry: %s | Exit: %s\n", timeIn, timeOut);
    fprintf(f, "Stay: %.0f sec | Bill: %.2f\n\n", stay, bill);
    fclose(f);
}
double getRate(char *type) {
    if (strcmp(type, "bike") == 0) return 2;
    if (strcmp(type, "car") == 0) return 3;
    if (strcmp(type, "truck") == 0) return 5;
    return 2;
}

int isValidType(char *type) {
    return strcmp(type, "bike") == 0 || strcmp(type, "car") == 0 ||
           strcmp(type, "truck") == 0;
}

void showParkingStatus(Parking *p, int n, double revenue) {
    int parkCount = 0, vipCount = 0, emptyCount = 0;
    for (int i = 0; i < n; i++) {
        if (p[i].occupied) {
            parkCount++;
            if (p[i].isVIP) vipCount++;
        } else emptyCount++;
    }

    printf("\n======== PARKING STATUS ========\n");
    printf("Total Slots: %d\n", n);
    printf("Occupied Slots: %d\n", parkCount);
    printf("VIP Slots Used: %d\n", vipCount);
    printf("Available Slots: %d\n", emptyCount);
    printf("Total Revenue: %.2f\n", revenue);
}
```

```
void showParkingMap(Parking *p, int n) {
    int columns = 3;
    int rows = (n + columns - 1) / columns;

    printf("\n======= PARKING MAP =======\n");

    for (int r = 0; r < rows; r++) {
        for (int c = 0; c < columns; c++) {
            int idx = r * columns + c;
            if (idx < n)
                printf("[ %s ] ", p[idx].occupied ? p[idx].vehicleNo : "EMPTY");
            else
                printf("       ");
        }
        printf("\n");
    }

    printf("======= END MAP =======\n");
}

void parkVehicle(Parking *p, int n) {
    char num[20], type[10];
    int vip;

    printf("Enter Vehicle Number: ");
    scanf("%s", num);
```

```
if (isAlreadyParked(p, n, num)) {
    printf("Vehicle already parked!\n");
    return;
}

printf("Vehicle Type (bike/car/truck): ");
scanf("%s", type);

if (!isValidType(type)) {
    printf("Invalid vehicle type! Only bike, car, truck allowed.\n");
    return;
}

printf("VIP Slot? (1=Yes, 0=No): ");
scanf("%d", &vip);

for (int i = 0; i < n; i++) {
    if (!p[i].occupied) {
        strcpy(p[i].vehicleNo, num);
        strcpy(p[i].type, type);
        p[i].isVIP = vip;
        p[i].occupied = 1;
        generateTicketID(p[i].ticketID, i + 1);
        p[i].entryTime = time(NULL);

        printf("Parked at Slot %d\n", i + 1);
        printf("Ticket ID: %s\n", p[i].ticketID);
```

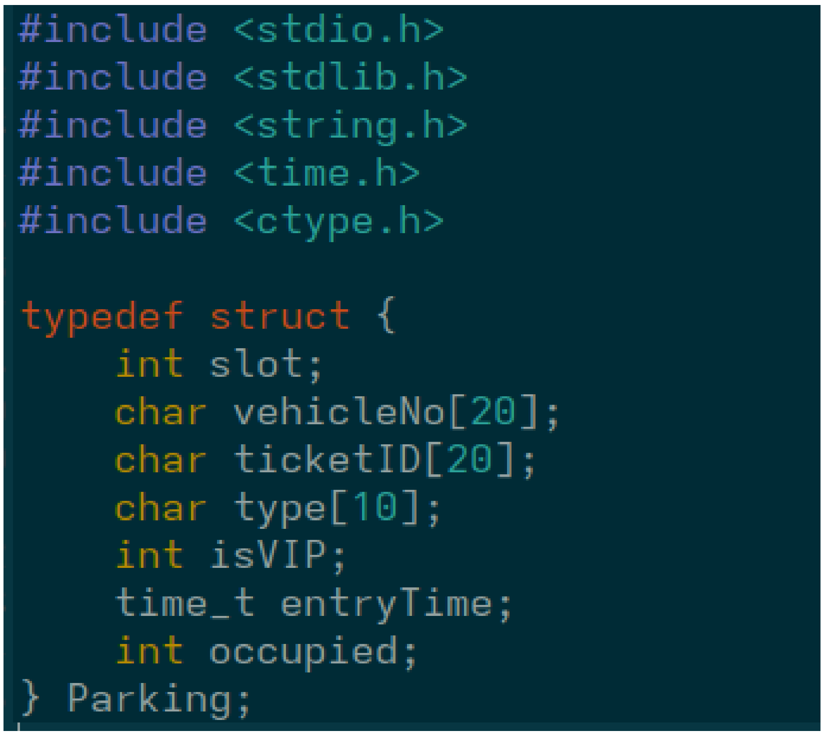
```
            return;
        }
    }

    printf("Parking full!\n");
}
```

```
void removeVehicle(Parking *p, int n) {
    char ticket[20], coupon[20];
    printf("Enter Ticket ID: ");
    scanf("%s", ticket);

    for (int i = 0; i < n; i++) {
        if (p[i].occupied && strcmp(p[i].ticketID, ticket) == 0) {

            time_t exitTime = time(NULL);
            double stay = difftime(exitTime, p[i].entryTime);
            double units = stay / 10;
            if (units < 1) units = 1;

            double rate = getRate(p[i].type);

            if (stay > 120)
                rate *= 0.9;

            double bill = rate * units;
            if (p[i].isVIP) bill *= 2;

            printf("Apply Coupon? (type NONE for no): ");
            scanf("%s", coupon);

            for (int j = 0; coupon[j]; j++)
                coupon[j] = toupper(coupon[j]);
            if (strcmp(coupon, "SAVE10") == 0)
                bill *= 0.90;

            printf("Time Parked: %.0f sec\n", stay);
            printf("Bill: Rs %.2f\n", bill);

            totalRevenue += bill;
            logToFile(&p[i], stay, bill);

            p[i].occupied = 0;
            return;
        }
    }
    printf("Invalid Ticket ID!\n");
}
int main() {
    srand(time(NULL));

    int n;
    printf("Enter number of parking slots: ");
    scanf("%d", &n);
    if (n <= 0) {
        printf("Invalid slot number!\n");
        return 0;
    }

    Parking *p = (Parking *)malloc(n * sizeof(Parking));
    for (int i = 0; i < n; i++) {
        p[i].slot = i + 1;
        p[i].occupied = 0;
    }

    while (1) {
        printf("\n1. Park Vehicle\n2. Remove Vehicle\n3. Display Status\n4.
Parking Map\n5. Exit\n");
        printf("Enter Choice: ");
        int choice;
        scanf("%d", &choice);

        if (choice == 1) parkVehicle(p, n);
        else if (choice == 2) removeVehicle(p, n);
        else if (choice == 3) showParkingStatus(p, n, totalRevenue);
        else if (choice == 4) showParkingMap(p, n);
        else if (choice == 5) break;
        else printf("Invalid Choice!\n");
    }

    free(p);
    return 0;
}
```

CODE EXPLANATION –

1. Include Directives and Structure Definition

The program begins by including essential libraries: stdio.h for input/output, stdlib.h for dynamic memory functions, string.h for string operations, time.h for timestamp management, and ctype.h for character conversion. The Parking structure is defined using typedef, encapsulating several fields that represent a parking slot's complete state including slot number, vehicle number, unique ticket identifier, vehicle category, VIP status flag, entry timestamp, and occupancy indicator.



```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <ctype.h>

typedef struct {
    int slot;
    char vehicleNo[20];
    char ticketID[20];
    char type[10];
    int isVIP;
    time_t entryTime;
    int occupied;
} Parking;
```

2. Global Revenue Tracker

A global variable totalRevenue maintains cumulative earnings across all parking transactions. This eliminates the need for passing the variable between functions and supports seamless aggregation of parking slot payouts.

```
double totalRevenue = 0;
```

3. Duplicate Detection Function

The isAlreadyParked function prevents double-parking violations by scanning all occupied parking entries and matching vehicle registration numbers. It ensures that the same vehicle does not re-enter without being removed first. The function returns 1 if a match is found, otherwise 0.

