COURSE: Bachelors in Computer Science

Submitted By:

Sanjana Yadav 590028472

Yugraja Samadhiya 590028470

Submitted To: Mr. Vinod Kumar

Date: November 30, 2025

C programming major project

TOPICS – Parking management

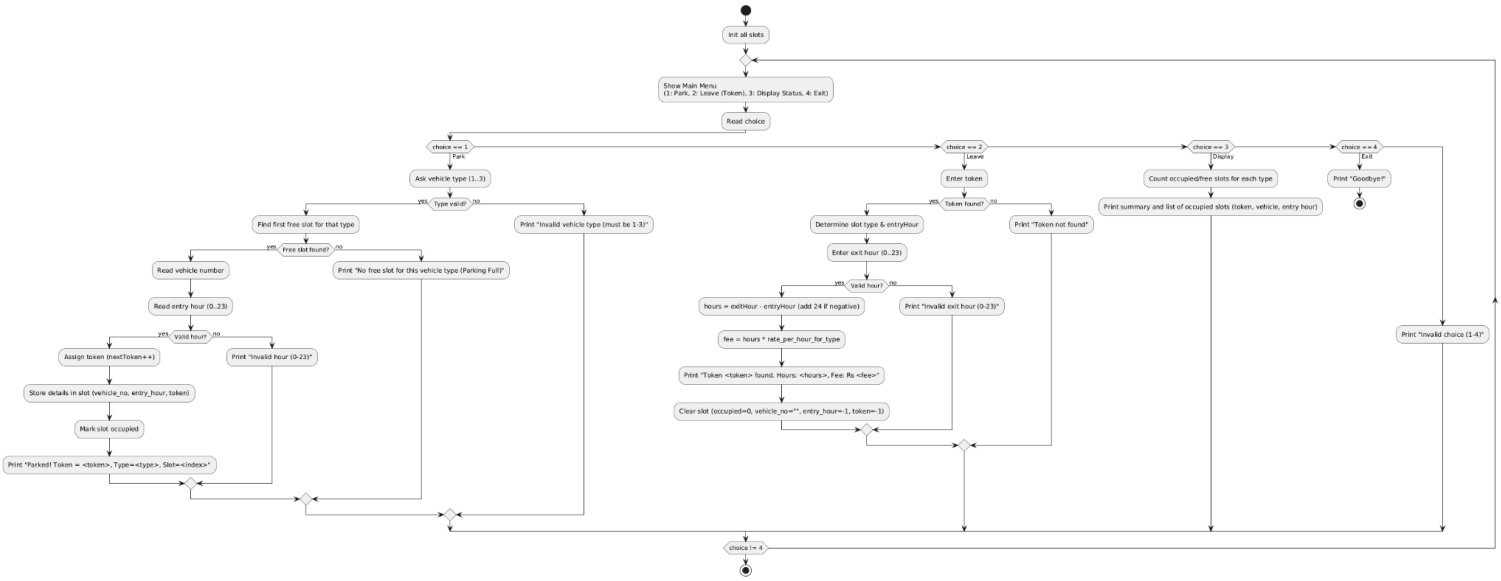
PROBLEM DEFINATION

The problem is that traditional parking areas are managed manually, leading to confusion, inaccurate record-keeping, difficulty in tracking available slots, and delays in vehicle entry and exit. There is a need for an automated system that can efficiently manage parking spaces, maintain real-time records, and reduce human errors.

ALGORITHM

1. START
2. Initialize all parking slots
   Set every slot's status to empty
   Clear vehicle number, entry hour, and token for all big, small, and two-wheeler slots
3. Set token counter to 1000
4. Display main menu
   Park Vehicle
   Leave (Using Token)
   Display Status
   Exit
5. If user chooses "Park Vehicle"
   Ask for vehicle type
   Search for the first free slot of that type
   If no free slot → show "Parking Full"
   Else collect vehicle number and entry hour
   Generate a new token
   Store details in the selected slot
   Mark slot as occupied
   Display token and slot number
6. If user chooses "Leave Using Token"
   Ask for token number
   Search all slots to find the matching token
   If token not found → show error
   Ask for exit hour
   Calculate total parked hours
   Calculate parking fee based on vehicle type
   Display hours and fee
   Clear the slot and mark it as free

