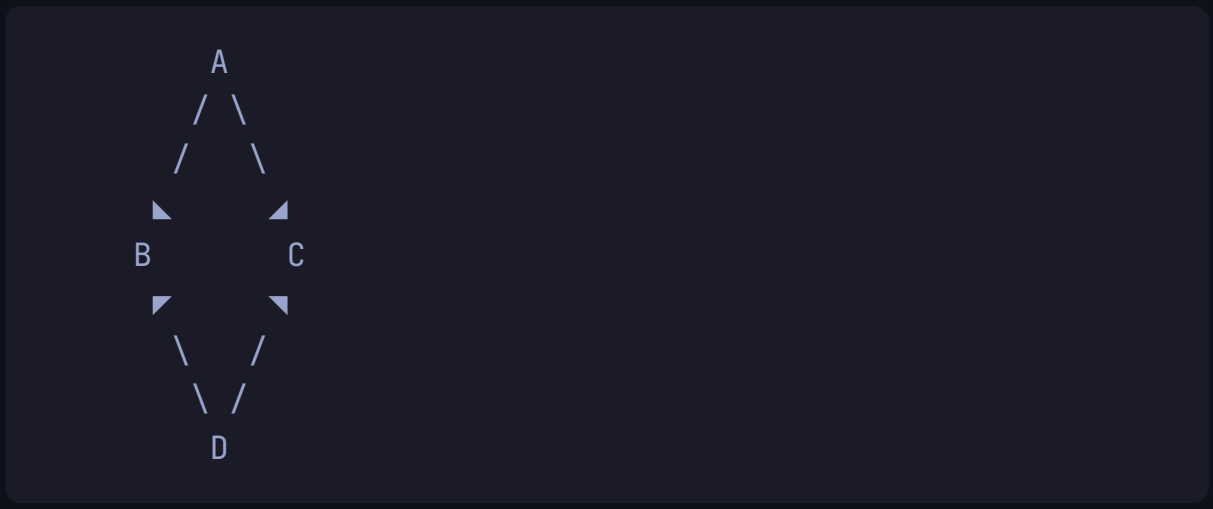
Class Test 2

1. Diamond Problem

For the following diagram, let there is a function in class "A" named `display()` which you want to invoke by the class "E" object. Does this scenario create any ambiguity? Apply the knowledge of inheritance and justify your answer with correct code implementation.



Ans.: Here is the correct code implementation of the given diamond problem:

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

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

```
class B : virtual public A
{
};

class C : virtual public A
{
};

class D : public B, public C
{
};

int main()
{
    D d;
    d.A::display();
    return 0;
}
```

Output: The code yields the following output in the terminal:

```
Class A
```

2. Theory-based Question

Explain different types of **Inheritance** with pseudo-code implementation.

Ans.:

C++ supports **five types** of inheritance:

- Single inheritance
- Multiple inheritance
- Multi-level inheritance
- Hierarchical inheritance
- Hybrid inheritance

1. **Single inheritance:** This is defined as the inheritance in which a derived class is inherited from the **only one base class**.

```
class A
{
    // ...
};

class B : public A
{
    // ...
};
```

2. **Multiple inheritance:** Multiple inheritance is the process of deriving a new class that inherits the attributes from two or more classes.

```
class A
{
    // ...
};
```

```
class B
{
    // ...
};

class C : public A, public B
{
    // ...
};
```

3. **Multi-level inheritance:** When one class inherits another class which is further inherited by another class, it is known as **multi-level inheritance** in C++. Inheritance is **transitive** so the last derived class acquires all the members of all its base classes.

```
class A
{
    // ...
};

class B: public A
{
    // ...
};

class C: public B
{
    // ...
};
```

4. **Hierarchical inheritance:** Hierarchical inheritance is defined as the process of **deriving more than one class from a base class**.

```
class A
{
    // ...
```

```
}
class B : public A
{
    // ...
}
class C : public A
{
    // ...
}
class D : public A
{
    // ...
}
```

5. **Hybrid inheritance:** Hybrid inheritance is a combination of **more than one type of inheritance**.

```
class A
{
    // ...
};

class B : public A
{
    // ...
};

class C
{
    // ...
};

class D : public B, public C
{
    // ...
};
```

3. Problem Solving

Consider an application that calculate different areas. The base class `Area` has a `calculate()` method that is overriden by each derived shape class (**Circle**, **Rectangle**, **Triangle**). Demonstrate runtime polymorphism in the `calculate()` method.

Hints:

```
Area of a circle = 3.14 × r ^ 2
Area of a rectangle = l × w
Area of a triangle = 0.5 × b × h
```

Ans.: Here is the C++ program of a class that satisfies the conditions above:

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

class Area
{
public:
    virtual void calculate()
    {
        cout << "Can't override." << endl;
    }
};

class Circle : public Area
{
protected:
    double radius = 5;

public:
    void calculate()
    {
```

```
        cout << 3.14 * radius * radius << endl;
    }
};

class Rectangle : public Area
{
protected:
    double length = 5, width = 10;

public:
    void calculate()
    {
        cout << length * width << endl;
    }
};

class Triangle : public Area
{
protected:
    double base = 10, height = 20;

public:
    void calculate()
    {
        cout << .5 * base * height << endl;
    }
};

int main()
{
    Area *ptr;
    Circle circle;
    ptr = &circle;
    ptr->calculate();

    Rectangle rectangle;
    ptr = &rectangle;
    ptr->calculate();
```

```
    Triangle triangle;
    ptr = &triangle;
    ptr->calculate();

    return 0;
}
```

Output: The code yields the following output in the terminal:

```
78.5
50
100
```