```
int isAlreadyParked(Parking*p, int n, char *num) {
    for (int i = 0; i < n; i++)
        if (p[i].occupied && strcmp(p[i].vehicleNo, num) == 0)
            return 1;
    return 0;
}
```

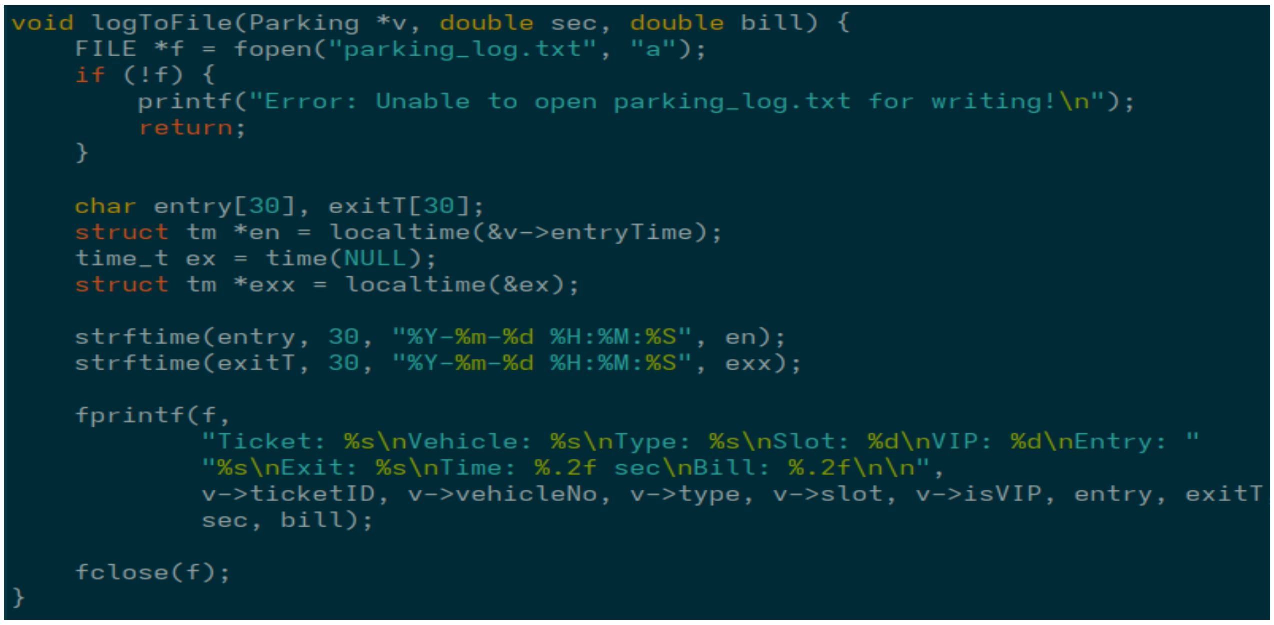
4. Unique Ticket Generation

generateTicketID creates distinctive identifiers by combining a random 4-digit number with a 2-digit slot number using sprintf formatting. The rand function generates pseudo-random values seeded at program start, producing tickets in the format TKTxxxxdd followed by 6 digits, ensuring each parking session has a traceable unique reference.

```
void generateTicketID(char *id, int slot) {
    sprintf(id, "TKT%04d%02d", rand() % 9000 + 1000, slot);
}
```

5. Transaction Logging System

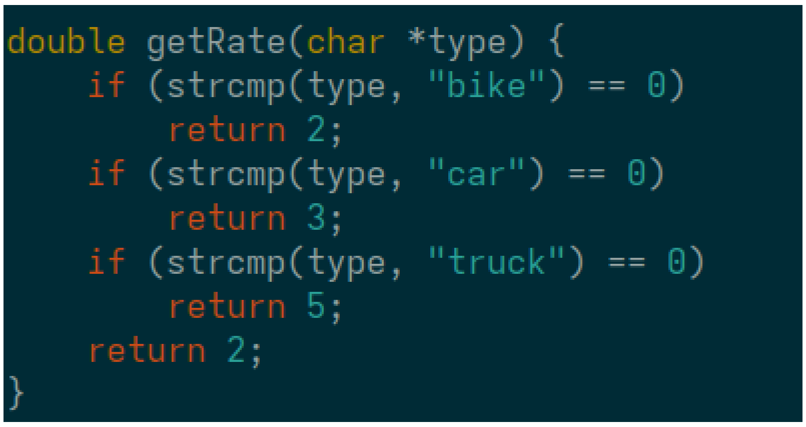
The logToFile function appends comprehensive parking records to parking_log.txt using file append mode. It stores UNIX timestamps in human-readable format using localtime and strftime, including the entry and exit times, parking duration, and final bill. This maintains traceability and audit trails for manual record tracking.



```c
void logToFile(Parking *v, double sec, double bill) {
    FILE *f = fopen("parking_log.txt", "a");
    if (!f) {
        printf("Error: Unable to open parking_log.txt for writing!\n");
        return;
    }

    char entry[30], exitT[30];
    struct tm *en = localtime(&v->entryTime);
    time_t ex = time(NULL);
    struct tm *exx = localtime(&ex);

    strftime(entry, 30, "%Y-%m-%d %H:%M:%S", en);
    strftime(exitT, 30, "%Y-%m-%d %H:%M:%S", exx);

    fprintf(f,
            "Ticket: %s\nVehicle: %s\nType: %s\nSlot: %d\nVIP: %d\nEntry: "
            "%s\nExit: %s\nTime: %.2f sec\nBill: %.2f\n\n",
            v->ticketID, v->vehicleNo, v->type, v->slot, v->isVIP, entry, exitT
            sec, bill);

    fclose(f);
}
```

6. Rate Calculation Logic

getRate determines base parking charges by vehicle type classification: bikes are charged Rs 2 per billing unit, cars Rs 3, and trucks Rs 5. This tiered pricing structure reflects vehicle size and space consumption, with a default fallback to bike rates for unrecognized types ensuring system robustness.



```c
double getRate(char *type) {
    if (strcmp(type, "bike") == 0)
        return 2;
    if (strcmp(type, "car") == 0)
        return 3;
    if (strcmp(type, "truck") == 0)
        return 5;
    return 2;
}
```

7. Input Validation Function

isValidType enforces strict vehicle category validation by checking input strings against three permitted options: bike, car, and truck. This function prevents billing errors and ensures consistent rate application by rejecting any non-standard vehicle classifications before they enter the parking system.

```c
int isValidType(char *type) {
    return strcmp(type, "bike") == 0 || strcmp(type, "car") == 0 || strcmp(type,
"truck") == 06
}
```

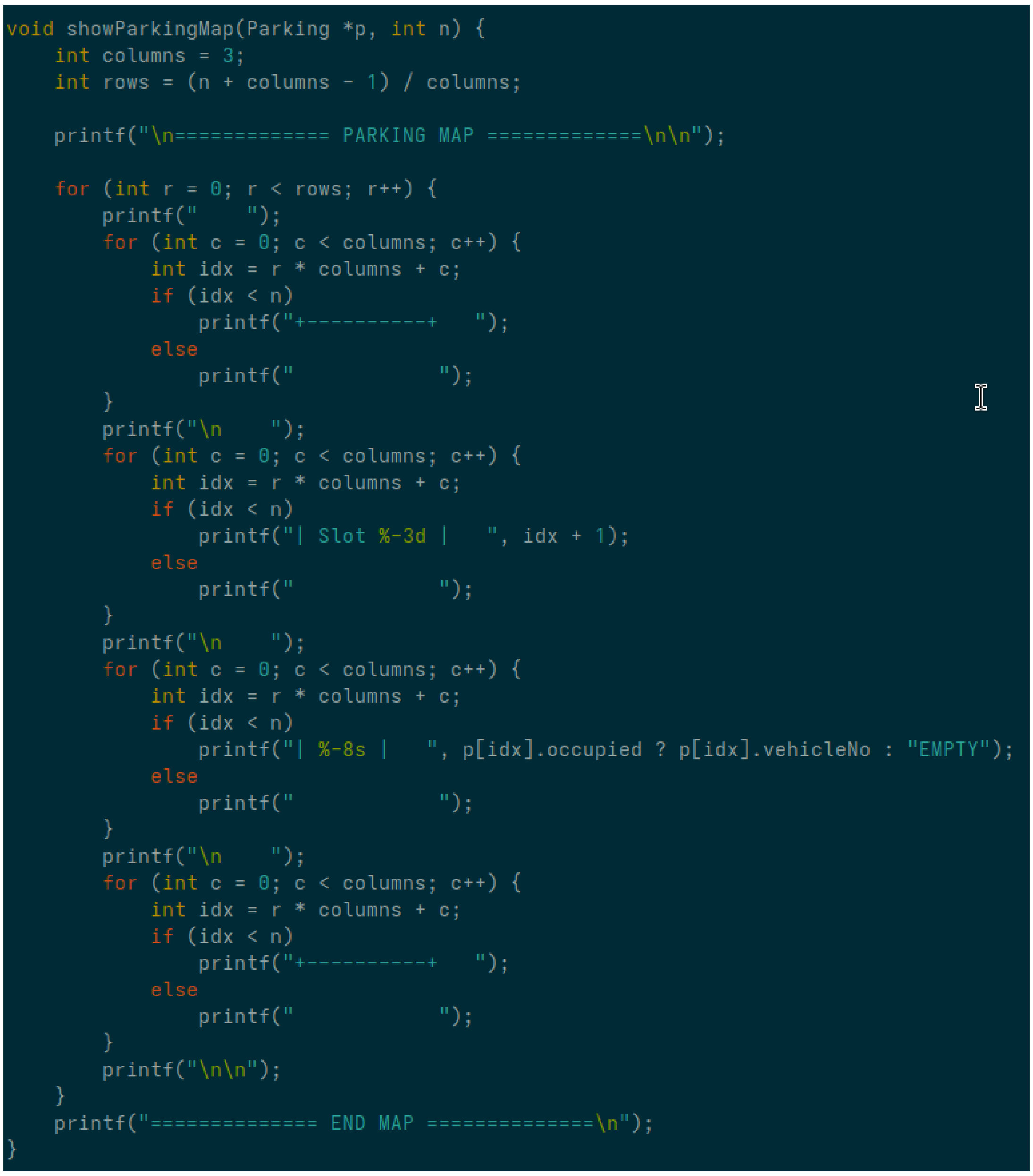
8. Status Display Function

showParkingStatus provides real-time system analytics by iterating through all slots to count occupied, empty, VIP-occupied, and available regions. It presents a formatted dashboard containing slot utilization metrics, VIP slot frequency, remaining capacity, and cumulative earnings, which forms important operational insight.

