RAJA MUHAMMAD JAHANZAIB

BCS 233157

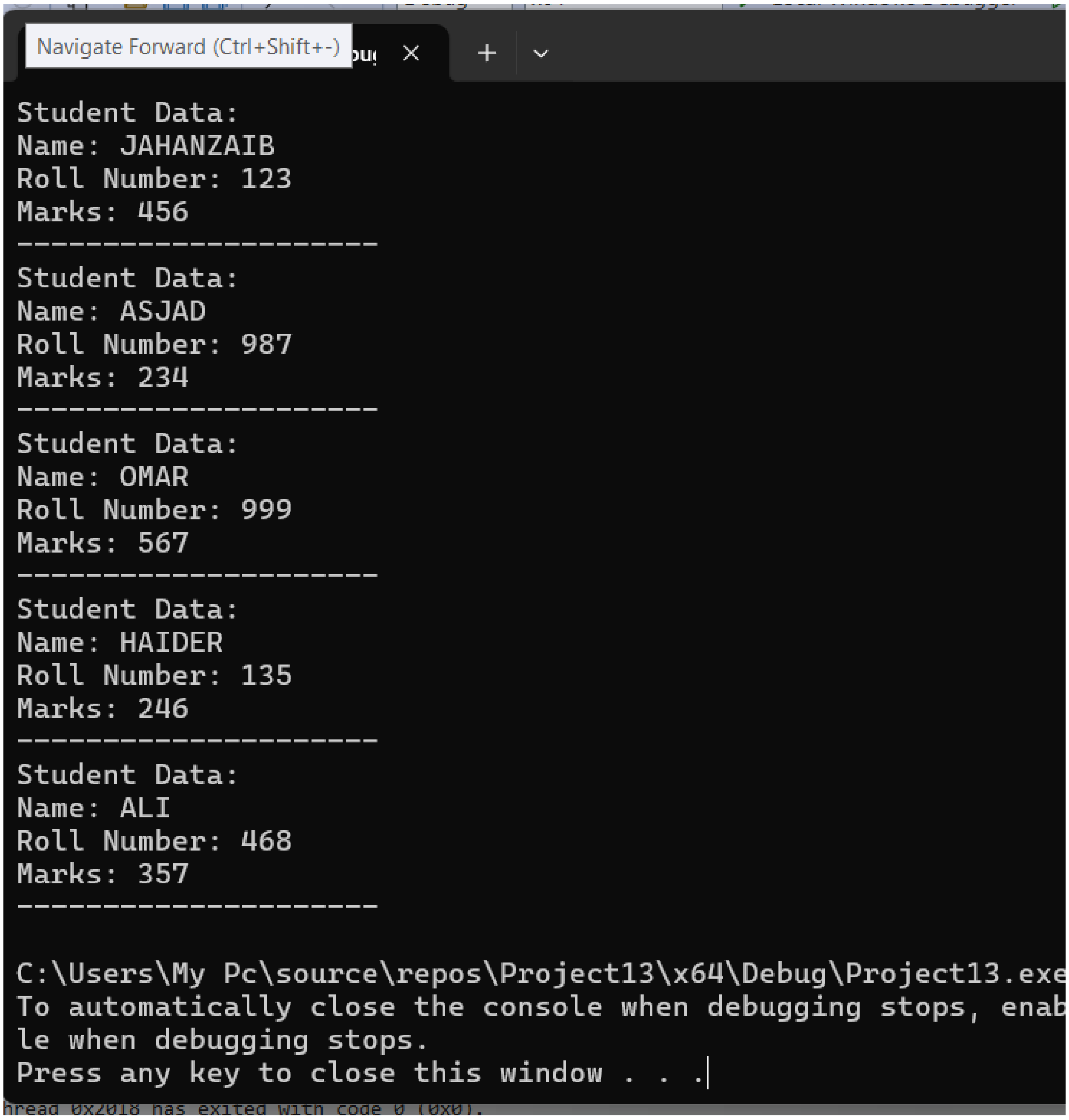
ASSIGNMENT # 03

TASK 1: FILE HANDLING

SOLUTION:

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

class Student {
public:
    string name;
    int roll_number;
    int marks;

    void input() {
        cout << "Enter name: ";
        cin >> name;
        cout << "Enter roll number: ";
        cin >> roll_number;
        cout << "Enter marks: ";
        cin >> marks;
    }

    void display() {
        cout << "Name: " << name << endl;
        cout << "Roll Number: " << roll_number << endl;
        cout << "Marks: " << marks << endl;
    }
};

int main() {
    // Writing data of 5 students to the file
    ofstream outFile("students.txt");
    if (outFile.is_open()) {
        for (int i = 0; i < 5; i++) {
            Student student;
            student.input();
            outFile << student.name << " " << student.roll_number << " " <<
student.marks << endl;
        }
        outFile.close();
    }
    else {
        cout << "Unable to open file for writing." << endl;
        return 1;
    }

    // Reading data from the file and displaying it
    ifstream inFile("students.txt");
    if (inFile.is_open()) {
        string name;
        int roll_number, marks;
        while (inFile >> name >> roll_number >> marks) {
            cout << "Student Data:" << endl;
            cout << "Name: " << name << endl;
            cout << "Roll Number: " << roll_number << endl;
            cout << "Marks: " << marks << endl;
            cout << "---------------------" << endl;
        }
        inFile.close();
    }
    else {
        cout << "Unable to open file for reading." << endl;
        return 1;
    }

    // Modifying the marks of a specific student
    fstream file("students.txt", ios::in | ios::out);
    if (file.is_open()) {
        string name;
        int roll_number, marks;
        int target_roll_number = 123; // Assume the roll number to modify marks
        while (file >> name >> roll_number >> marks) {
            if (roll_number == target_roll_number) {
                // Modify the marks
                marks = 95; // New marks value

                // Move the file pointer to the beginning of the record
                file.seekp(file.tellg());
                file << name << " " << roll_number << " " << marks << endl;
                break;
            }
        }
        file.close();
    }
    else {
        cout << "Unable to open file for modification." << endl;
        return 1;
    }

    return 0;
}
```

OUTPUT:



```
Student Data:
Name: JAHANZAIB
Roll Number: 123
Marks: 456
---------------------
Student Data:
Name: ASJAD
Roll Number: 987
Marks: 234
---------------------
Student Data:
Name: OMAR
Roll Number: 999
Marks: 567
---------------------
Student Data:
Name: HAIDER
Roll Number: 135
Marks: 246
---------------------
Student Data:
Name: ALI
Roll Number: 468
Marks: 357
---------------------

C:\Users\My Pc\source\repos\Project13\x64\Debug\Project13.exe
To automatically close the console when debugging stops, enab
le when debugging stops.
Press any key to close this window . . .
hread 0x2018 has exited with code 0 (0x0).
```

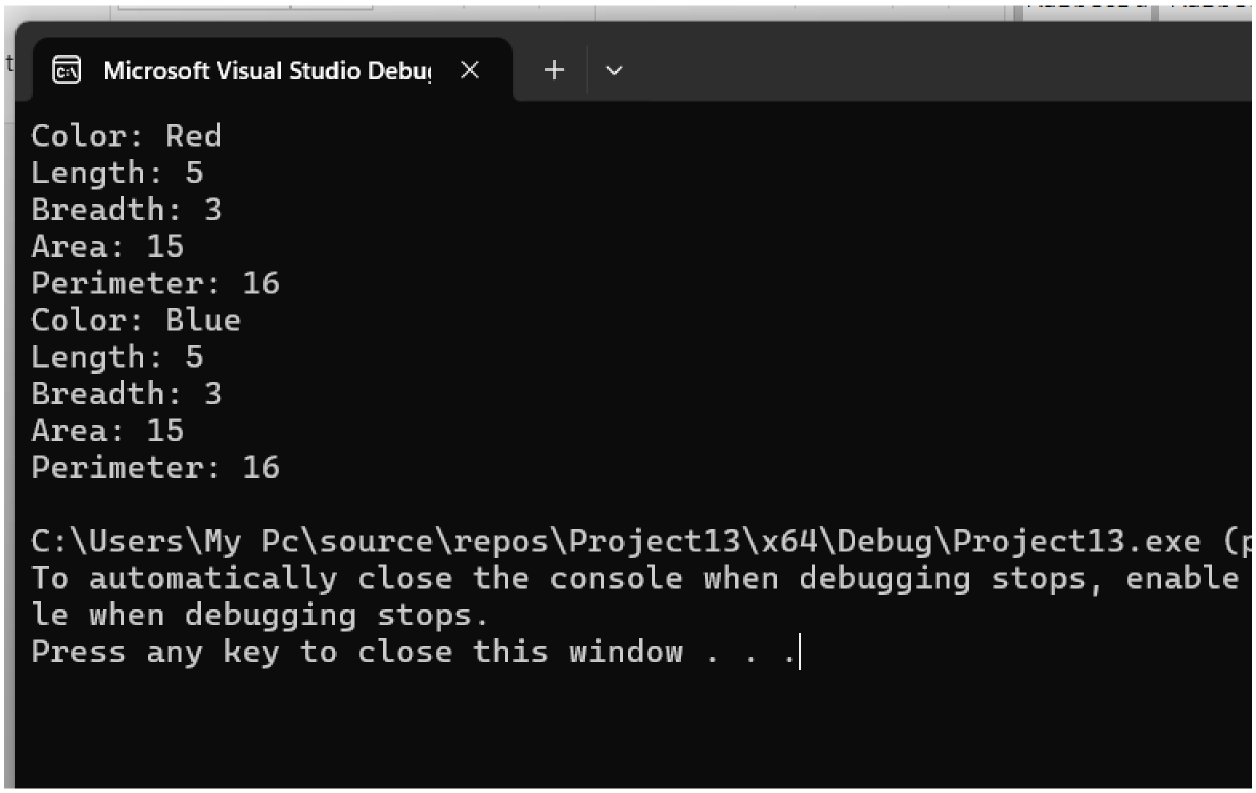
TASK 2(A): INHERITANCE

SOLUTION:

```cpp
#include <iostream>
#include <string>

using namespace std;

// Base class Shape
class Shape {
protected:
    string color;
```

```cpp
public:
    // Constructor
    Shape(string c) : color(c) {}

    // Member function to set color
    void setColor(string c) {
        color = c;
    }

    // Member function to get color
    string getColor() {
        return color;
    }
};

// Derived class Rectangle
class Rectangle : public Shape {
private:
    double length;
    double breadth;

public:
    // Constructor
    Rectangle(string c, double l, double b) : Shape(c), length(l), breadth(b) {}

    // Member function to calculate area
    double calculateArea() {
        return length * breadth;
    }

    // Member function to calculate perimeter
    double calculatePerimeter() {
        return 2 * (length + breadth);
    }

    // Member function to display rectangle details
    void display() {
        cout << "Color: " << getColor() << endl;
        cout << "Length: " << length << endl;
        cout << "Breadth: " << breadth << endl;
        cout << "Area: " << calculateArea() << endl;
        cout << "Perimeter: " << calculatePerimeter() << endl;
    }
};

int main() {
    // Create a Rectangle object
    Rectangle r("Red", 5.0, 3.0);

    // Display rectangle details
    r.display();

    // Change the color of the rectangle
    r.setColor("Blue");

    // Display updated rectangle details
    r.display();
    return 0;
}
```

OUTPUT:



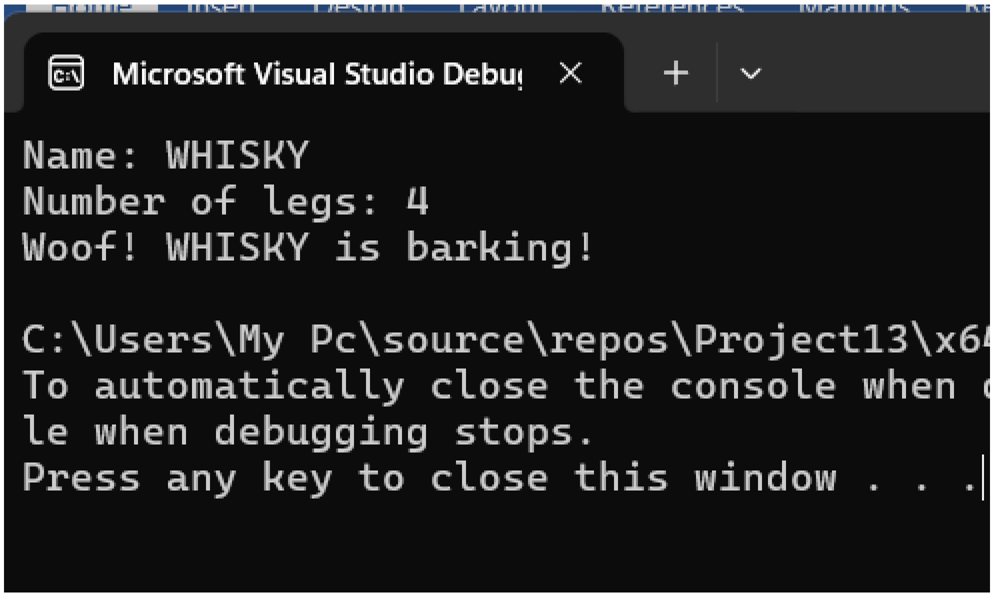
```
Color: Red
Length: 5
Breadth: 3
Area: 15
Perimeter: 16
Color: Blue
Length: 5
Breadth: 3
Area: 15
Perimeter: 16

