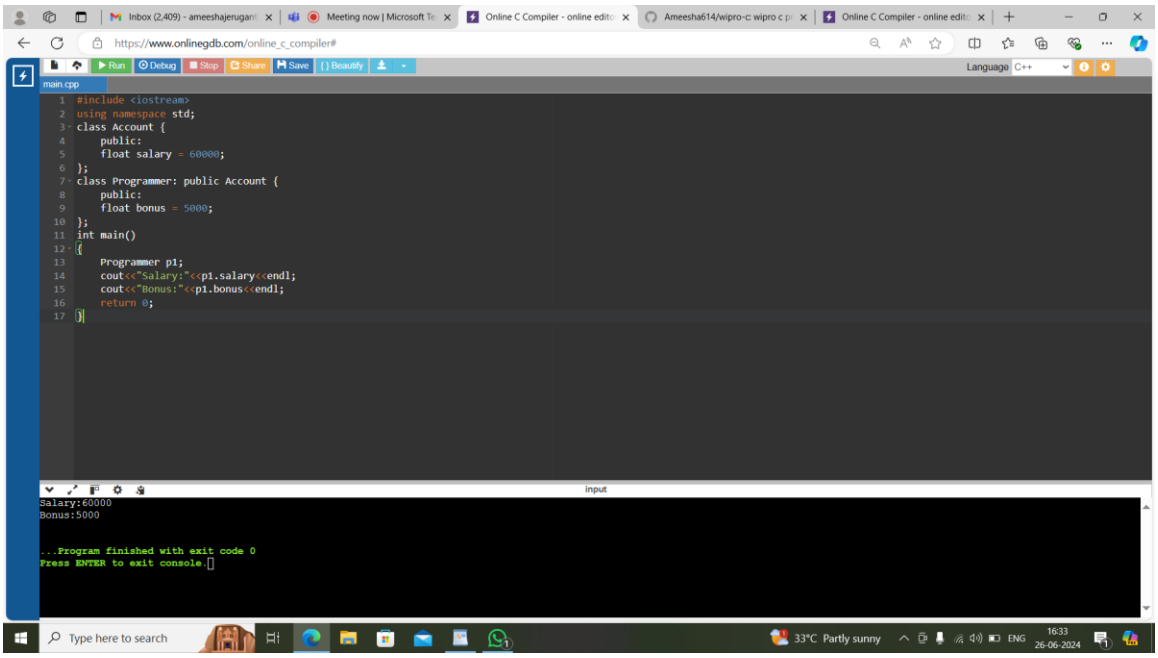
SINGLE INHERITANCE :-

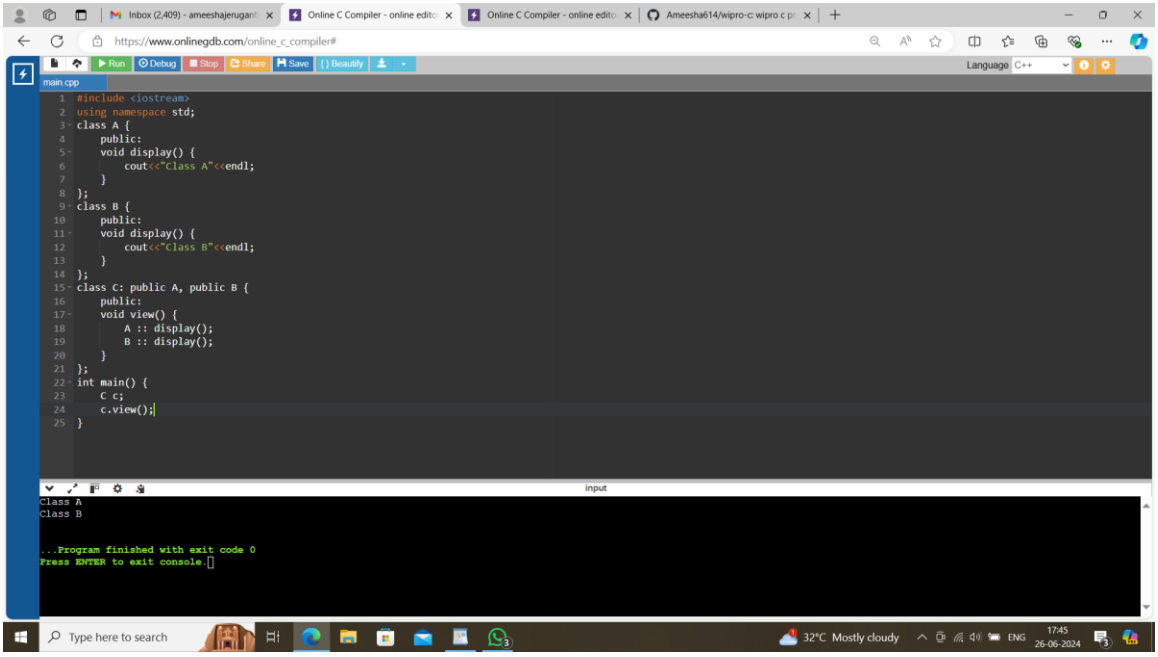
```
#include <iostream>

using namespace std;

class Account {

	public:

	float salary = 60000;

};

class Programmer: public Account {

	public:

	float bonus = 5000;

};

int main()

{

	Programmer p1;

	cout<<"Salary:"<<p1.salary<<endl;

	cout<<"Bonus:"<<p1.bonus<<endl;

	return 0;

}
```