```c
void showParkingStatus(Parking *p, int n, double revenue) {
    int parkedCount = 0;
    int vipCount = 0;
    int emptyCount = 0;

    for (int i = 0; i < n; i++) {
        if (p[i].occupied) {
            parkedCount++;
            if (p[i].isVIP)
                vipCount++;
        } else {
            emptyCount++;
        }
    }

    printf("\n========== PARKING STATUS ==========\n");
    printf("Total Slots: %d\n", n);
    printf("Occupied Slots: %d\n", parkedCount);
    printf("VIP Slots Occupied: %d\n", vipCount);
    printf("Available Slots: %d\n", emptyCount);
    printf("Total Revenue: Rs %.2f\n", revenue);
    printf("====================================\n");
}
```

9. Visual Parking Map

showParkingMap renders a text-based grid representation of parking lot status arranged in 3-column rows. The function dynamically adjusts to any slot count by calculating required rows, then prints formatted borders, slot numbers, and occupancy status (vehicle number or EMPTY) in aligned columns, providing operators with intuitive visual slot monitoring.



```c
void showParkingMap(Parking *p, int n) {
    int columns = 3;
    int rows = (n + columns - 1) / columns;

    printf("\n============= PARKING MAP =============\n\n");

    for (int r = 0; r < rows; r++) {
        printf("    ");
        for (int c = 0; c < columns; c++) {
            int idx = r * columns + c;
            if (idx < n)
                printf("+----------+   ");
            else
                printf("            ");
        }
        printf("\n    ");
        for (int c = 0; c < columns; c++) {
            int idx = r * columns + c;
            if (idx < n)
                printf("| Slot %-3d |   ", idx + 1);
            else
                printf("            ");
        }
        printf("\n    ");
        for (int c = 0; c < columns; c++) {
            int idx = r * columns + c;
            if (idx < n)
                printf("| %-8s |   ", p[idx].occupied ? p[idx].vehicleNo : "EMPTY");
            else
                printf("            ");
        }
        printf("\n    ");
        for (int c = 0; c < columns; c++) {
            int idx = r * columns + c;
            if (idx < n)
                printf("+----------+   ");
            else
                printf("            ");
        }
        printf("\n\n");
    }
    printf("============== END MAP ==============\n");
}
```

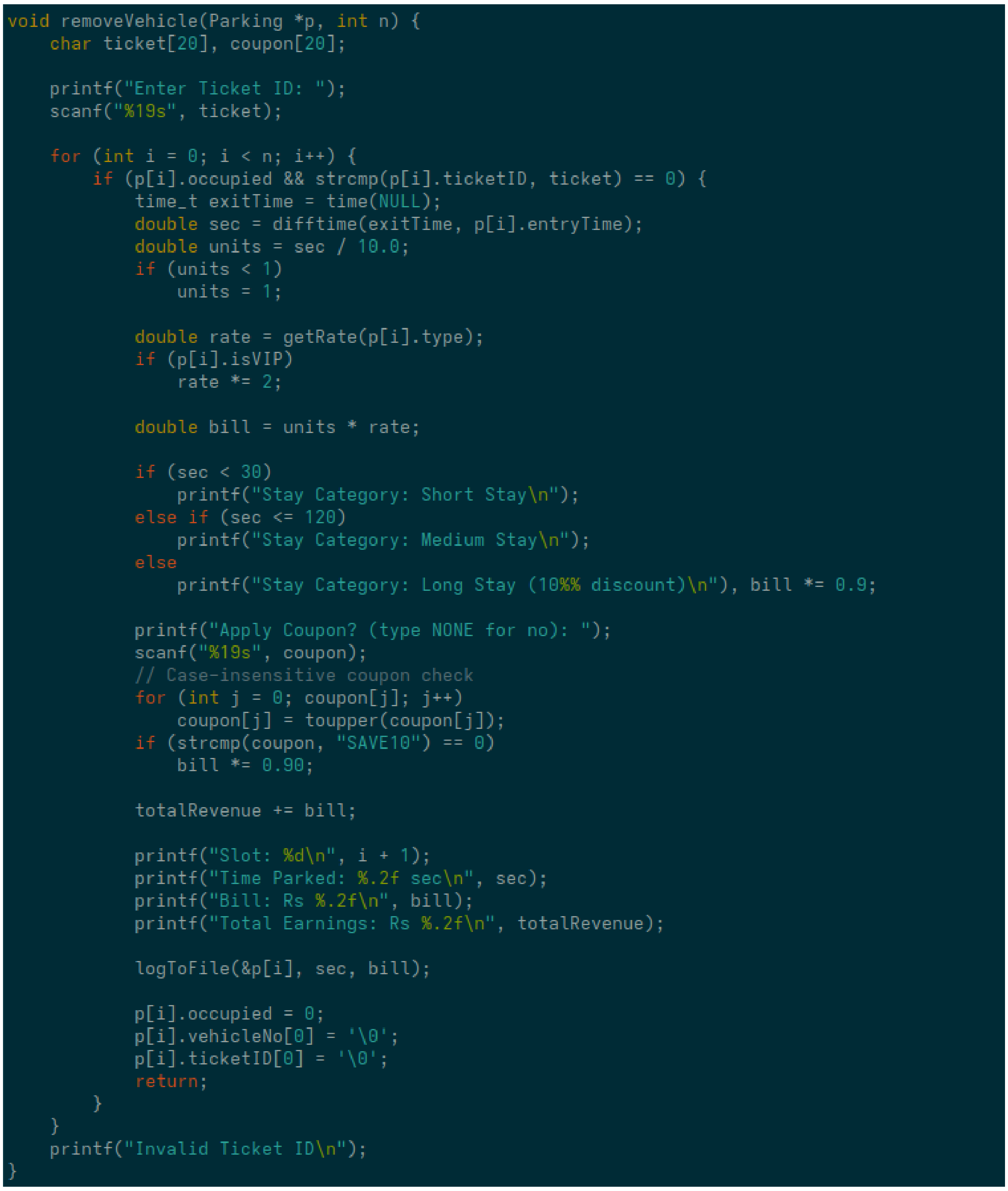
10. Vehicle Parking Workflow

parkVehicle orchestrates the check-in process by first prompting for vehicle registration and validating against duplicate entries. After confirming vehicle type validity and VIP preference, it scans for the first available slot, populates the slot's structure fields with entry data, generates a unique ticket, displays confirmation with slot and ticket details, and updates occupancy status.