7. If user chooses “Display Status”
   Count and show total free and occupied slots of each type
   Display details of all currently parked vehicles
8. If user chooses “Exit”
   Stop the program
9. END



```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>

#define BIG_CAP 50
#define SMALL_CAP 100
#define TWO_CAP 150

struct Slot {
    int occupied;
    char vehicle_no[30];
    int entry_hour;
    int token;
};

// YE SAB GLOBAL ARRAY HAI {BIG FOUR WHEELER }, {SMALL FOUR WHEELER },
{TWO WHEELER}
struct Slot bigSlots[BIG_CAP];
struct Slot smallSlots[SMALL_CAP];
struct Slot twoSlots[TWO_CAP];

//  YE GLOBAL TOKEN COUNTER HAI
int nextToken = 1000;

// YE SAARE FUNCTIONS HAI
void initAllSlots(void);   // YE INITIAL SLOTS  HAI
int findFreeSlotForType(struct Slot slots[], int cap); // YE FUNCTION
FREE SLOT FIND KAREGA USS PARTICULAR VECHILE K LIYE
int parkVehicle(void); // YE FUNCTION PARKING KAREGA
int findSlotByToken(int token, int *typeOut, int *indexOut); // YE FUN.
TOKEN KI HELP SE SLOT FIND KAREGA
void leaveByToken(void); // YE FUN.
void displayStatus(void);  // YE FUN. STATUS SHOW KAREGA
void clearInputLine(void);

// Prices per hour
const int PRICE_BIG = 60;   // BIG CARS
const int PRICE_SMALL = 40; // SMALL CARS
const int PRICE_TWO = 20;   // TWO WHEELER

// YE HAI  MAIN FUNCTION
```

```c
int main(void) {
    initAllSlots();
// YE LINES SABSE PHELE SHOW HONGI
    while (1) {
        int choice;
        printf("\n=== PARKING SYSTEM (Tokens) ===\n");
        printf("1. Park vehicle\n");
        printf("2. Leave (use token)\n");
        printf("3. Display status\n");
        printf("4. Exit\n");
        printf("Enter choice: ");

        if (scanf("%d", &choice) != 1) {
            printf("Invalid input. Try again.\n");
            clearInputLine();
            continue;
        }

        if (choice == 1) {
            parkVehicle();
        } else if (choice == 2) {
            leaveByToken();
        } else if (choice == 3) {
            displayStatus();
        } else if (choice == 4) {
            printf("Goodbye!\n");
            break;
        } else {
            printf("Invalid choice. 1-4 only.\n");
        }
    }

    return 0;
}         // YAHA MAIN FUNCTION END HO GAYA

// YAHA PE SAARE SLOTS INITIALIZE HONE KA FUNCTION HAI
void initAllSlots(void) {
    // YE BIG CARS K LIYE SLOTS HAI
    for (int i = 0; i < BIG_CAP; i++) {
        bigSlots[i].occupied = 0;
        bigSlots[i].vehicle_no[0] = '\0';
        bigSlots[i].entry_hour = -1;
        bigSlots[i].token = -1;
```

```
    }
// YE SMALL CARS K LIYE SLOTS HAI
    for (int i = 0; i < SMALL_CAP; i++) {
        smallSlots[i].occupied = 0;
        smallSlots[i].vehicle_no[0] = '\0';
        smallSlots[i].entry_hour = -1;
        smallSlots[i].token = -1;
    }
// YE 2 WHEELERS K LIYE SLOTS HAI
    for (int i = 0; i < TWO_CAP; i++) {
        twoSlots[i].occupied = 0;
        twoSlots[i].vehicle_no[0] = '\0';
        twoSlots[i].entry_hour = -1;
        twoSlots[i].token = -1;
    }
}

// YE FREE SLOT DHOONDTA HAI ARRAY MAI
int findFreeSlotForType(struct Slot slots[], int cap) {
    for (int i = 0; i < cap; i++) {
        if (slots[i].occupied == 0) return i;
    }
    return -1;
}

