Program No. 1: Write a program to perform the following tasks:

1. Imagine a tollbooth at a bridge. Cars passing by the booth are expected to pay a 50 cent toll. Mostly they do, but sometimes a car goes by without paying. The tollbooth keeps track of the number of cars that have gone by, and of the total amount of money collected. Model this tollbooth with a class called tollBooth. The two data items are a type unsigned int to hold the total number of cars, and a type double to hold the total amount of money collected. A constructor initializes both of these to 0. A member function called payingCar() increments the car total and adds 0.50 to the cash total. Another function, called nopayCar(), increments the car total but adds nothing to the cash total. Finally, a member function called display() displays the two totals. Make appropriate member functions const.
Include a program to test this class. This program should allow the user to push one key to count a paying car, and another to count a nonpaying car. Pushing the Esc key should cause the program to print out the total cars and total cash and then exit.

Source Code

```
#include <iostream>
#include <conio.h>
using namespace std;
class tollBooth {
private:
    unsigned int car;
    double cash;

public:
    // Constructor
    tollBooth() : car(0), cash(0.0) {}

    // Member function for paying car
    void paying() {
        car++;
        cash += 0.50;
    }

    // Member function for non-paying car
    void nonpaying() {
        car++;
    }

    // display function
    void disp() const {
       cout << "\nTotal cars passed: " << car <<endl;
       cout << "Total cash collected: ₹" << cash <<endl;
    }
};

int main() {
```

```
    tollBooth booth;
    char ch;

   cout << "TollBooth Interface\n";
   cout << "Press 'p' for paying car, 'n' for non-paying car, 'Esc' to exit and
display totals.\n";

    while (true) {
        ch = _getch();  // Get character input without Enter

        if (ch == 27) { // ASCII code for Esc key
            booth.disp();
            break;
        } else if (ch == 'p' || ch == 'P') {
            booth.paying();
           cout << "Paying car recorded.\n";
        } else if (ch == 'n' || ch == 'N') {
            booth.nonpaying();
           cout << "Non-paying car recorded.\n";
        } else {
           cout << "Invalid input. Use 'p', 'n', or Esc.\n";
        }
    }

    return 0;
}
```

1. Output (Screenshot)

```
TollBooth Interface
Press 'p' for paying car, 'n' for non-paying car, 'Esc' to exit and display totals.
Paying car recorded.
Paying car recorded.
Paying car recorded.
Paying car recorded.
Paying car recorded.
Paying car recorded.
Paying car recorded.
Paying car recorded.
Paying car recorded.
Paying car recorded.
Paying car recorded.
Non-paying car recorded.
Non-paying car recorded.
Non-paying car recorded.
Non-paying car recorded.

Total cars passed: 15
Total cash collected: $5.5
```

Program No. 2: Write a program to perform the following tasks:

1. In an airline reservation system, when a family or group books tickets together, many details (such as flight number and route) remain the same. Instead of re-entering all the details for each passenger, the system can duplicate an existing ticket object.
You are required to design a class Ticket in C++ that demonstrates how a copy constructor can be used to duplicate ticket details for multiple passengers in a group booking.
Class Design
• Data Members
o string passengerName
o string flightNumber
o string seatNumber
• Constructors
o Parameterized Constructor – initializes passenger details.
o Copy Constructor – duplicates an existing Ticket object.
• Member Function
o void displayTicket() – prints ticket details.
• Destructor
o Displays a message when a ticket object is destroyed (to track lifecycle).
Detailed Tasks
1. Create a primary ticket for the first passenger using the parameterized constructor.
2. Use the copy constructor to generate duplicate tickets for family/group members.
3. Modify only passenger name and seat number for new passengers (since flight number remains the same).
4. Display all ticket details for the group.
5. Show how the destructor is called for each ticket object at the end of the program.

Source Code

```
#include <iostream>
#include <string>
using namespace std;

class Ticket {
private:
    string pass_name;
    string flight_num;
    string seat_num;

public:
    // Constructor
    Ticket(string name, string flight, string seat)
        : pass_name(name), flight_num(flight), seat_num(seat) {
        cout << "Ticket created for " << pass_name << endl;
    }

    // Copy Constructor
    Ticket(const Ticket& original)
        : pass_name(original.pass_name),
          flight_num(original.flight_num),
```

```
          seat_num(original.seat_num) {
        cout << "Ticket copied for " << pass_name << endl;
    }

    // Modify name and seat
    void upd_pass(string name, string seat) {
        pass_name = name;
        seat_num = seat;
    }

    // Displaydetails
    void disp() const {
        cout << "Passenger Name: " << pass_name << endl;
        cout << "Flight Number : " << flight_num << endl;
        cout << "Seat Number   : " << seat_num << endl;
        cout << "     " << endl;
    }

    // Destructor
    ~Ticket() {
        cout << "Ticket for " << pass_name << " is destroyed." << endl;
    }
};

int main() {
    // Step 1: Create primary ticket
    Ticket primary("Robins", "AI202", "12A");

    // Step 2: Use copy constructor for group members
    Ticket m1 = primary;
    Ticket m2 = primary;

    // Step 3: Update passenger name and seat number
    m1.upd_pass("Pranav", "12B");
    m2.upd_pass("Dinesh", "12C");

    // Step 4: Display all details
    cout << "\nGroup Booking Details:\n";
    cout << "       " << endl;
    primary.disp();
    m1.disp();
    m2.disp();
    return 0;
}
```