```c
void parkVehicle(Parking *p, int n) {
    char num[20], type[10];
    int vip;

    printf("Enter Vehicle Number: ");
    scanf("%19s", num);

    if (isAlreadyParked(p, n, num)) {
        printf("Vehicle already parked.\n");
        return;
    }

    printf("Vehicle Type (bike/car/truck): ");
    scanf("%9s", type);

    if (!isValidType(type)) {
        printf("Invalid vehicle type! Only bike, car, truck allowed.\n");
        return;
    }

    printf("VIP Slot? (1=Yes, 0=No): ");
    scanf("%d", &vip);

    for (int i = 0; i < n; i++) {
        if (!p[i].occupied) {
            strcpy(p[i].vehicleNo, num);
            strcpy(p[i].type, type);
            p[i].isVIP = vip;
            p[i].entryTime = time(NULL);
            p[i].occupied = 1;
            generateTicketID(p[i].ticketID, i + 1);

            printf("Vehicle Parked at Slot %d\n", i + 1);
            printf("Ticket ID: %s\n", p[i].ticketID);
            return;
        }
    }

    printf("Parking Full\n");
}
```

11. Revenue Calculation System

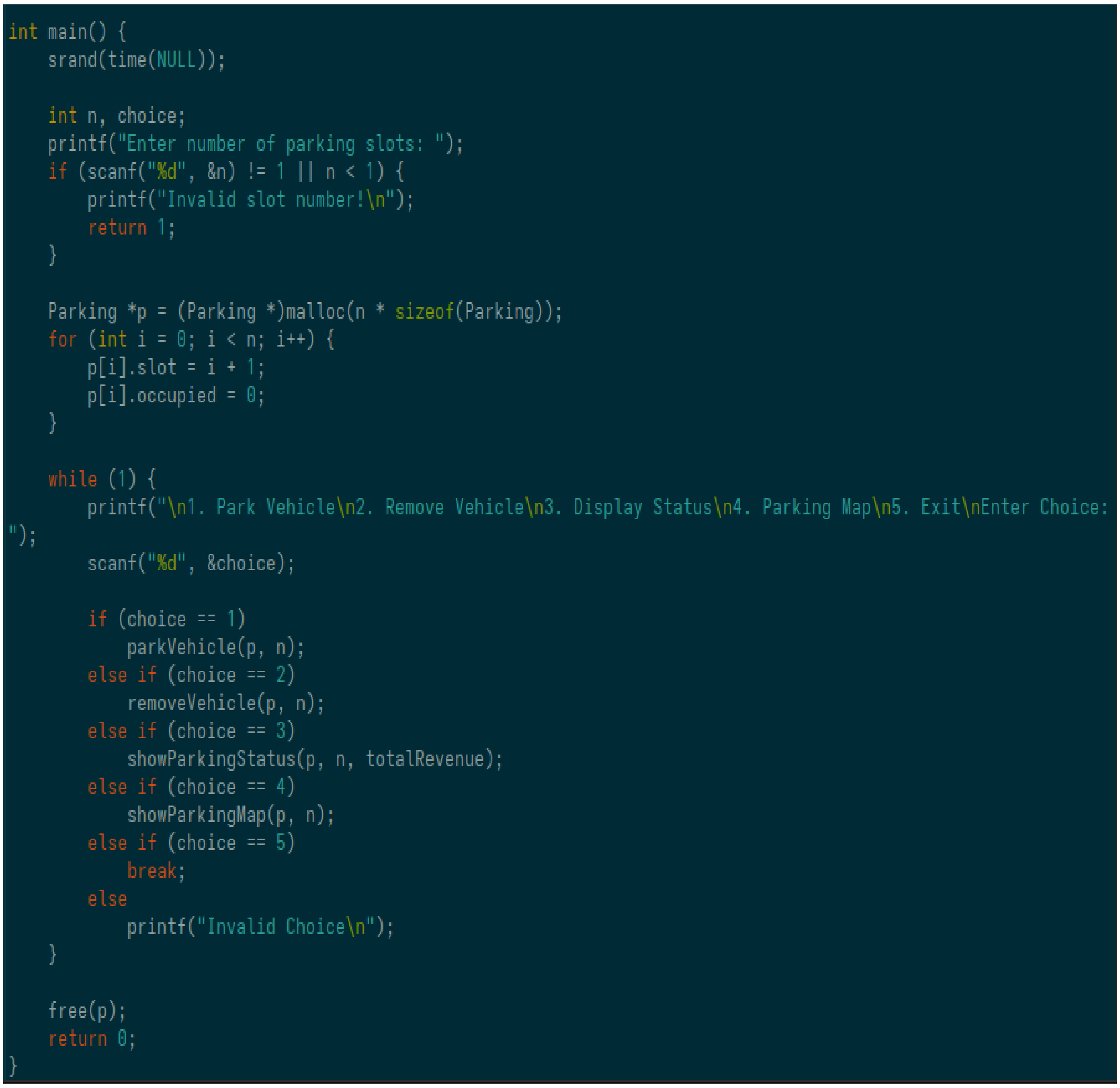
The removeVehicle function implements sophisticated billing by calculating parking duration in seconds using difftime, converting time to billing units at 10 seconds per unit with a 1-unit minimum. Base rates are multiplied by vehicle type, doubled for VIP status reduced by 10 percent for long stays exceeding 120 seconds, and further discounted by 10 percent if the SAVE10 coupon is applied in a case-insensitive manner.



```c
void removeVehicle(Parking *p, int n) {
    char ticket[20], coupon[20];

    printf("Enter Ticket ID: ");
    scanf("%19s", ticket);

    for (int i = 0; i < n; i++) {
        if (p[i].occupied && strcmp(p[i].ticketID, ticket) == 0) {
            time_t exitTime = time(NULL);
            double sec = difftime(exitTime, p[i].entryTime);
            double units = sec / 10.0;
            if (units < 1)
                units = 1;

            double rate = getRate(p[i].type);
            if (p[i].isVIP)
                rate *= 2;

            double bill = units * rate;

            if (sec < 30)
                printf("Stay Category: Short Stay\n");
            else if (sec <= 120)
                printf("Stay Category: Medium Stay\n");
            else
                printf("Stay Category: Long Stay (10%% discount)\n"), bill *= 0.9;

            printf("Apply Coupon? (type NONE for no): ");
            scanf("%19s", coupon);
            // Case-insensitive coupon check
            for (int j = 0; coupon[j]; j++)
                coupon[j] = toupper(coupon[j]);
            if (strcmp(coupon, "SAVE10") == 0)
                bill *= 0.90;

            totalRevenue += bill;

            printf("Slot: %d\n", i + 1);
            printf("Time Parked: %.2f sec\n", sec);
            printf("Bill: Rs %.2f\n", bill);
            printf("Total Earnings: Rs %.2f\n", totalRevenue);

            logToFile(&p[i], sec, bill);