//  Park vehicle: asks type, vehicle no, entry hour
int parkVehicle(void) {
    printf("\nChoose vehicle type:\n");
    printf("1. Big 4-wheeler (Rs 60/hr)\n");
    printf("2. Small 4-wheeler (Rs 40/hr)\n");
    printf("3. 2-wheeler (Rs 20/hr)\n");
    printf("Enter type (1-3): ");
    int type;
    if (scanf("%d", &type) != 1) {
        printf("Invalid type input.\n");
        clearInputLine();
        return -1;
    }

    int idx = -1;
    if (type == 1) idx = findFreeSlotForType(bigSlots, BIG_CAP);
    else if (type == 2) idx = findFreeSlotForType(smallSlots,
SMALL_CAP);
```

```
    else if (type == 3) idx = findFreeSlotForType(twoSlots, TWO_CAP);
    else {
        printf("Type must be 1-3.\n");
        return -1;
    }

    if (idx == -1) {
        printf("No free slot for this vehicle type.\n");
        return -1;
    }

    char vno[30];
    int hour;
    printf("Enter vehicle number: ");
    if (scanf("%29s", vno) != 1) {
        printf("Invalid vehicle number.\n");
        clearInputLine();
        return -1;
    }

    printf("Enter entry hour (0-23): ");
    if (scanf("%d", &hour) != 1 || hour < 0 || hour > 23) {
        printf("Invalid hour. Must be 0-23.\n");
        clearInputLine();
        return -1;
    }

    // assign into correct array and give token
    int myToken = ++nextToken;

    if (type == 1) {
        bigSlots[idx].occupied = 1;
        snprintf(bigSlots[idx].vehicle_no, sizeof
bigSlots[idx].vehicle_no, "%s", vno);
        bigSlots[idx].entry_hour = hour;
        bigSlots[idx].token = myToken;
    } else if (type == 2) {
        smallSlots[idx].occupied = 1;
        snprintf(smallSlots[idx].vehicle_no, sizeof
smallSlots[idx].vehicle_no, "%s", vno);
        smallSlots[idx].entry_hour = hour;
        smallSlots[idx].token = myToken;
    } else {
```

```
        twoSlots[idx].occupied = 1;
        snprintf(twoSlots[idx].vehicle_no, sizeof
twoSlots[idx].vehicle_no, "%s", vno);
        twoSlots[idx].entry_hour = hour;
        twoSlots[idx].token = myToken;
    }

    printf("Parked! Token = %d. Slot type=%d slotIndex=%d\n", myToken,
type, idx + 1);
    return myToken;
}

// Find slot by token across all arrays
// returns 0 if not found, else 1 and sets typeOut
(1=big,2=small,3=two) and indexOut
int findSlotByToken(int token, int *typeOut, int *indexOut) {
    if (token <= 0) return 0;
    for (int i = 0; i < BIG_CAP; i++) {
        if (bigSlots[i].occupied && bigSlots[i].token == token) {
            *typeOut = 1; *indexOut = i; return 1;
        }
    }
    for (int i = 0; i < SMALL_CAP; i++) {
        if (smallSlots[i].occupied && smallSlots[i].token == token) {
            *typeOut = 2; *indexOut = i; return 1;
        }
    }
    for (int i = 0; i < TWO_CAP; i++) {
        if (twoSlots[i].occupied && twoSlots[i].token == token) {
            *typeOut = 3; *indexOut = i; return 1;
        }
    }
    return 0;
}

// Leave by token
void leaveByToken(void) {
    int token;
    printf("Enter token: ");
    if (scanf("%d", &token) != 1) {
        printf("Invalid token input.\n");
        clearInputLine();
        return;
```

```
    }

    int type, index;
    if (!findSlotByToken(token, &type, &index)) {
        printf("Token not found.\n");
        return;
    }

