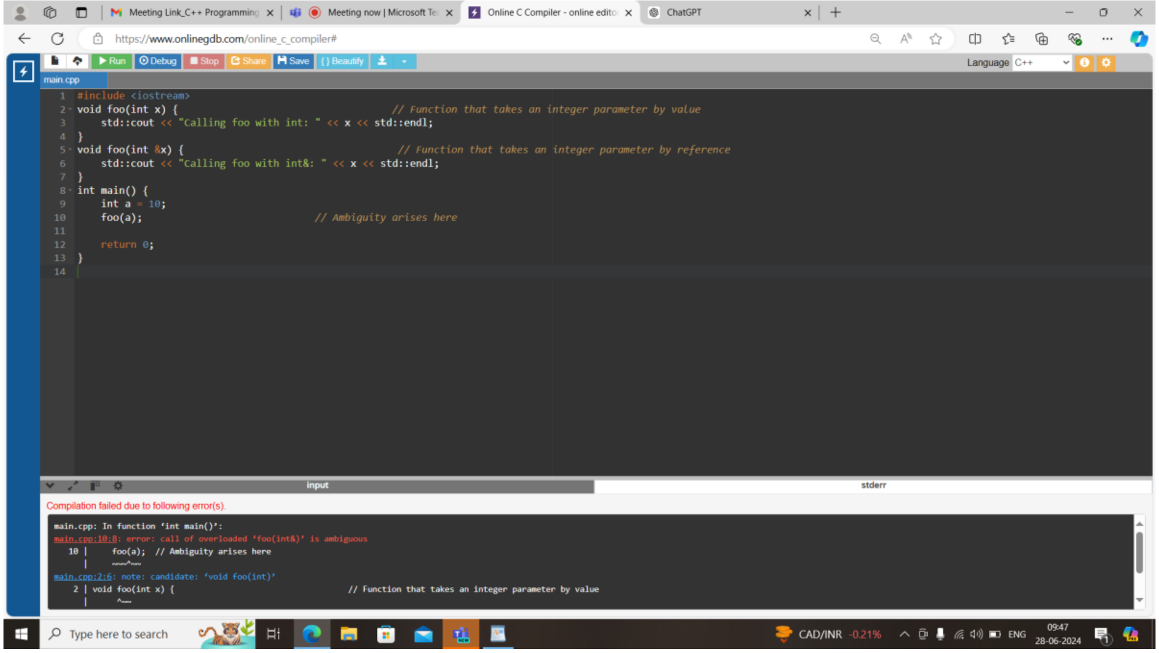
Call by value and Call by Reference Ambiguity :-

```
#include <iostream>

void foo(int x) {                                   // Function that takes an integer parameter by value

    std::cout << "Calling foo with int: " << x << std::endl;

}

void foo(int &x) {                                  // Function that takes an integer parameter by reference

    std::cout << "Calling foo with int&: " << x << std::endl;

}

int main() {

    int a = 10;

    foo(a);                                   // Ambiguity arises here

    return 0;

}
```

OUTPUT :-



```
#include <iostream>
void foo(int x) {                           // Function that takes an integer parameter by value
    std::cout << "Calling foo with int: " << x << std::endl;
}
void foo(int &x) {                          // Function that takes an integer parameter by reference
    std::cout << "Calling foo with int&: " << x << std::endl;
}
int main() {
    int a = 10;
    foo(a);                          // Ambiguity arises here

    return 0;
}
```

Compilation failed due to following error(s).

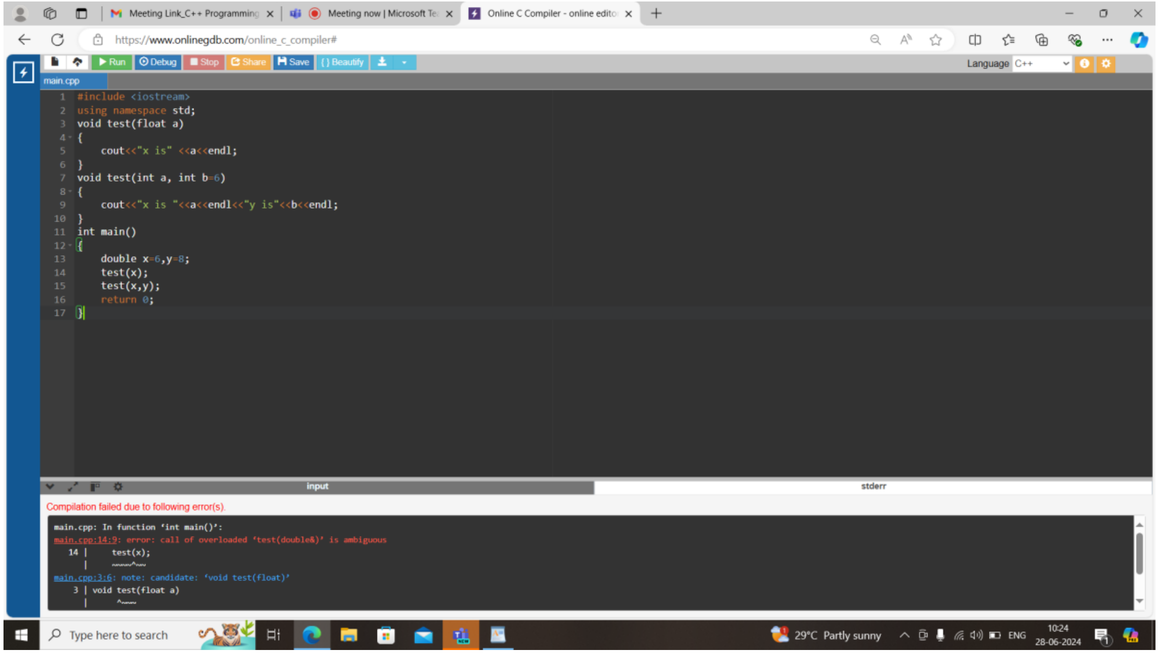
```
main.cpp: In function ‘int main()’:
main.cpp:10:8: error: call of overloaded ‘foo(int&)’ is ambiguous
   10 |     foo(a);  // Ambiguity arises here
      |     ~~~^~~
main.cpp:2:6: note: candidate: ‘void foo(int)’
    2 | void foo(int x) {                              // Function that takes an integer parameter by value
      |      ^~~
```