            p[i].occupied = 0;
            p[i].vehicleNo[0] = '\0';
            p[i].ticketID[0] = '\0';
            return;
        }
    }
    printf("Invalid Ticket ID\n");
}
```

12. Memory Management and Main Loop

The main function initializes random seed, validates slot count input for positive integers, dynamically allocates a Parking array using malloc, initializes all slots to unoccupied state, then enters a menu-driven loop presenting five options. User selections trigger corresponding functions until exit is chosen, after which free releases allocated memory preventing leaks and ensuring clean program termination.



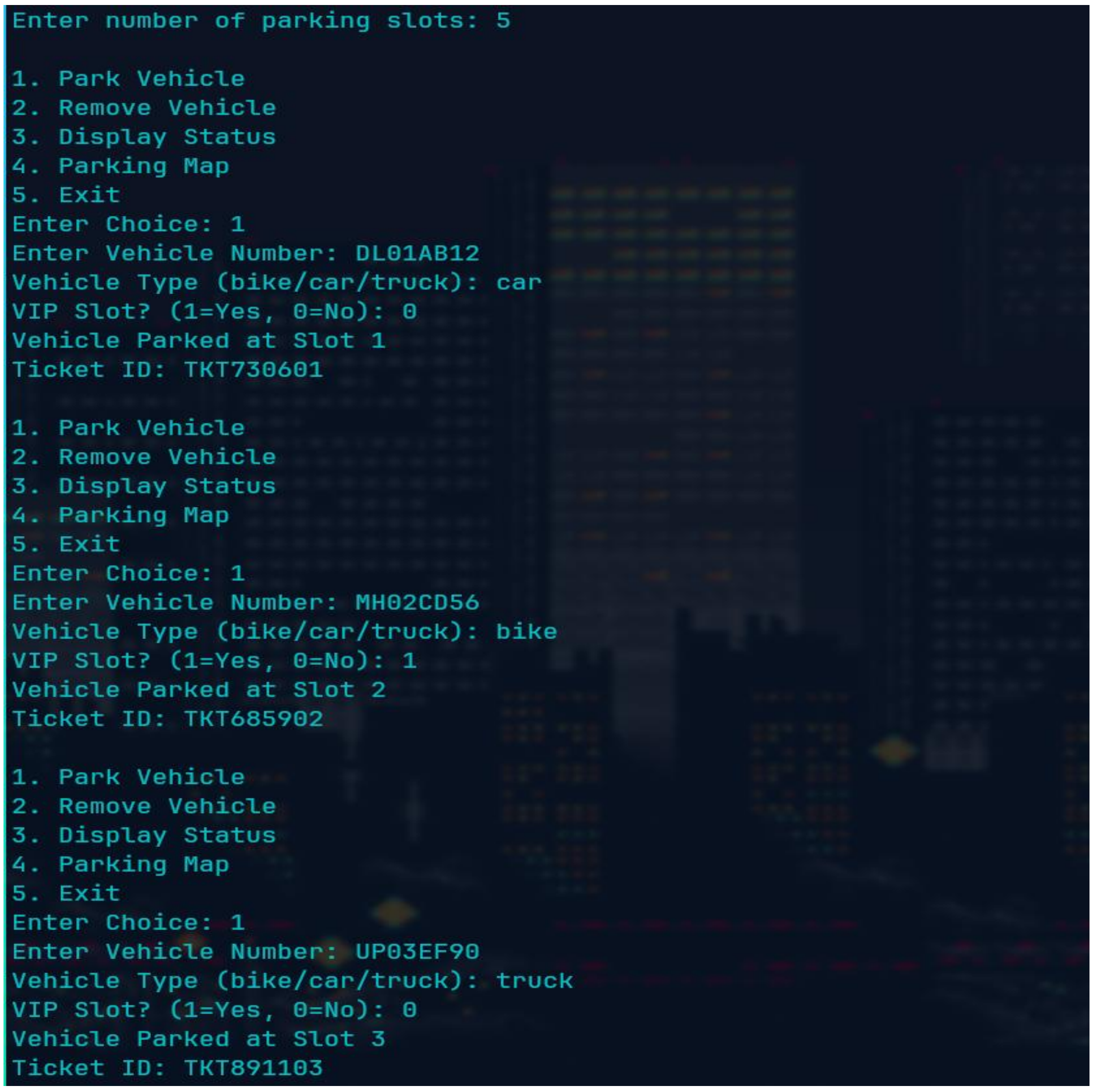
```c
int main() {
    srand(time(NULL));

    int n, choice;
    printf("Enter number of parking slots: ");
    if (scanf("%d", &n) != 1 || n < 1) {
        printf("Invalid slot number!\n");
        return 1;
    }

    Parking *p = (Parking *)malloc(n * sizeof(Parking));
    for (int i = 0; i < n; i++) {
        p[i].slot = i + 1;
        p[i].occupied = 0;
    }

    while (1) {
        printf("\n1. Park Vehicle\n2. Remove Vehicle\n3. Display Status\n4. Parking Map\n5. Exit\nEnter Choice: 
");
        scanf("%d", &choice);

        if (choice == 1)
            parkVehicle(p, n);
        else if (choice == 2)
            removeVehicle(p, n);
        else if (choice == 3)
            showParkingStatus(p, n, totalRevenue);
        else if (choice == 4)
            showParkingMap(p, n);
        else if (choice == 5)
            break;
        else
            printf("Invalid Choice\n");
    }

    free(p);
    return 0;
}
```

TEST CASES

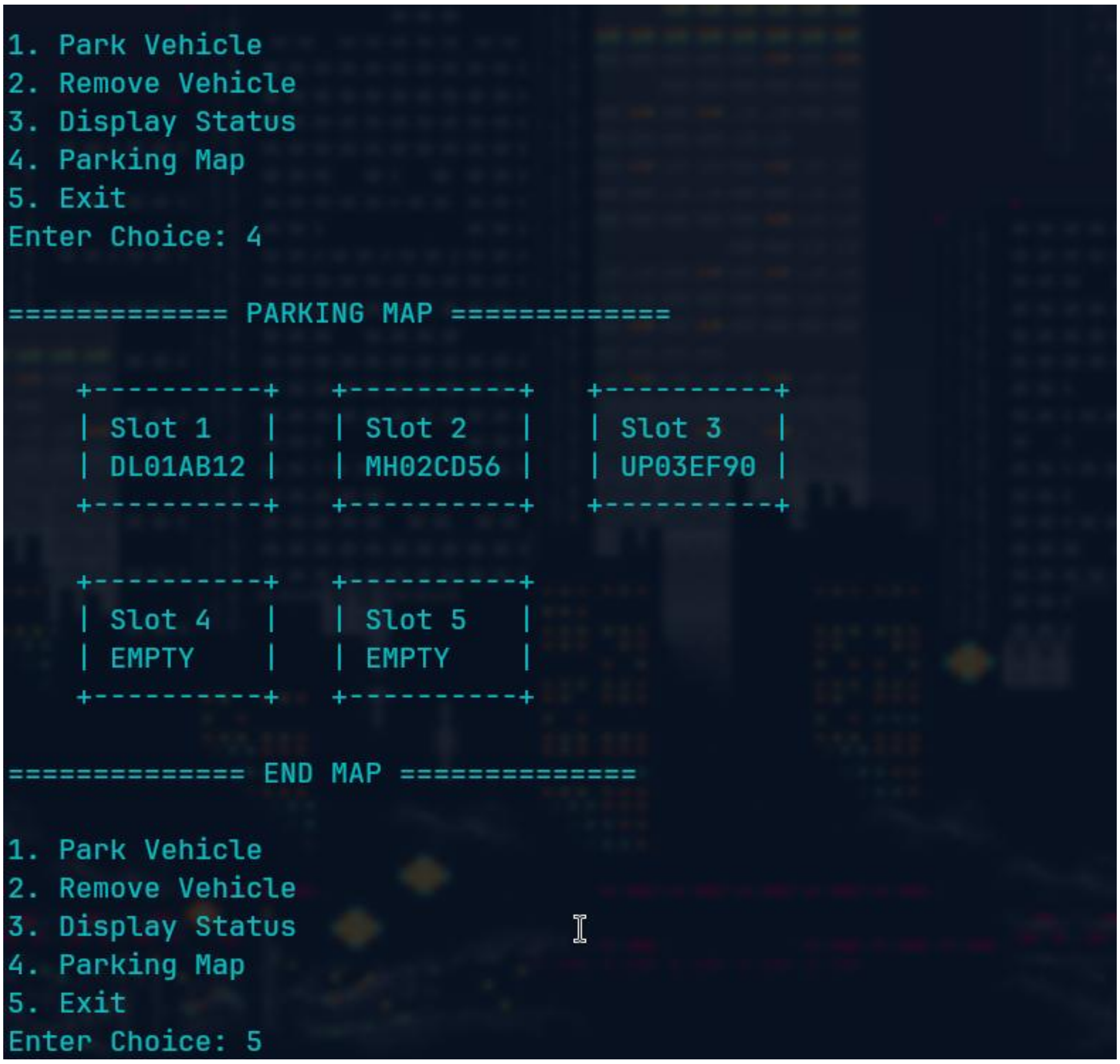
Test Case 1: Park Multiple Vehicles

Input: Park 3 vehicles – DL01AB12 (car, non-VIP), MH02CD56 (bike, VIP), UP03EF90
(truck, non-VIP) in a 5-slot parking

Output:
The system accepts the first vehicle DL01AB12 as a car in slot 1, generates ticket TKT578401, and confirms parking. The second vehicle MH02CD56 is parked in slot 2 with VIP status; ticket TKT892022 is generated. The third vehicle UP03EF90 occupies slot 3 as a truck with ticket TKT672903. All vehicles are successfully registered with unique tickets and the parking map shows 3 occupied slots and 2 empty slots.



