Task 9: Smart Pointers – std::unique_ptr in C++

Objective:
To understand and demonstrate the use of std::unique_ptr in C++. This task illustrates how unique_ptr manages dynamic memory automatically and ensures exclusive ownership of an object.

1. Concept Overview

Smart pointers are objects that manage dynamically allocated memory automatically. std::unique_ptr is a smart pointer that owns a resource exclusively. It cannot be copied, only moved, which prevents memory leaks and double deletion.

2. Program Code

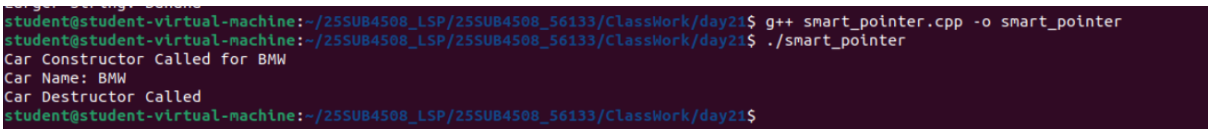
```
#include <iostream>
 #include <memory>
 using namespace std;

 class Car {
 public:
        Car(string name) {
        cout << "Car Constructor Called for " << name << endl;
        carName = name;
        }

        void display() {
        cout << "Car Name: " << carName << endl;
        }

        ~Car() {
        cout << "Car Destructor Called" << endl;
        }

 private:
        string carName;
};

 int main() {
        unique_ptr<Car> carPtr = make_unique<Car>("BMW");

        carPtr->display();
```

```
    return 0;
}
```

3. Compilation Instructions

```
g++ smart_pointer.cpp -o smart_pointer
```

4. Sample Output



5. Observations & Explanation

1. std::unique_ptr is used to create a Car object dynamically.
2. make_unique() safely allocates memory and returns a unique_ptr.
3. The arrow operator (->) is used to access class members.
4. The Car destructor is automatically called when unique_ptr goes out of scope.
5. Manual memory deallocation is not required.

6. Advantages of std::unique_ptr

• Prevents memory leaks
• Ensures single ownership
• Automatic resource cleanup
• Safer than raw pointers

7. Conclusion

This task demonstrates how std::unique_ptr simplifies memory management in C++. By enforcing exclusive ownership and automatic destruction, unique_ptr helps write safer and more reliable programs.