Meeting Link_C++ Programming
Meeting now | Microsoft Te
Online C Compiler - online edito
ChatGPT
https://www.onlinegdb.com/online_c_compiler#
Run
Debug
Stop
Share
Save
{ } Beautify
Language C++
main.cpp
input
stderr
Type here to search
CAD/INR -0.21%
ENG
09:47
28-06-2024

AMBIGUITY ERROR :-

```
#include <iostream>

using namespace std;

void test(float a)

{

        cout<<"x is" <<a<<endl;

}

void test(int a, int b=6)

{

        cout<<"x is "<<a<<endl<<"y is"<<b<<endl;

}

int main()

{

        double x=6,y=8;

        test(x);

        test(x,y);

        return 0;

}
```

OUTPUT :-


Meeting Link_C++ Programming
Meeting now | Microsoft Te
Online C Compiler - online editor
https://www.onlinegdb.com/online_c_compiler#
Run
Debug
Stop
Share
Save
{ } Beautify
Language C++
main.cpp
input
stderr
Type here to search
29°C Partly sunny
ENG
10:24
28-06-2024

```
#include <iostream>
using namespace std;
void test(float a)
{
    cout<<"x is" <<a<<endl;
}
void test(int a, int b=6)
{
    cout<<"x is "<<a<<endl<<"y is"<<b<<endl;
}
int main()
{
    double x=6,y=8;
    test(x);
    test(x,y);
    return 0;
}
```

Compilation failed due to following error(s).

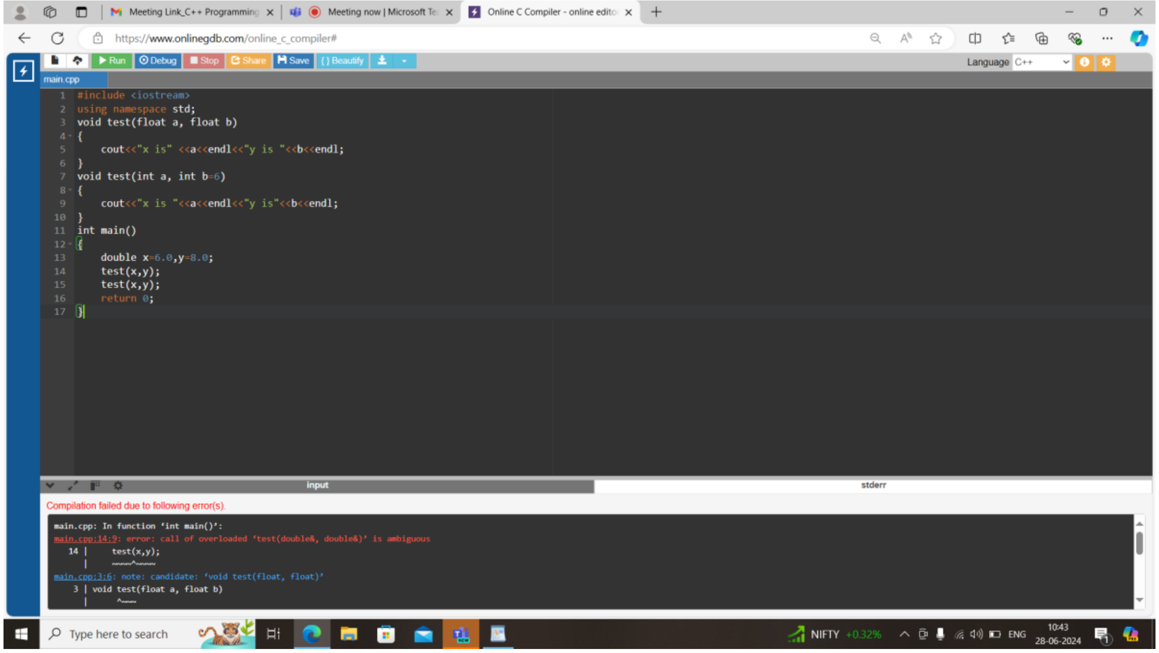
```
main.cpp: In function ‘int main()’:
main.cpp:14:9: error: call of overloaded ‘test(double&)’ is ambiguous
   14 |     test(x);
      |     ~~~~^~~
main.cpp:3:6: note: candidate: ‘void test(float)’
    3 | void test(float a)
      |      ^~~~
```