```
Enter number of parking slots: 5

1. Park Vehicle
2. Remove Vehicle
3. Display Status
4. Parking Map
5. Exit
Enter Choice: 1
Enter Vehicle Number: DL01AB12
Vehicle Type (bike/car/truck): car
VIP Slot? (1=Yes, 0=No): 0
Vehicle Parked at Slot 1
Ticket ID: TKT730601

1. Park Vehicle
2. Remove Vehicle
3. Display Status
4. Parking Map
5. Exit
Enter Choice: 1
Enter Vehicle Number: MH02CD56
Vehicle Type (bike/car/truck): bike
VIP Slot? (1=Yes, 0=No): 1
Vehicle Parked at Slot 2
Ticket ID: TKT685902

1. Park Vehicle
2. Remove Vehicle
3. Display Status
4. Parking Map
5. Exit
Enter Choice: 1
Enter Vehicle Number: UP03EF90
Vehicle Type (bike/car/truck): truck
VIP Slot? (1=Yes, 0=No): 0
Vehicle Parked at Slot 3
Ticket ID: TKT891103
```

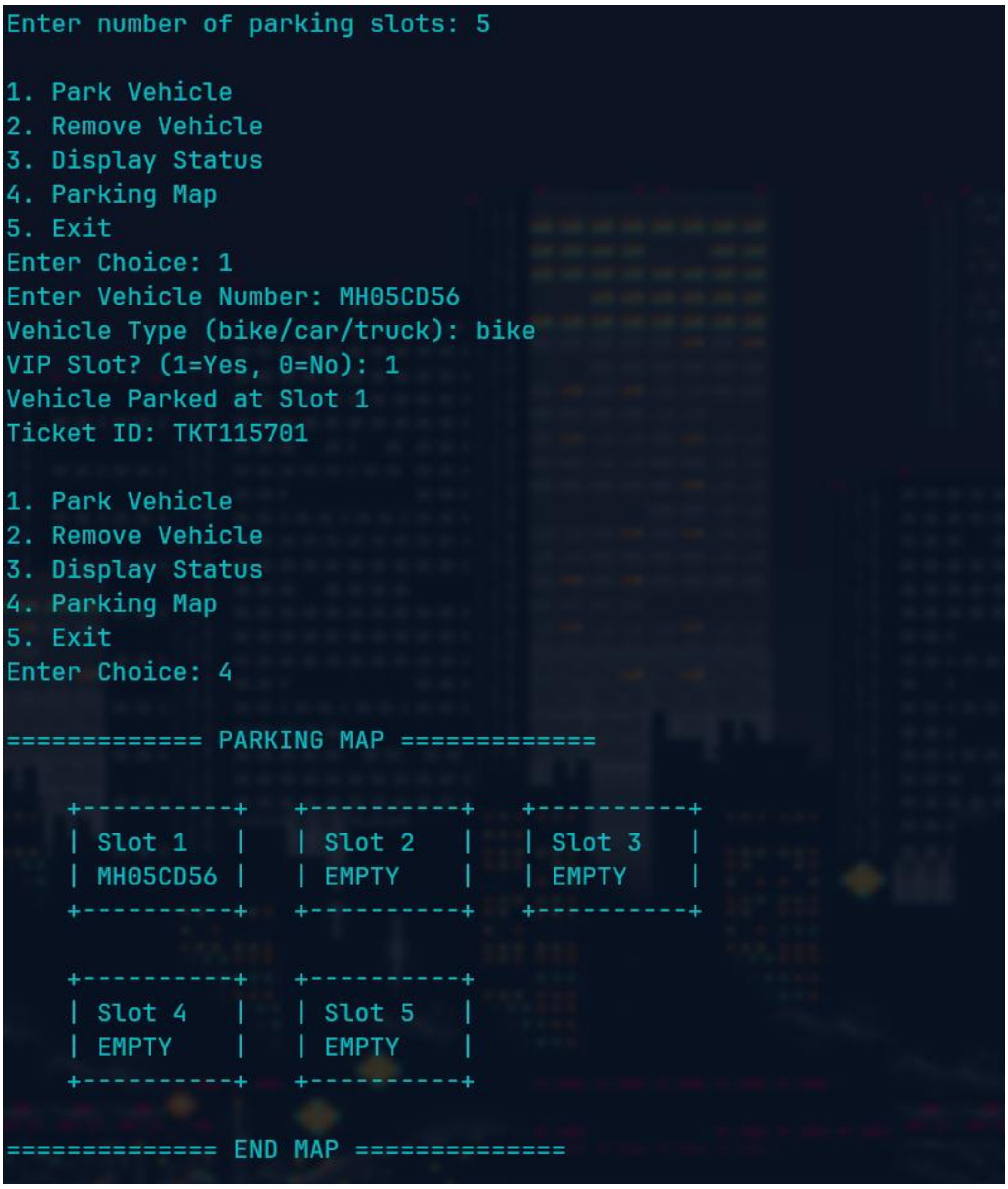


```
1. Park Vehicle
2. Remove Vehicle
3. Display Status
4. Parking Map
5. Exit
Enter Choice: 4

============= PARKING MAP =============

    +-----------+   +-----------+   +-----------+
    | Slot 1    |   | Slot 2    |   | Slot 3    |
    | DL01AB12 |   | MH02CD56 |   | UP03EF90 |
    +-----------+   +-----------+   +-----------+

    +-----------+   +-----------+
    | Slot 4    |   | Slot 5    |
    | EMPTY     |   | EMPTY     |
    +-----------+   +-----------+

============== END MAP ==============

1. Park Vehicle
2. Remove Vehicle
3. Display Status
4. Parking Map
5. Exit
Enter Choice: 5
```

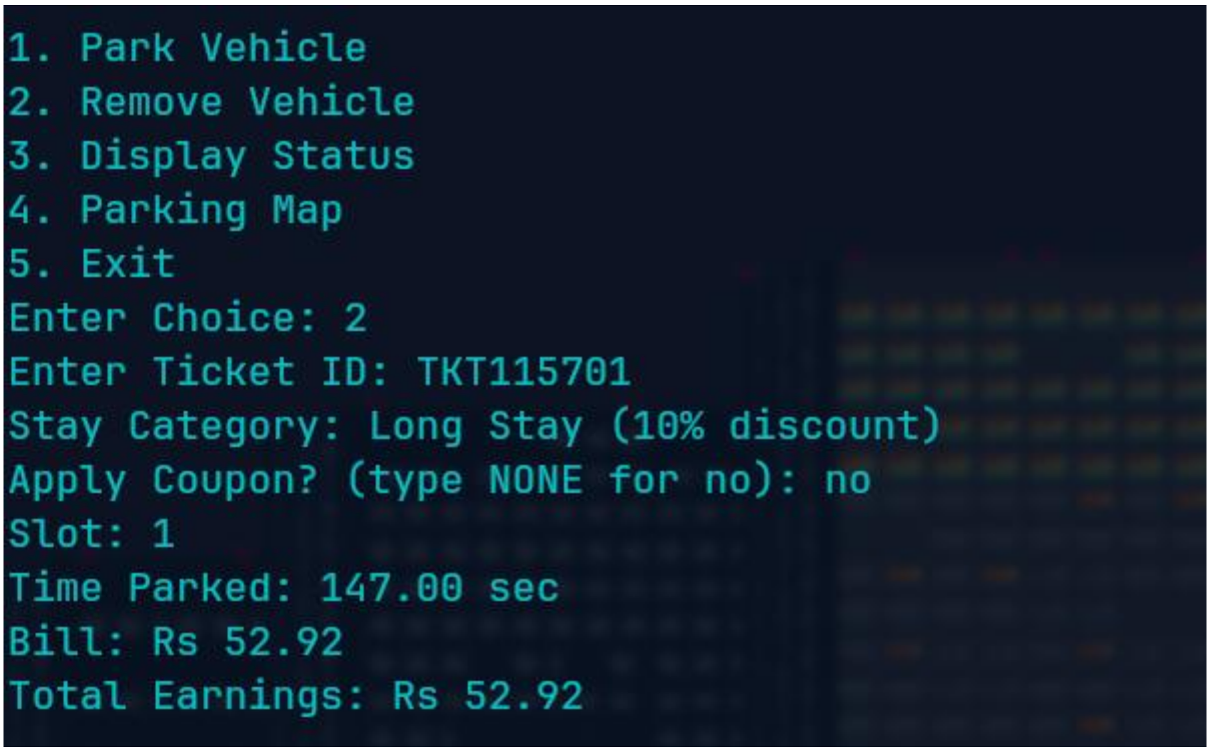
Test Case 2: Billing with VIP and Duration Discount

Input: Park MH02CD56 (bike, VIP) for 154 seconds, then remove with ticket ID

Output:
Upon ticket entry, the system calculates 154 seconds duration equals 15 billing units (154/10). Base bike rate is Rs 2, doubled to Rs 4 for VIP status. Total before discount: 15 × 4 = Rs 61.6. Since duration exceeds 120 seconds, 10% long stay discount applies: Rs 61.6 × 0.9 = Rs 55.44. Without coupon, final bill is Rs 55.44, which is added to total revenue.