AMBIGUITY IN INHERITANCE :-

```
#include <iostream>

using namespace std;

class A {

        public:

        void display() {

                cout<<"Class A"<<endl;

        }

};

class B {

        public:

        void display() {

                cout<<"Class B"<<endl;

        }

};

class C: public A, public B {
```

```
        public:
        void view() {
                A :: display();
                B :: display();
            }
};
int main() {
        C c;
        c.view();
}
```



POLYMORPHISM :-

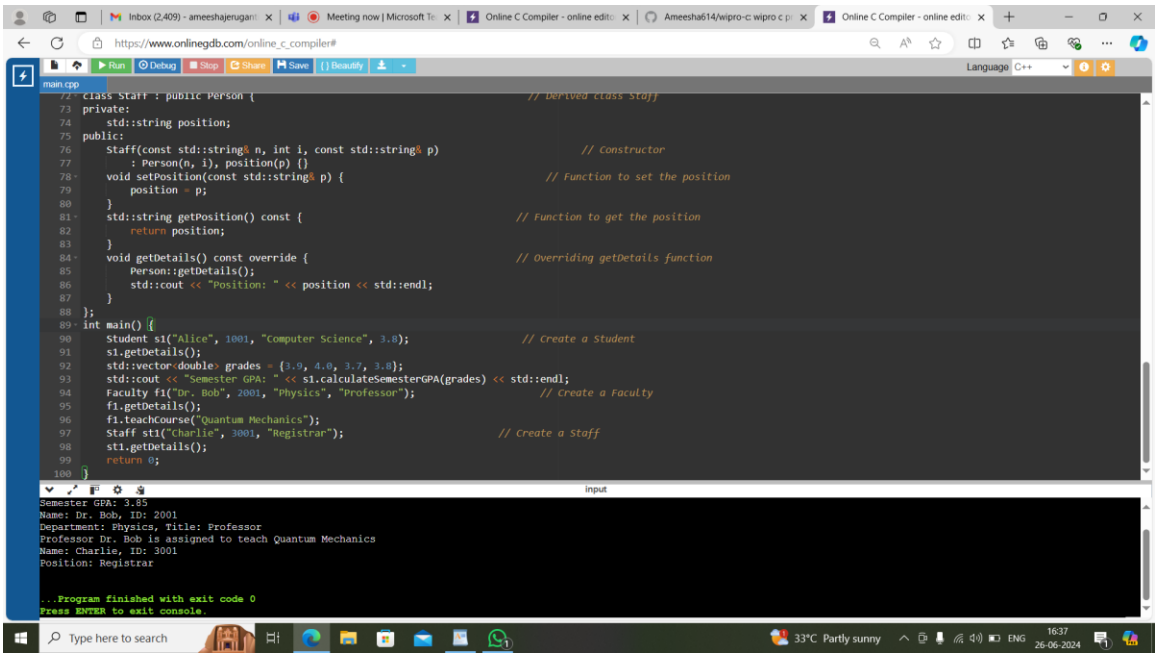
```
#include <iostream>
#include <string>
#include <vector>
#include <numeric>          // For accumulate
```

```
class Person {
protected:
    std::string name;
    int id;
public:
    Person(const std::string& n, int i) : name(n), id(i) {}          // Constructor
    virtual void getDetails() const {                                // Virtual function to print details
        std::cout << "Name: " << name << ", ID: " << id << std::endl;
    }
    virtual ~Person() {}                                             // Virtual destructor
};
class Student : public Person {                                      // Derived class Student
private:
    std::string major;
    double gpa;
public:
    Student(const std::string& n, int i, const std::string& m, double g)          // Constructor
        : Person(n, i), major(m), gpa(g) {}
    void setMajor(const std::string& m) {                            // Function to set the major
        major = m;
    }
    std::string getMajor() const {                                   // Function to get the major
        return major;
    }
    void setGPA(double g) {                                          // Function to set the GPA
        gpa = g;
    }
    double getGPA() const {                                          // Function to get the GPA
```

```
        return gpa;
    }
    double calculateSemesterGPA(const std::vector<double>& grades) const {                    // Function
to calculate semester GPA
        if (grades.empty()) return 0.0;
        double total = std::accumulate(grades.begin(), grades.end(), 0.0);
        return total / grades.size();
    }
    void getDetails() const override {                                                    // Overriding getDetails
function
        Person::getDetails();
        std::cout << "Major: " << major << ", GPA: " << gpa << std::endl;
    }
};
class Faculty : public Person {                                          // Derived class Faculty
private:
    std::string department;
    std::string title;
public:
    Faculty(const std::string& n, int i, const std::string& d, const std::string& t)