AMBIGUITY ERROR :-

```
#include <iostream>

using namespace std;

void test(float a, float b)

{

        cout<<"x is" <<a<<endl<<"y is "<<b<<endl;

}

void test(int a, int b=6)

{

        cout<<"x is "<<a<<endl<<"y is"<<b<<endl;

}

int main()

{

        double x=6.0,y=8.0;

        test(x,y);

        test(x,y);
```

```
        return 0;
}
```

OUTPUT :-



Meeting Link_C++ Programming
Meeting now | Microsoft Te
Online C Compiler - online editor
https://www.onlinegdb.com/online_c_compiler#
Run
Debug
Stop
Share
Save
{ } Beautify
Language C++
main.cpp
input
stderr
Type here to search
NIFTY +0.32%
ENG
10:43
28-06-2024

```
#include <iostream>
using namespace std;
void test(float a, float b)
{
    cout<<"x is" <<a<<endl<<"y is "<<b<<endl;
}
void test(int a, int b=6)
{
    cout<<"x is "<<a<<endl<<"y is"<<b<<endl;
}
int main()
{
    double x=6.0,y=8.0;
    test(x,y);
    test(x,y);
    return 0;
}
```

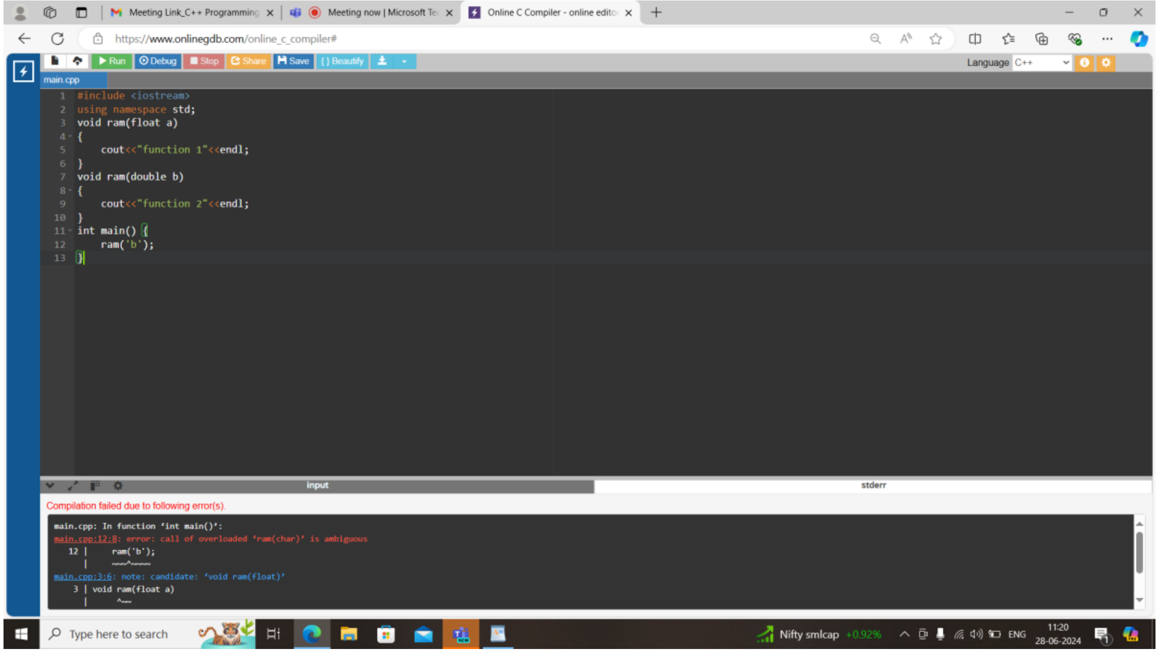
Compilation failed due to following error(s).

```
main.cpp: In function ‘int main()’:
main.cpp:14:9: error: call of overloaded ‘test(double&, double&)’ is ambiguous
   14 |     test(x,y);
      |     ~~~~^~~~~
main.cpp:3:6: note: candidate: ‘void test(float, float)’
    3 | void test(float a, float b)
      |      ^~~~
```

AMBIGUITY ERROR :-

```
#include <iostream>

using namespace std;

void ram(float a)

{

        cout<<"function 1"<<endl;

}

void ram(double b)

{

        cout<<"function 2"<<endl;

}

int main() {

        ram('b');

}
```

OUTPUT:-



```
#include <iostream>
using namespace std;
void ram(float a)
{
    cout<<"function 1"<<endl;
}
void ram(double b)
{
    cout<<"function 2"<<endl;
}
int main() {
    ram('b');
}
```

Compilation failed due to following error(s).

```
main.cpp: In function ‘int main()’:
main.cpp:12:8: error: call of overloaded ‘ram(char)’ is ambiguous
   12 |     ram('b');
      |     ~~~^~~~~
main.cpp:3:6: note: candidate: ‘void ram(float)’
    3 | void ram(float a)
      |      ^~~
```

Problem Statement: Distance Calculation Using Operator Overloading

You are required to implement a program that calculates distances using operator overloading in C++. The program should be able to perform the following operations on distances:

Addition of Distances:

Implement an addition operator (+) that adds two distances together.