1. Output (Screenshot)

```
Ticket created for Robins
Ticket copied for Robins
Ticket copied for Robins

Group Booking Details:

Passenger Name: Robins
Flight Number : AI202
Seat Number   : 12A

Passenger Name: Pranav
Flight Number : AI202
Seat Number   : 12B

Passenger Name: Dinesh
Flight Number : AI202
Seat Number   : 12C

Ticket for Dinesh is destroyed.
Ticket for Pranav is destroyed.
Ticket for Robins is destroyed.
```

Program No. 3: Write a program to perform the following tasks:

1. In a company, it is essential to track how many employees are currently active in the system. Each time a new employee joins, the system should increase the employee count, and when an employee leaves (or the object is destroyed), the count should decrease automatically. You are required to design a class Employee in C++ that demonstrates how static data members can be used to track the number of live objects in a program.
Class Design
• Data Members
o string name – Employee name
o int id – Employee ID
o static int employeeCount – Static counter for active employees
• Constructors
o Parameterized constructor – Initializes employee details and increments employeeCount.
• Destructor
o Decrements employeeCount and displays a message indicating which employee left.
• Member Function
o void displayEmployee() – Prints the employee's name and ID.
o static void displayCount() – Prints the current number of active employees.
Detailed Tasks
1. Create several employee objects using the parameterized constructor.
2. Display the total number of active employees after each creation.
3. Destroy some employee objects (either manually or when they go out of scope).
4. Display the total number of active employees after destruction.
5. Demonstrate that the static counter is shared among all objects.

Source Code

```
#include <iostream>
#include <string>
using namespace std;

class emp {
private:
    string name;
    int id;
    static int count;  // Shared among all objects
public:
    // Parameterized Constructor
    emp(string empName, int empId) : name(empName), id(empId) {
        count++;
        cout << "Employee " << name << " (ID: " << id << ") joined. ";
        disp_count();
    }

    // Destructor
    ~emp() {
```

```
        count--;
        cout << "Employee " << name << " (ID: " << id << ") left. ";
        disp_count();
    }
    // Displaydetails
    void disp_emp() const {
        cout << "Name: " << name << ", ID: " << id << endl;
    }
    // display count
    static void disp_count() {
        cout << "Active Employees: " << count << endl;
    }
};

// Definition of static member
int emp::count = 0;

int main() {
    cout << "Company Employee Tracker\n\n";

    // Step 1: Create employee objects
    emp e1("Robins", 101);
    emp e2("Pranav", 102);

    // Step 2: Display individual
    cout << "\nEmployee Details:\n";
    e1.disp_emp();
    e2.disp_emp();

    // Step 3:Create a employee
    {
        emp e3("Rudrakshi", 103);
        e3.disp_emp();
        // e3 will be destroyed
    }

    // Step 4:Display count
    cout << "\nAfter Rudrakshi leaves:\n";
    emp::disp_count();

    // Step 5:shared static counter
    cout << "\nCreating another employee:\n";
    emp e4("Jass", 104);
    emp::disp_count();

    return 0;
}
```

1. Output (Screenshot)

```
Company Employee Tracker

Employee Robins (ID: 101) joined. Active Employees: 1
Employee Pranav (ID: 102) joined. Active Employees: 2

Employee Details:
Name: Robins, ID: 101
Name: Pranav, ID: 102
Employee Rudrakshi (ID: 103) joined. Active Employees: 3
Name: Rudrakshi, ID: 103
Employee Rudrakshi (ID: 103) left. Active Employees: 2

After Rudrakshi leaves:
Active Employees: 2

Creating another employee:
Employee Jass (ID: 104) joined. Active Employees: 3
Active Employees: 3
Employee Jass (ID: 104) left. Active Employees: 2
Employee Pranav (ID: 102) left. Active Employees: 1
Employee Robins (ID: 101) left. Active Employees: 0
```