    // calculate hours and fee
    int entryHour;
    int pricePerHour;
    if (type == 1) {
        entryHour = bigSlots[index].entry_hour;
        pricePerHour = PRICE_BIG;
    } else if (type == 2) {
        entryHour = smallSlots[index].entry_hour;
        pricePerHour = PRICE_SMALL;
    } else {
        entryHour = twoSlots[index].entry_hour;
        pricePerHour = PRICE_TWO;
    }

    int exitHour;
    printf("Enter exit hour (0-23): ");
    if (scanf("%d", &exitHour) != 1 || exitHour < 0 || exitHour > 23) {
        printf("Invalid hour.\n");
        clearInputLine();
        return;
    }

    int hours = exitHour - entryHour;
    if (hours < 0) hours += 24;
    int fee = hours * pricePerHour;

    printf("Token %d found in type %d slot %d.\n", token, type, index +
1);
    printf("Parked hours: %d. Rate: Rs %d/hr. Fee: Rs %d\n", hours,
pricePerHour, fee);

    // free the slot
    if (type == 1) {
        bigSlots[index].occupied = 0;
        bigSlots[index].vehicle_no[0] = '\0';
```

```
        bigSlots[index].entry_hour = -1;
        bigSlots[index].token = -1;
    } else if (type == 2) {
        smallSlots[index].occupied = 0;
        smallSlots[index].vehicle_no[0] = '\0';
        smallSlots[index].entry_hour = -1;
        smallSlots[index].token = -1;
    } else {
        twoSlots[index].occupied = 0;
        twoSlots[index].vehicle_no[0] = '\0';
        twoSlots[index].entry_hour = -1;
        twoSlots[index].token = -1;
    }
}

// Display counts and occupied slots
void displayStatus(void) {
    int bigFree = 0, bigOcc = 0;
    for (int i = 0; i < BIG_CAP; i++) {
        if (bigSlots[i].occupied) bigOcc++; else bigFree++;
    }

    int smallFree = 0, smallOcc = 0;
    for (int i = 0; i < SMALL_CAP; i++) {
        if (smallSlots[i].occupied) smallOcc++; else smallFree++;
    }

    int twoFree = 0, twoOcc = 0;
    for (int i = 0; i < TWO_CAP; i++) {
        if (twoSlots[i].occupied) twoOcc++; else twoFree++;
    }

    printf("\nSTATUS SUMMARY\n");
    printf("Big 4-wheeler slots: %d occupied, %d free (rate Rs
%d/hr)\n", bigOcc, bigFree, PRICE_BIG);
    printf("Small 4-wheeler slots: %d occupied, %d free (rate Rs
%d/hr)\n", smallOcc, smallFree, PRICE_SMALL);
    printf("2-wheeler slots: %d occupied, %d free (rate Rs %d/hr)\n",
twoOcc, twoFree, PRICE_TWO);

    // Optionally list occupied slots briefly
    printf("\nOccupied slots (sample):\n");
    for (int i = 0; i < BIG_CAP; i++) {
```

```
        if (bigSlots[i].occupied)
            printf("BIG slot %d token %d vehicle %s entry %d\n", i + 1,
bigSlots[i].token, bigSlots[i].vehicle_no, bigSlots[i].entry_hour);
    }
    for (int i = 0; i < SMALL_CAP; i++) {
        if (smallSlots[i].occupied)
            printf("SMALL slot %d token %d vehicle %s entry %d\n", i +
1, smallSlots[i].token, smallSlots[i].vehicle_no,
smallSlots[i].entry_hour);
    }
    for (int i = 0; i < TWO_CAP; i++) {
        if (twoSlots[i].occupied)
            printf("TWO slot %d token %d vehicle %s entry %d\n", i + 1,
twoSlots[i].token, twoSlots[i].vehicle_no, twoSlots[i].entry_hour);
    }
}

// Helper: clear input line when scanf failed
void clearInputLine(void) {
    int c;
    while ((c = getchar()) != '\n' && c != EOF) {
        ;
    }
}
```