The distance should be represented in feet and inches.

Subtraction of Distances:

Implement a subtraction operator (-) that subtracts one distance from another.

Ensure that the subtraction operation handles cases where the result may involve negative values or borrowing (like in subtraction of inches).

Comparison of Distances:

Implement comparison operators (==, !=, <, >, <=, >=) to compare distances based on their total length (combined feet and inches).

Use these operators to determine which distance is greater, less than, or equal to another.

Requirements:

Distance Class: Implement a Distance class with appropriate member variables (feet and inches).

Constructors: Implement constructors to initialize distances.

Member Functions: Implement member functions for display and any other necessary operations.

Operator Overloading: Overload the necessary operators (+, -, ==, !=, <, >, <=, >=) inside the Distance class to perform the specified operations.

Testing: Create a main() function to test the implemented Distance class and its operator overloading functionality. Test various scenarios including addition, subtraction, and comparison of distance.

```
#include <iostream>

class Distance {

private:

    int feet;

    int inches;

public:

    Distance() : feet(0), inches(0) {}                                    // Constructors

    Distance(int ft, int in) : feet(ft), inches(in) {}

    void display() const {                                       // Display function

        std::cout << "Distance: " << feet << " feet " << inches << " inches" << std::endl;

    }

    Distance operator+(const Distance& d2) const {                          // Overloading +
operator for addition

        int totalFeet = feet + d2.feet;

        int totalInches = inches + d2.inches;

        if (totalInches >= 12) {

            totalFeet++;

            totalInches -= 12;

        }

        return Distance(totalFeet, totalInches);

    }

    Distance operator-(const Distance& d2) const {                      // Overloading - operator
for subtraction
```

```
        int totalFeet = feet - d2.feet;

        int totalInches = inches - d2.inches;

        if (totalInches < 0) {

            totalFeet--;

            totalInches += 12;

        }

        return Distance(totalFeet, totalInches);

    }

    bool operator==(const Distance& d2) const {                                    // Overloading comparison operators

        return (feet == d2.feet && inches == d2.inches);

    }

    bool operator!=(const Distance& d2) const {

        return !(*this == d2);

    }

    bool operator<(const Distance& d2) const {

        return (feet < d2.feet) || (feet == d2.feet && inches < d2.inches);

    }

    bool operator>(const Distance& d2) const {

        return !(*this < d2) && !(*this == d2);

    }

    bool operator<=(const Distance& d2) const {

        return (*this < d2) || (*this == d2);

    }

    bool operator>=(const Distance& d2) const {

        return !(*this < d2);

    }

};
```

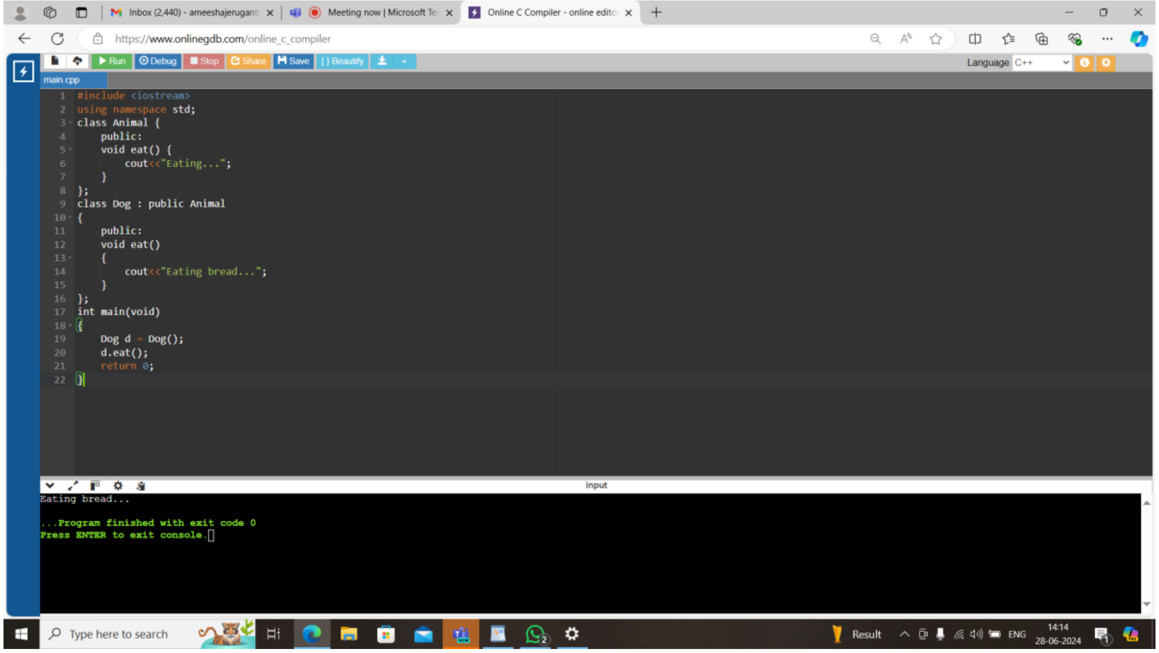
```
int main() {

    Distance d1(5, 10);

    Distance d2(3, 8);

    // Addition

    Distance sum = d1 + d2;

    sum.display();                          // Output: Distance: 9 feet 6 inches

    // Subtraction

    Distance diff = d1 - d2;

    diff.display();                         // Output: Distance: 2 feet 2 inches

    // Comparisons

    if (d1 == d2)

        std::cout << "d1 is equal to d2" << std::endl;

    if (d1 != d2)

        std::cout << "d1 is not equal to d2" << std::endl;

    if (d1 < d2)

        std::cout << "d1 is less than d2" << std::endl;

    if (d1 > d2)

        std::cout << "d1 is greater than d2" << std::endl;

    if (d1 <= d2)

        std::cout << "d1 is less than or equal to d2" << std::endl;

    if (d1 >= d2)

        std::cout << "d1 is greater than or equal to d2" << std::endl;

    return 0;

}
```

