Q1: No, it’s not a member function, so it cannot be overridden in a derived class.

Q2: When writing 5 + obj, 5 is the first argument. In case of obj + 5, obj becomes the calling object and 5 is sent as an argument. For the statement 5 + obj, 5 would be expected to be the calling object. But 5 is a primitive (int) and cannot be the calling object. So, it would fail.

Q3: Because static members are created regardless of whether there are any objects created or not. Each object is created with its own data members however the static member is not reallocated because it has already been created at the time of program execution.

Q4: No, we cannot tell until we see if performTask() is a virtual function in the parent class

Q5: Yes, it will create ambiguity as the statement will theoretically result in both function calls, so neither can be called / will result in a compile time error.

You are tasked with designing an AI-powered smart car system that consists of multiple AI modules working together to enhance driving capabilities.

<u>Note:</u> Program each part separately, do not merge them together in classes- you can comment which class the code snippets belong to when attempting each part.

a. The system includes a base class **AIComponent**, which serves as the core AI framework. The car has different AI **subsystems**; **NavigationAI** which handles GPS and route planning has data members start, destination and estimated_time, **EntertainmentAI** which manages music and voice interactions has data members volume and song playlist and **SelfDrivingAI** which combines navigation and entertainment while making autonomous driving decisions has a data member current_speed. Each AI module should have a function performTask() to demonstrate dynamic polymorphism. There need not be any function logic, but it should print the task that the AI is performing. **(3 Marks)**

b. **NavigationAI** has a function **routeMap** that accepts a **starting point**(string) and **destination**(string) to plan a route towards the destination (It must print a message "Starting navigation from Source to Destination. Drive safely!"). **SelfdrivingAI** should override this function to also ask the user their entertainment choices, and then **invoke the routeMap method present in NavigationAI**. **(2 Marks)**

c. Create a function **demonstratePolymorphism**() : Make an array of **AIComponent*** pointers (in this function) to store different AI modules(one of each type of AI) and call **performTask() dynamically. (4 marks)**

d. **compareRoutes(NavigationAI, NavigationAI)** – A friend function that takes two NavigationAI objects as arguments and checks if they have identical routes. If both objects have the same start and destination, it returns true, otherwise false. **(2 marks)**

e. **RemoteAIControl (Friend of SelfDrivingAI)** – A specialized class that acts as an external override mechanism for SelfDrivingAI. It allows a remote operator to modify the speed of the car to ensure passenger safety, or to update the destination of the car. **(3 marks)**

f. You are required to implement operator overloading that enables the following operations to work in the main function. **(3 x 2 = 6 marks)**

```
int main() {
     NavigationAI n1;
     SelfDrivingAI s1;
     EntertainmentAI e1,e2;
     n1*5; //updates estimated_time of n1 to estimated_time*5;
     if(s1==n1) cout<< “Routes are identical”;
     e1+=e2; //Merges e2’s playlist into a e1
```

```
}
```

Solution:

Part a:

```
class AIComponent {
public:
    virtual void performTask() const = 0;
    virtual ~AIComponent() {}
};

class NavigationAI : virtual public AIComponent {
protected:
    string start, destination;
    int estimated_time;

public:
    NavigationAI(string s = "", string d = "", int time = 0)
        : start(s), destination(d), estimated_time(time) {}

    void performTask() const override {
        cout << "NavigationAI: Calculating best route...\n";
    }
    string getStart() const { return start; }
    string getDestination() const { return destination; }
};
```

Part b:

```
virtual void NavigationAI::routeMap(string s, string d) {
        start = s;
        destination = d;
        cout << "Starting navigation from " << start << " to " << destination << ".
Drive safely!\n";
    }
```

```
class EntertainmentAI : virtual public AIComponent {
    int volume;
    string playlist[10]; // Fixed size playlist
    int playlistCount;

public:
    EntertainmentAI(int vol = 5) : volume(vol), playlistCount(0) {}

    void addToPlaylist(const string& song) {
        if (playlistCount < 10) {
            playlist[playlistCount++] = song;
        }
    }

    void performTask() const override {
        cout << "EntertainmentAI: Playing music and responding to voice
commands...\n";
    }

    void printPlaylist() const {
        cout << "Playlist:\n";
        for (int i = 0; i < playlistCount; ++i) {
            cout << " - " << playlist[i] << '\n';
        }
    }
};

class SelfDrivingAI : public NavigationAI, public EntertainmentAI {
    int current_speed;

public:
    SelfDrivingAI(string s = "", string d = "", int time = 0, int speed = 0)
```

```
        : NavigationAI(s, d, time), current_speed(speed) {}

    void performTask() const override {
        cout << "SelfDrivingAI: Making real-time autonomous driving decisions...\n";
    }

    void routeMap(string s, string d) override {
        cout << "SelfDrivingAI: Asking for entertainment preferences...\n";
        NavigationAI::routeMap(s, d);
    }
};
```

Part c:

```
void demonstratePolymorphism() {
    AIComponent* modules[3];

    modules[0] = new NavigationAI("Home", "Work", 15);
    modules[1] = new EntertainmentAI();
    modules[2] = new SelfDrivingAI("Home", "Work", 15, 50);

    for (int i = 0; i < 3; i++) {
        modules[i]->performTask();
        delete modules[i];
    }
}
```

Part d:

```
//in NavigationAI class
friend bool compareRoutes(const NavigationAI&, const NavigationAI&);
//globa bool compareRoutes(const NavigationAI& a, const NavigationAI& b) {
    return (a.start == b.start && a.destination == b.destination);
}l
```

Part e:

```
//goes in SelfDrivingAI class
friend class RemoteAIControl;
//global
class RemoteAIControl {
public:
    void changeSpeed(SelfDrivingAI& car, int newSpeed) {
        car.current_speed = newSpeed;
        cout << "RemoteAIControl: Speed updated to " << newSpeed << " km/h\n";
    }

    void changeDestination(SelfDrivingAI& car, const string& newDest) {
        car.destination = newDest;
        cout << "RemoteAIControl: Destination changed to " << newDest << "\n";
    }
};
```

Part f:

```
//in NavigationAI class
void NavigationAI::operator*(int multiplier) {
        estimated_time *= multiplier;
    }
//in SelfDrivingAI class
bool SelfDrivingAI::operator==(const NavigationAI& nav) {
        return (start == nav.getStart() && destination == nav.getDestination());
    }
//in Entertainment AI class
void EntertainmentAI::operator+=(const EntertainmentAI& other) {
        for (int i = 0; i < other.playlistCount && playlistCount < 10; ++i) {
            playlist[playlistCount++] = other.playlist[i];
        }
    }
```