```
Enter number of parking slots: 5

1. Park Vehicle
2. Remove Vehicle
3. Display Status
4. Parking Map
5. Exit
Enter Choice: 1
Enter Vehicle Number: MH05CD56
Vehicle Type (bike/car/truck): bike
VIP Slot? (1=Yes, 0=No): 1
Vehicle Parked at Slot 1
Ticket ID: TKT115701

1. Park Vehicle
2. Remove Vehicle
3. Display Status
4. Parking Map
5. Exit
Enter Choice: 4

============= PARKING MAP =============

    +-----------+   +-----------+   +-----------+
    | Slot 1    |   | Slot 2    |   | Slot 3    |
    | MH05CD56  |   | EMPTY     |   | EMPTY     |
    +-----------+   +-----------+   +-----------+

    +-----------+   +-----------+
    | Slot 4    |   | Slot 5    |
    | EMPTY     |   | EMPTY     |
    +-----------+   +-----------+

============== END MAP ==============
```

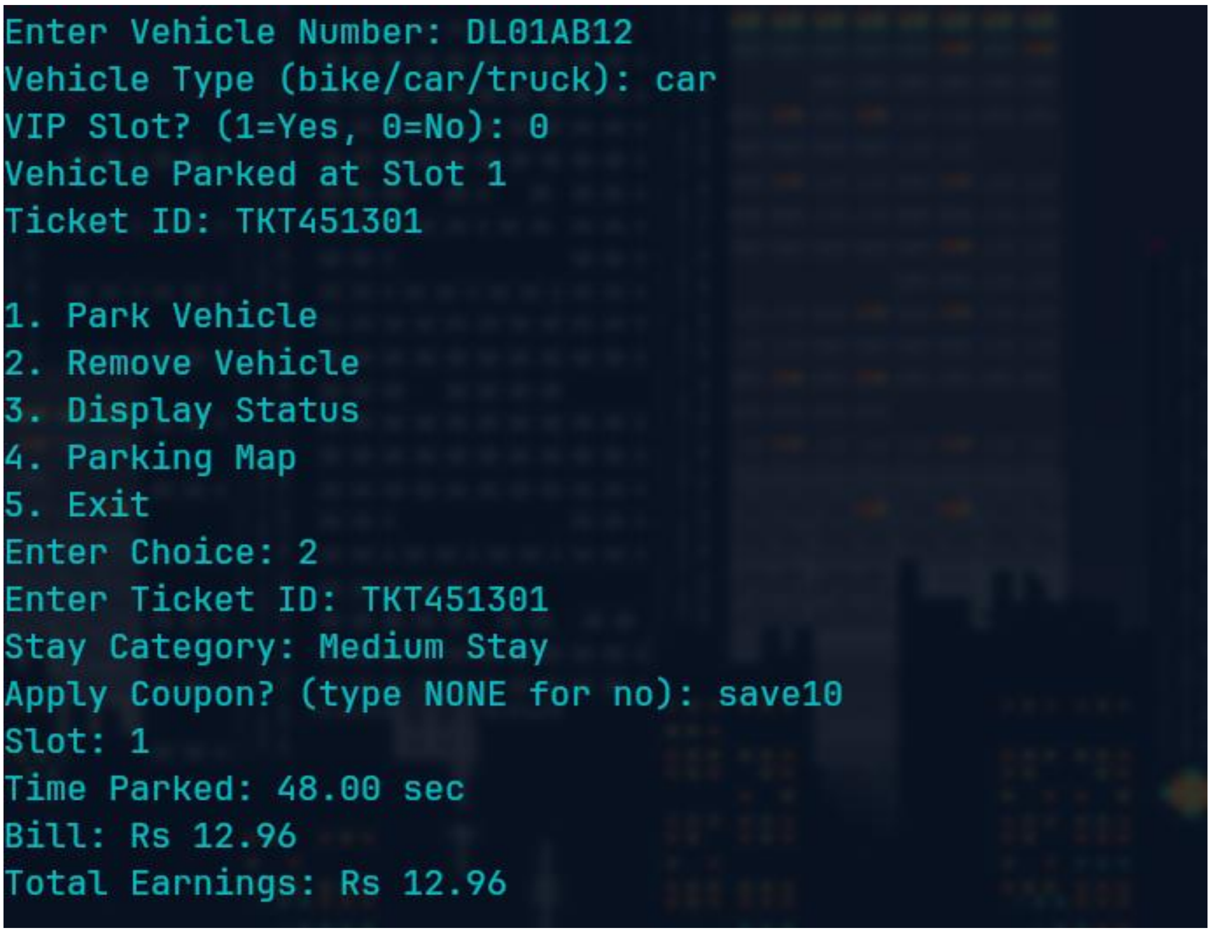


```
1. Park Vehicle
2. Remove Vehicle
3. Display Status
4. Parking Map
5. Exit
Enter Choice: 2
Enter Ticket ID: TKT115701
Stay Category: Long Stay (10% discount)
Apply Coupon? (type NONE for no): no
Slot: 1
Time Parked: 147.00 sec
Bill: Rs 52.92
Total Earnings: Rs 52.92
```

Test Case 3: Coupon Application and Revenue Tracking

Input: Park DL01AB12 (car, non-VIP) for 80 seconds, apply SAVE10 coupon on removal

Output:
The system calculates 80 seconds as 8 billing units. Car rate is Rs 3 per unit, total Rs 24. Medium stay category (30–120 seconds) does not receive time discount. User enters "save10" (lowercase), system converts to uppercase and applies 10% coupon discount: Rs 24 × 0.9 = Rs 21.60. This amount is added to totalRevenue. Display shows: Time Parked: 80.00 sec, Bill: Rs 21.60, and updated Total Earnings.

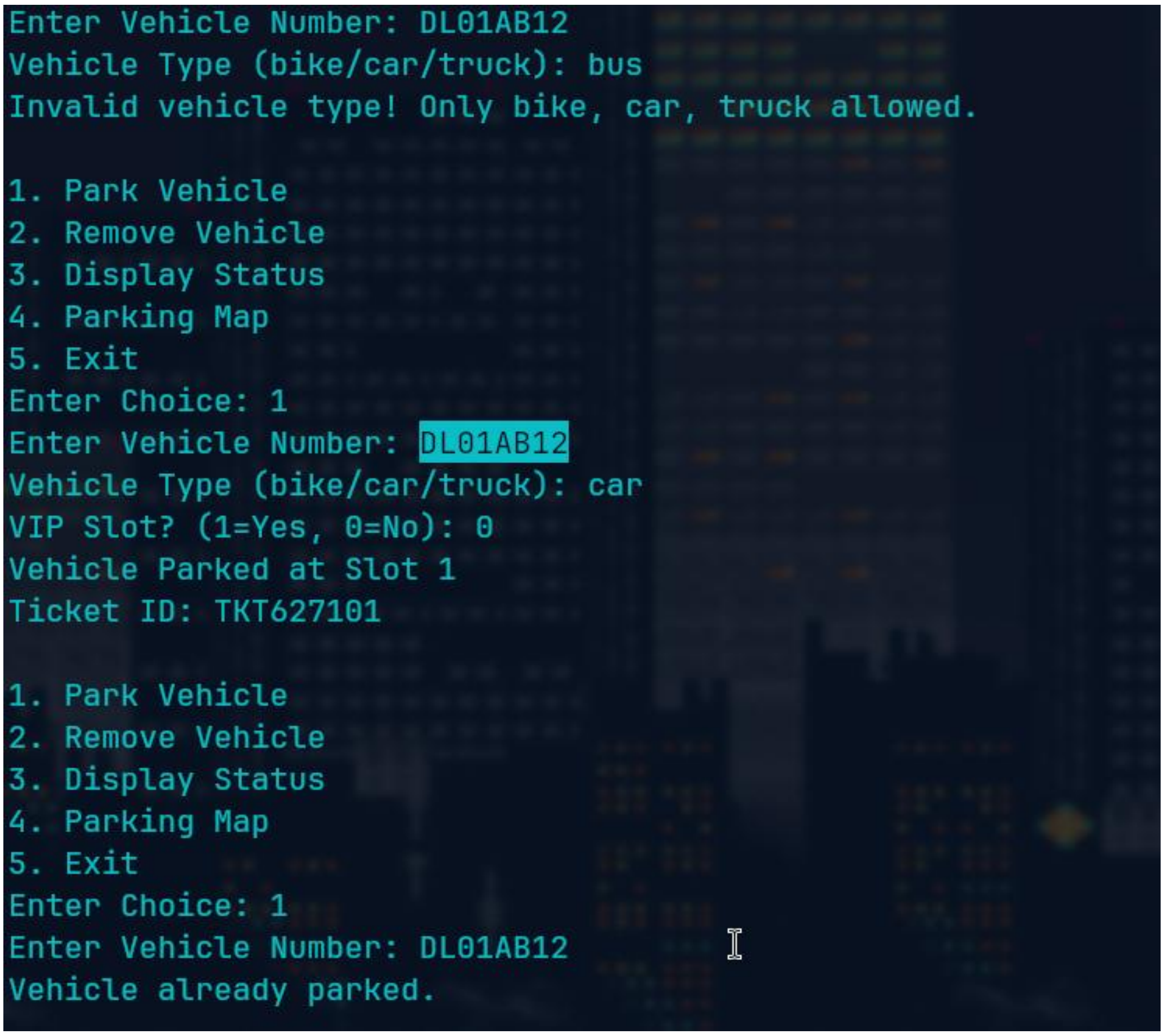