Function Overriding :-

```
#include <iostream>

using namespace std;
```

```
class Animal {
        public:
        void eat() {
                cout<<"Eating...";
        }
};
class Dog : public Animal
{
        public:
        void eat()
        {
                cout<<"Eating bread...";
        }
};
int main(void)
{
        Dog d = Dog();
        //d.eat();
        d.Animal::eat();
        return 0;
}
```

OUTPUT :-


Inbox (2,440) - ameeshajerugant
Meeting now | Microsoft Te
Online C Compiler - online edito
https://www.onlinegdb.com/online_c_compiler
Run
Debug
Stop
Share
Save
{ } Beautify
Language C++
main.cpp
input
Type here to search
Result
ENG
14:14
28-06-2024

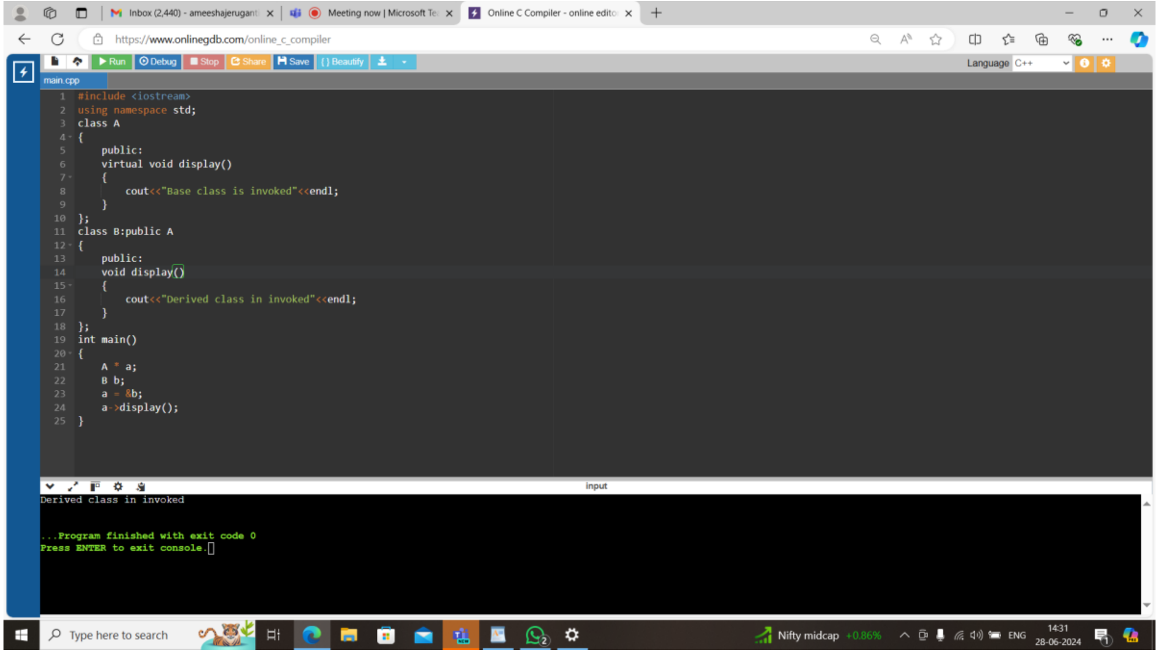
```
#include <iostream>
using namespace std;
class Animal {
    public:
    void eat() {
        cout<<"Eating...";
    }
};
class Dog : public Animal
{
    public:
    void eat()
    {
        cout<<"Eating bread...";
    }
};
int main(void)
{
    Dog d = Dog();
    d.eat();
    return 0;
}
```

```
Eating bread...

...Program finished with exit code 0
Press ENTER to exit console.
```

Virtual Function :-

```
#include <iostream>

using namespace std;

class A

{

        public:

        virtual void display()

        {

                cout<<"Base class is invoked"<<endl;

        }

};

class B:public A

{

        public:

        void display()

        {
                cout<<"Derived class in invoked"<<endl;

        }

};

int main()

{

        A * a;

        B b;

        a = &b;

        a->display();

}
```