C:\Users\My Pc\source\repos\Project13\x64\Debug\Project13.exe (p
To automatically close the console when debugging stops, enable
le when debugging stops.
Press any key to close this window . . .
```

TASK 2(B): MULTILEVEL INHERITANCE

SOLUTION:

```
#include <iostream>
#include <string>

using namespace std;

// Base class Animal
class Animal {
protected:
    string name;

public:
    // Constructor
    Animal(string n) : name(n) {}

    // Member function to set name
    void setName(string n) {
        name = n;
    }
    // Member function to get name
    string getName() {
        return name;
    }
};

// Derived class Mammal
class Mammal : public Animal {
private:
    int numberOfLegs;

public:
    // Constructor
    Mammal(string n, int legs) : Animal(n), numberOfLegs(legs) {}

    // Member function to set number of legs
    void setNumberOfLegs(int legs) {
        numberOfLegs = legs;
    }

    // Member function to get number of legs
    int getNumberOfLegs() {
        return numberOfLegs;
    }
};

// Derived class Dog
class Dog : public Mammal {
public:
    // Constructor
    Dog(string n, int legs) : Mammal(n, legs) {}

    // Member function to bark
    void bark() {
        cout << "Woof! " << getName() << " is barking!" << endl;
    }
};

int main() {
    // Create a Dog object
    Dog myDog("WHISKY", 4);

    // Access members of all classes
    cout << "Name: " << myDog.getName() << endl;
    cout << "Number of legs: " << myDog.getNumberOfLegs() << endl;
    myDog.bark();

    return 0;
}
```

OUTPUT:



```
Name: WHISKY
Number of legs: 4
Woof! WHISKY is barking!

C:\Users\My Pc\source\repos\Project13\x6
To automatically close the console when 
le when debugging stops.
Press any key to close this window . . .
```

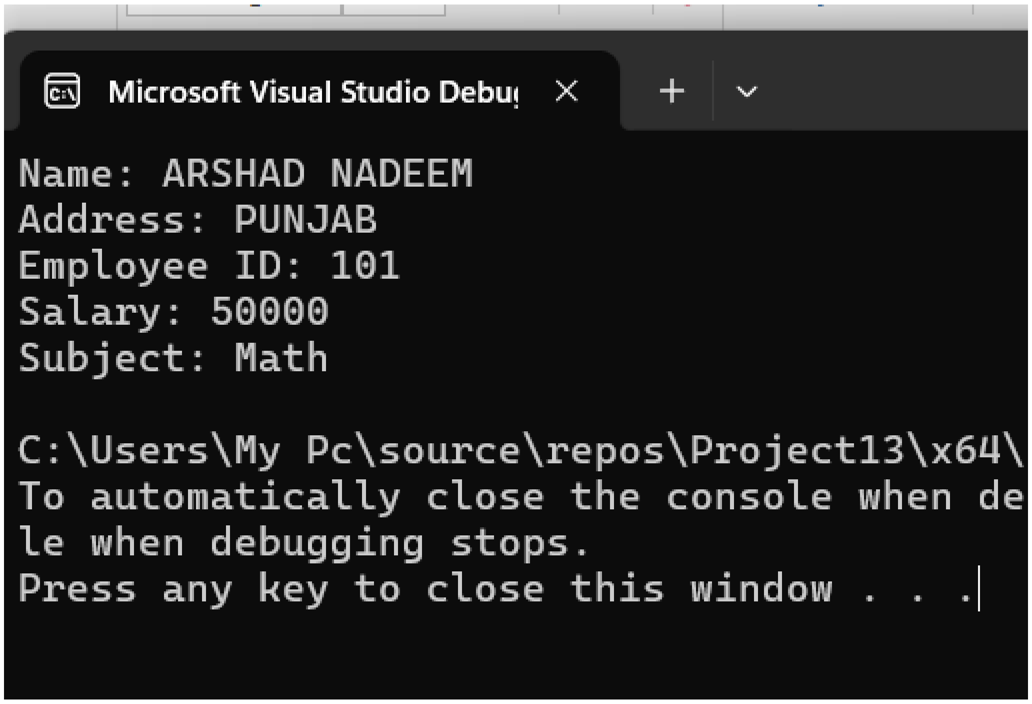
TASK 2(C): MULTIPLE INHERITANCE

SOLUTION:

```cpp
#include <iostream>
#include <string>

using namespace std;

// Base class Person
class Person {
protected:
    string name;
    string address;

public:
    // Constructor
    Person(string n, string a) : name(n), address(a) {}

    // Member function to set name
    void setName(string n) {
        name = n;
    }

    // Member function to get name
    string getName() {
        return name;
    }

    // Member function to set address
    void setAddress(string a) {
        address = a;
    }

    // Member function to get address
    string getAddress() {
        return address;
    }
};

// Base class Employee
class Employee {
protected:
    int employeeID;
    double salary;

public:
    // Constructor
    Employee(int id, double s) : employeeID(id), salary(s) {}

    // Member function to set employee ID
    void setEmployeeID(int id) {
        employeeID = id;
    }

    // Member function to get employee ID
    int getEmployeeID() {
        return employeeID;
    }

    // Member function to set salary
    void setSalary(double s) {
        salary = s;
    }

    // Member function to get salary
    double getSalary() {
        return salary;
    }
};

// Derived class Teacher
class Teacher : public Person, public Employee {
private:
    string subject;

public:
    // Constructor
    Teacher(string n, string a, int id, double s, string sub)
        : Person(n, a), Employee(id, s), subject(sub) {}

    // Member function to set subject
    void setSubject(string sub) {
        subject = sub;
    }

    // Member function to get subject
    string getSubject() {
```

```cpp
        return subject;
    }

    // Member function to display teacher details
    void display() {
        cout << "Name: " << getName() << endl;
        cout << "Address: " << getAddress() << endl;
        cout << "Employee ID: " << getEmployeeID() << endl;
        cout << "Salary: " << getSalary() << endl;
        cout << "Subject: " << getSubject() << endl;
    }
};

int main() {
    // Create a Teacher object
    Teacher t("ARSHAD NADEEM", "PUNJAB", 101, 50000.0, "Math");

    // Access members of all classes
    t.display();

    return 0;
}
```

OUTPUT:



```
Name: ARSHAD NADEEM
Address: PUNJAB
Employee ID: 101
Salary: 50000
Subject: Math

C:\Users\My Pc\source\repos\Project13\x64\
To automatically close the console when de
le when debugging stops.
Press any key to close this window . . .
```

TASK 3: THEORETICAL

1. INHERITACE AND TYPES

Inheritance is a fundamental concept in object-oriented programming that allows a new class (derived class) to inherit properties and behavior from an existing class (base class). This helps promote code reusability and establishes a hierarchical relationship between classes.

There are several types of inheritance in C++:

1. Single Inheritance:
- In single inheritance, a derived class inherits from only one base class.
- Example:

```
cpp
class Animal {
public:
    void eat() {
        cout << "Animal is eating." << endl;
    }
};

class Dog : public Animal {
public:
    void bark() {
        cout << "Dog is barking." << endl;
    }
};
```

2. Multiple Inheritance:
- Multiple inheritance allows a derived class to inherit from more than one base class.
- Example:

```
cpp
class A {
public:
    void displayA() {
        cout << "Class A" << endl;
    }
};

class B {
public:
    void displayB() {
        cout << "Class B" << endl;
    }
};

class C : public A, public B {
public:
    void displayC() {
        cout << "Class C" << endl;
    }
};
```

3. Multilevel Inheritance:
- In multilevel inheritance, a class is derived from another derived class.
- Example:

```cpp
class A {
public:
    void displayA() {
        cout << "Class A" << endl;
    }
};

class B : public A {
public:
    void displayB() {
        cout << "Class B" << endl;
    }
};

class C : public B {
public:
    void displayC() {
        cout << "Class C" << endl;
    }
};
```

2. CONSTRUCTOR CHAINING

Constructor chaining in C++ refers to the process of one constructor calling another constructor in the same class or in a base class. This mechanism allows for the initialization of objects through different constructors within the same class hierarchy.

The process of constructor chaining involves one constructor invoking another constructor to perform common initialization tasks. This is often used to avoid code duplication and ensure that common initialization logic is centralized in one place.

In C++, constructor chaining can be achieved using initializer lists. When a constructor is called, it can pass arguments to another constructor in the same class or in a base class using the initializer list syntax. By doing this, the called constructor initializes the object with the provided arguments before executing its own body.

Constructor chaining is particularly useful in scenarios where multiple constructors in a class need to perform similar initialization tasks. By chaining constructors, you can ensure that the common initialization logic is handled consistently across different constructors, promoting code reusability and maintainability in C++ programs.

3. DIFFERENCE BETWEEN PUBLIC , PRIVATE AND PROTECTED INHERITANCE

In object-oriented programming, public, private, and protected are access specifiers that define the visibility of inherited members in a derived class. Here's a breakdown of the differences between public, private, and protected inheritance:

1. Public Inheritance:

- When a class is publicly inherited, all public members of the base class become public in the derived class, all protected members of the base class become protected in the derived class, and the base class's private members remain inaccessible in the derived class.

- Public inheritance establishes an "is-a" relationship, meaning the derived class is a type of the base class.

- Public inheritance allows the derived class to access the public and protected members of the base class.

2. Private Inheritance:

- With private inheritance, all public and protected members of the base class become private in the derived class, and the base class's private members are inaccessible in the derived class.

- Private inheritance establishes a "has-a" relationship, indicating that the derived class has an instance of the base class but is not a type of the base class.

- Private inheritance restricts access to the base class's members to only the derived class and its friends.

3. Protected Inheritance:

- In protected inheritance, all public and protected members of the base class become protected in the derived class, and the base class's private members are inaccessible in the derived class.

- Protected inheritance is rarely used compared to public and private inheritance.

- Protected inheritance allows derived classes to access the protected members of the base class but not its private members.

In summary, public inheritance allows the derived class to inherit and access both public and protected members of the base class, private inheritance restricts access to the base class's members to only the derived class, and protected inheritance allows derived classes to access the protected members of the base class. Each type of inheritance serves different purposes based on the desired relationship between the base and derived classes.

4. DIAMOND PROBLEM

The diamond problem in multiple inheritance is a common issue that arises in programming languages that support multiple inheritance, such as C++. The problem occurs when a class inherits from two classes that have a common base class. This results in ambiguity for the derived class on how to handle inherited members from the shared base class.

Imagine you have a class A, and two other classes B and C that both inherit from A. Now, if a new class D inherits from both B and C, and there is a method or attribute in class A that is inherited by both B and C, the ambiguity arises in class D.

The diamond problem gets its name from the diamond shape that forms in the class inheritance diagram. This shape occurs when class D inherits from classes B and C, which both inherit from class A, creating a diamond-like structure in the inheritance hierarchy.

To resolve the diamond problem, programming languages use different mechanisms like virtual inheritance in C++. Virtual inheritance ensures that only one instance of the shared base class is inherited by the derived class, preventing issues related to duplicate inherited members and ambiguity.

By using virtual inheritance or other similar mechanisms, the diamond problem can be mitigated, allowing for a clear and unambiguous inheritance hierarchy in cases of multiple inheritance.

5. STREAMS

ifstream, **ofstream**, and **fstream** are classes in C++ that are used for file input, output, and both input/output operations, respectively.

- ifstream: This class is used for input operations from files. It is specifically designed for reading input from files. Objects of ifstream can only be used for reading data from files and not for writing to files.

- ofstream: This class is used for output operations to files. It is used for writing data to files. Objects of ofstream can only be used for writing data to files and not for reading from files.

- fstream: This class is used for both input and output operations on files. Objects of fstream can be used for both reading and writing data to files.

File modes like ios::in, ios::out, ios::app, and ios::binary are used to specify how the file should be opened and how data should be read from or written to the file. Here's a brief explanation of each file mode:

- ios::in: This mode is used for input operations. It indicates that the file is being opened for reading.

- ios::out: This mode is used for output operations. It indicates that the file is being opened for writing. If the file already exists, its contents are truncated.

- ios::app: This mode is used for appending data to the end of the file. It indicates that data is to be written at the end of the file without truncating the existing content.

- ios::binary: This mode is used for binary file operations. It indicates that the file should be opened in binary mode, where data is read or written in binary format without any translation.

By combining these file modes with the appropriate file stream classes (ifstream, ofstream, fstream), you can perform various file operations like reading from a file, writing to a file, appending data to a file, and handling binary data in files effectively in C++.