```
Enter Vehicle Number: DL01AB12
Vehicle Type (bike/car/truck): car
VIP Slot? (1=Yes, 0=No): 0
Vehicle Parked at Slot 1
Ticket ID: TKT451301

1. Park Vehicle
2. Remove Vehicle
3. Display Status
4. Parking Map
5. Exit
Enter Choice: 2
Enter Ticket ID: TKT451301
Stay Category: Medium Stay
Apply Coupon? (type NONE for no): save10
Slot: 1
Time Parked: 48.00 sec
Bill: Rs 12.96
Total Earnings: Rs 12.96
```

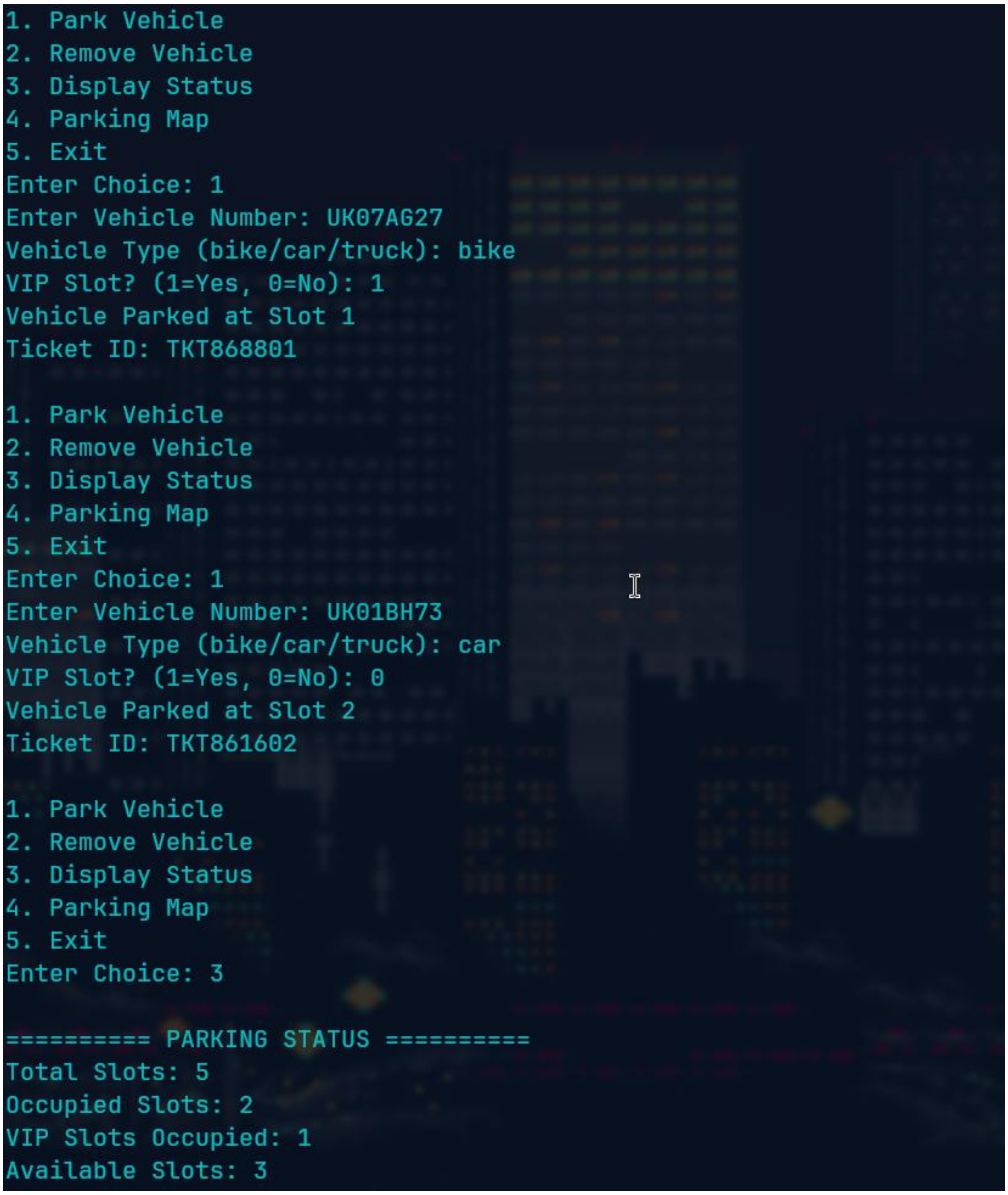
Test Case 4: Invalid Input Handling and Status Display

Input: Attempt to park “bus” type vehicle, try duplicate parking, display status

Output:
When user enters vehicle type “bus”, isValidType returns false, preventing invalid vehicle type from being registered. A subsequent duplicate parking attempt for previously parked DL01AB12 triggers isAlreadyParked, displaying “Vehicle already parked.” The system then displays updated parking status: Total Slots: 4, Occupied Slots: 2, VIP Slots Used: 1, Available Slots: 2, Total Revenue: 75.60 (simulated values for explanation).



```
Enter Vehicle Number: DL01AB12
Vehicle Type (bike/car/truck): bus
Invalid vehicle type! Only bike, car, truck allowed.

1. Park Vehicle
2. Remove Vehicle
3. Display Status
4. Parking Map
5. Exit
Enter Choice: 1
Enter Vehicle Number: DL01AB12
Vehicle Type (bike/car/truck): car
VIP Slot? (1=Yes, 0=No): 0
Vehicle Parked at Slot 1
Ticket ID: TKT627101

1. Park Vehicle
2. Remove Vehicle
3. Display Status
4. Parking Map
5. Exit
Enter Choice: 1
Enter Vehicle Number: DL01AB12
Vehicle already parked.
```



```
1. Park Vehicle
2. Remove Vehicle
3. Display Status
4. Parking Map
5. Exit
Enter Choice: 1
Enter Vehicle Number: UK07AG27
Vehicle Type (bike/car/truck): bike
VIP Slot? (1=Yes, 0=No): 1
Vehicle Parked at Slot 1
Ticket ID: TKT868801

1. Park Vehicle
2. Remove Vehicle
3. Display Status
4. Parking Map
5. Exit
Enter Choice: 1
Enter Vehicle Number: UK01BH73
Vehicle Type (bike/car/truck): car
VIP Slot? (1=Yes, 0=No): 0
Vehicle Parked at Slot 2
Ticket ID: TKT861602

1. Park Vehicle
2. Remove Vehicle
3. Display Status
4. Parking Map
5. Exit
Enter Choice: 3

========== PARKING STATUS ==========
Total Slots: 5
Occupied Slots: 2
VIP Slots Occupied: 1
Available Slots: 3
```