OUTPUT :-



Inbox (2,440) - ameeshajerugant
Meeting now | Microsoft Te
Online C Compiler - online edito
https://www.onlinegdb.com/online_c_compiler
Run
Debug
Stop
Share
Save
{ } Beautify
Language C++
main.cpp
input
Type here to search
Nifty midcap +0.86%
ENG
14:31
28-06-2024

```
#include <iostream>
using namespace std;
class A
{
    public:
    virtual void display()
    {
        cout<<"Base class is invoked"<<endl;
    }
};
class B:public A
{
    public:
    void display()
    {
        cout<<"Derived class in invoked"<<endl;
    }
};
int main()
{
    A * a;
    B b;
    a = &b;
    a->display();
}
```

```
Derived class in invoked


...Program finished with exit code 0
Press ENTER to exit console.
```

Problem Statement: Shape Area Calculator Using Method Overloading

You are required to implement a program that calculates the area of different shapes using compile-time polymorphism (method overloading) in C++. The program should support calculation of areas for the following shapes:

Rectangle

Circle

Triangle

Requirements:

Shape Class: Implement a Shape class as a base class with virtual functions to calculate and display the area of each shape.

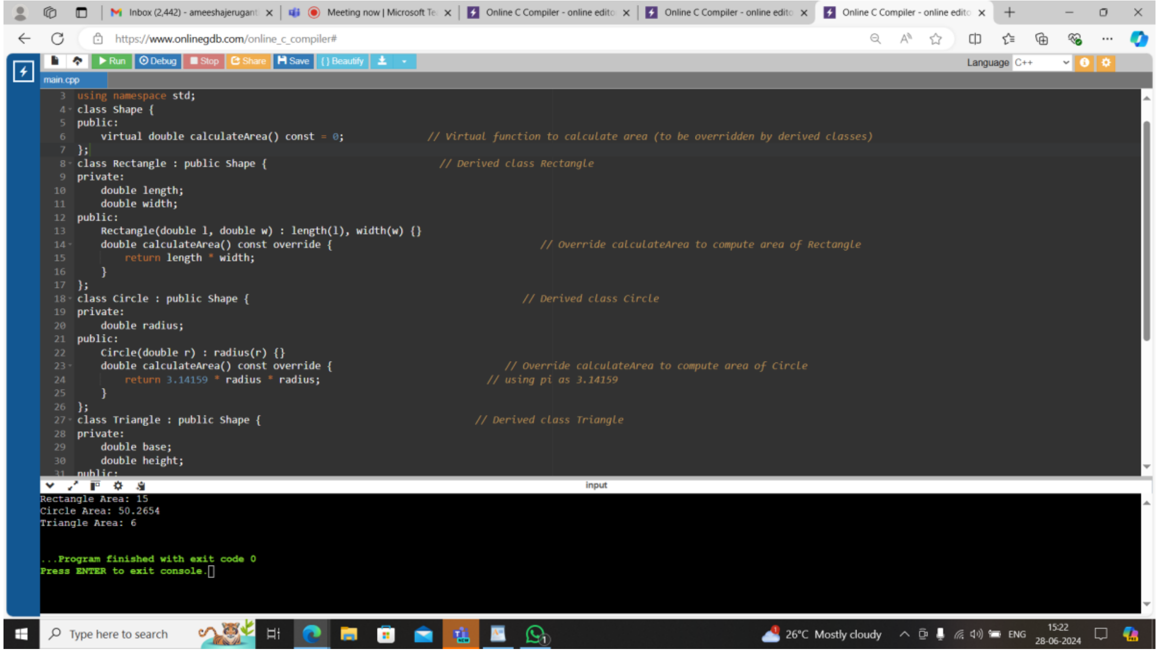
Derived Classes: Implement derived classes Rectangle, Circle, and Triangle, inheriting from Shape, each with overridden functions to calculate and display their respective areas.

Method Overloading: Use method overloading in the Shape class to define multiple calculateArea functions, each specific to one shape.

Input and Output: Implement a main() function to test the implemented classes by creating instances of each shape, inputting dimensions, and displaying their calculated areas.

```
#include <iostream>

using namespace std;

class Shape {

public:

    virtual double calculateArea() const = 0;                   // Virtual function to calculate area (to be overridden by derived classes)

};

class Rectangle : public Shape {                                // Derived class Rectangle

private:

    double length;

    double width;

public:

    Rectangle(double l, double w) : length(l), width(w) {}

    double calculateArea() const override {                     // Override calculateArea to compute area of Rectangle

        return length * width;

    }

};

class Circle : public Shape {                                   // Derived class Circle

private:
    double radius;
public:
    Circle(double r) : radius(r) {}
    double calculateArea() const override {                                   // Override
calculateArea to compute area of Circle
        return 3.14159 * radius * radius;                                    // using pi as 3.14159
    }
};
class Triangle : public Shape {                                              // Derived class Triangle
private:
    double base;
    double height;
public:
    Triangle(double b, double h) : base(b), height(h) {}
        double calculateArea() const override {                              // Override
calculateArea to compute area of Triangle
        return 0.5 * base * height;
    }
};
int main() {
    Rectangle rect(5.0, 3.0);
    Circle circle(4.0);
    Triangle triangle(6.0, 2.0);
        cout << "Rectangle Area: " << rect.calculateArea() << endl;                  // Display
areas
    cout << "Circle Area: " << circle.calculateArea() << endl;
    cout << "Triangle Area: " << triangle.calculateArea() << endl;
    return 0;
}
```