        : Person(n, i), department(d), title(t) {}
    void setDepartment(const std::string& d) {                                    // Function to set the
department
        department = d;
    }
    std::string getDepartment() const {                                          // Function to get the
department
        return department;
    }
```

```
    void setTitle(const std::string& t) {                    // Function to set the title
        title = t;
    }
    std::string getTitle() const {                           // Function to get the title
        return title;
    }
    void teachCourse(const std::string& courseName) const {  // Function to assign a course to teach
        std::cout << title << " " << name << " is assigned to teach " << courseName << std::endl;
    }
    void getDetails() const override {                       // Overriding getDetails function
        Person::getDetails();
        std::cout << "Department: " << department << ", Title: " << title << std::endl;
    }
};
class Staff : public Person {                                // Derived class Staff
private:
    std::string position;
public:
    Staff(const std::string& n, int i, const std::string& p)  // Constructor
        : Person(n, i), position(p) {}
    void setPosition(const std::string& p) {                 // Function to set the position
        position = p;
    }
    std::string getPosition() const {                        // Function to get the position
        return position;
```

```
    }
    void getDetails() const override {                                  // Overriding
getDetails function
        Person::getDetails();
        std::cout << "Position: " << position << std::endl;
    }
};
int main() {
    Student s1("Alice", 1001, "Computer Science", 3.8);                 // Create a Student
    s1.getDetails();
    std::vector<double> grades = {3.9, 4.0, 3.7, 3.8};
    std::cout << "Semester GPA: " << s1.calculateSemesterGPA(grades) << std::endl;
    Faculty f1("Dr. Bob", 2001, "Physics", "Professor");                // Create a Faculty
    f1.getDetails();
    f1.teachCourse("Quantum Mechanics");
    Staff st1("Charlie", 3001, "Registrar");                            // Create a Staff
    st1.getDetails();
    return 0;
}
```



FUNCTION OVERLOADING :-

```
#include <iostream>

using namespace std;

class cal {

        public:

        static int add(int a, int b) {

                return a + b;

        }

        static int add(int a, int b, int c)

        {

                return a + b + c;

        }

};

int main(void) {

        cal C;

        cout<<C.add(10,20)<<endl;
```

```
    cout<<C.add(12,20,23);

    return 0;

}
```