OUTPUT :-


Inbox (2,442) - ameeshajerugant
Meeting now | Microsoft Te
Online C Compiler - online edito
Online C Compiler - online edito
Online C Compiler - online edito
https://www.onlinegdb.com/online_c_compiler#
Run
Debug
Stop
Share
Save
{ } Beautify
Language C++
main.cpp
input
Type here to search
26°C Mostly cloudy
ENG
15:22
28-06-2024

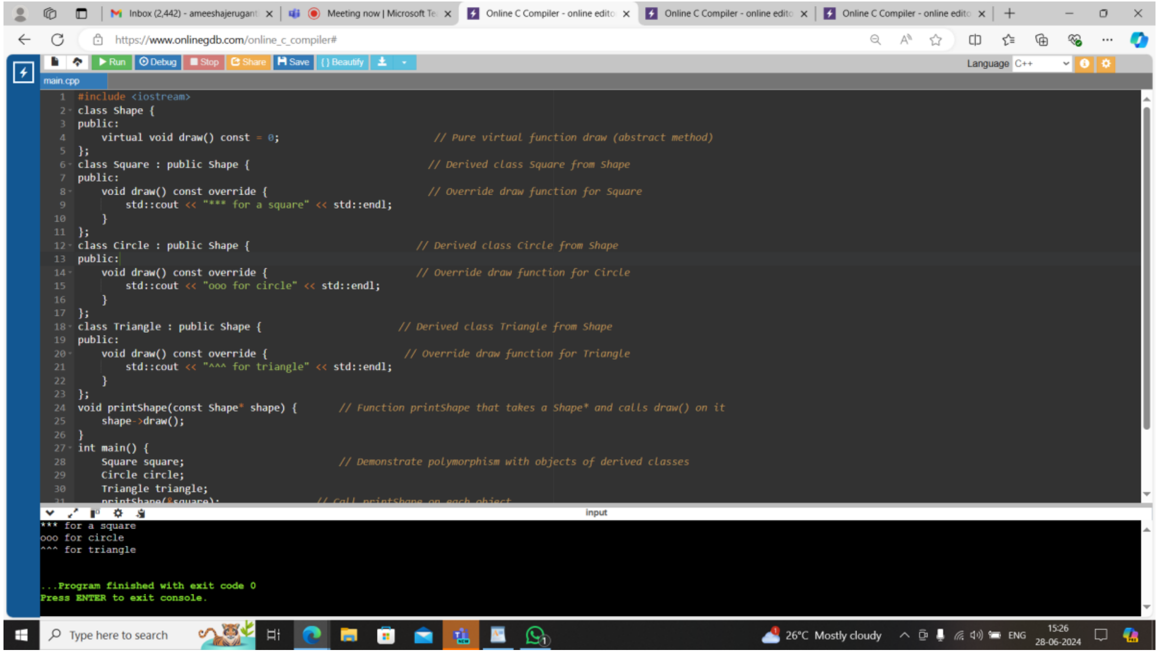
```
using namespace std;
class Shape {
public:
    virtual double calculateArea() const = 0;            // Virtual function to calculate area (to be overridden by derived classes)
};
class Rectangle : public Shape {                          // Derived class Rectangle
private:
    double length;
    double width;
public:
    Rectangle(double l, double w) : length(l), width(w) {}
    double calculateArea() const override {                          // Override calculateArea to compute area of Rectangle
        return length * width;
    }
};
class Circle : public Shape {                                       // Derived class Circle
private:
    double radius;
public:
    Circle(double r) : radius(r) {}
    double calculateArea() const override {                     // Override calculateArea to compute area of Circle
        return 3.14159 * radius * radius;                    // using pi as 3.14159
    }
};
class Triangle : public Shape {                            // Derived class Triangle
private:
    double base;
    double height;
public:
```

```
Rectangle Area: 15
Circle Area: 50.2654
Triangle Area: 6


...Program finished with exit code 0
Press ENTER to exit console.
```

Create a base class Shape with a pure virtual function draw() that has no implementation. Derive classes Square, Circle, and Triangle from Shape. Each derived class should override draw() to provide its specific drawing behavior (e.g., printing "" for square, "OOO" for circle, etc.). Write a function printShape(Shape* shape) that takes a base class pointer and calls draw() on it. Demonstrate polymorphism by creating objects of the derived classes, storing them in a Shape* array, and calling printShape() on each element.

```
#include <iostream>

class Shape {

public:

    virtual void draw() const = 0;                                    // Pure virtual function draw
(abstract method)

};

class Square : public Shape {                                        // Derived class Square from Shape

public:

    void draw() const override {                                     // Override draw function for Square

        std::cout << "*** for a square" << std::endl;

    }
};
class Circle : public Shape {                          // Derived class Circle from Shape
public:
    void draw() const override {                        // Override draw function for Circle
        std::cout << "ooo for circle" << std::endl;
    }
};
class Triangle : public Shape {                      // Derived class Triangle from Shape
public:
    void draw() const override {                      // Override draw function for Triangle
        std::cout << "^^^ for triangle" << std::endl;
    }
};
void printShape(const Shape* shape) {       // Function printShape that takes a Shape* and calls draw() on it
    shape->draw();
}
int main() {
    Square square;                               // Demonstrate polymorphism with objects of derived classes
    Circle circle;
    Triangle triangle;
    printShape(&square);                 // Call printShape on each object
    printShape(&circle);
    printShape(&triangle);
    return 0;
}
```

OUTPUT :-



```
#include <iostream>
class Shape {
public:
    virtual void draw() const = 0;                              // Pure virtual function draw (abstract method)
};
class Square : public Shape {                                  // Derived class Square from Shape
public:
    void draw() const override {                               // Override draw function for Square
        std::cout << "*** for a square" << std::endl;
    }
};
class Circle : public Shape {                                // Derived class Circle from Shape
public:
    void draw() const override {                             // Override draw function for Circle
        std::cout << "ooo for circle" << std::endl;
    }
};
class Triangle : public Shape {                           // Derived class Triangle from Shape
public:
    void draw() const override {                            // Override draw function for Triangle
        std::cout << "^^^ for triangle" << std::endl;
    }
};
void printShape(const Shape* shape) {       // Function printShape that takes a Shape* and calls draw() on it
    shape->draw();
}
int main() {
    Square square;                              // Demonstrate polymorphism with objects of derived classes
    Circle circle;
    Triangle triangle;
    printShape(&square);                     // Call printShape on each object
```

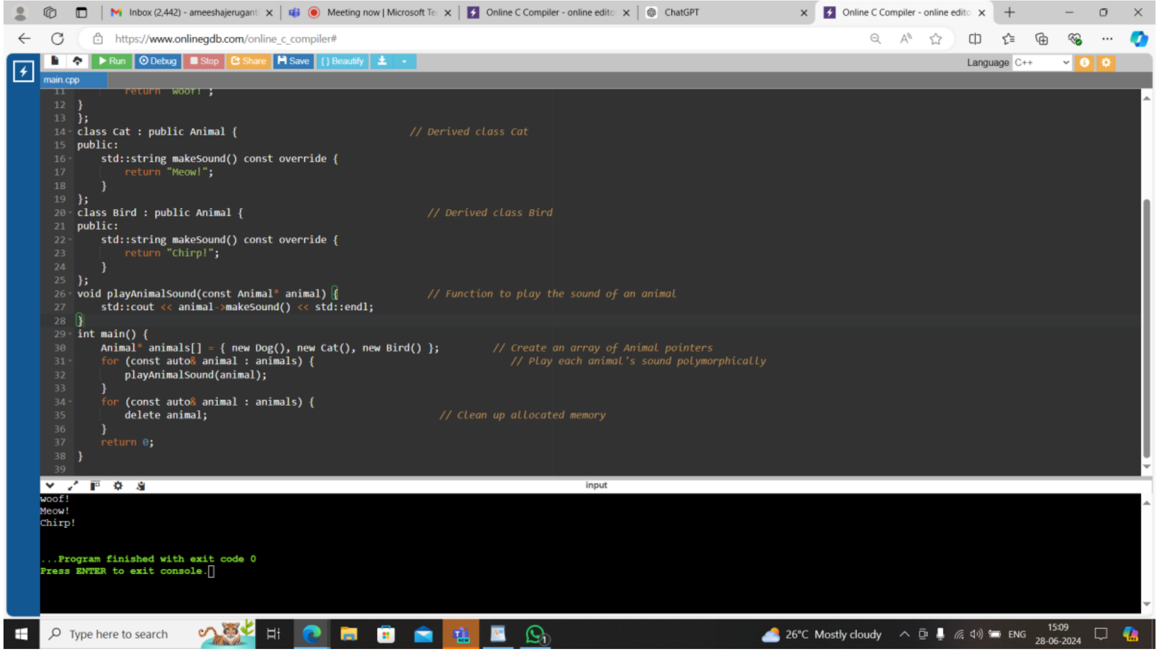
```
*** for a square
ooo for circle
^^^ for triangle


...Program finished with exit code 0
Press ENTER to exit console.
```

Design a base class Animal with a pure virtual function makeSound() that returns a string representing the animal's sound. Derive classes like Dog, Cat, and Bird from Animal, each overriding makeSound() with the appropriate sound ("Woof!", "Meow!", "Chirp!"). Create a function playAnimalSound(Animal* animal) that takes an Animal pointer and calls makeSound(). Populate an Animal* array with various animal objects and use playAnimalSound() to hear their sounds polymorphically.

```
#include <iostream>

#include <string>

class Animal {

public:

    virtual std::string makeSound() const = 0;                    // Pure virtual function

    virtual ~Animal() {}                                                  // Virtual destructor for proper
polymorphic behavior

};

class Dog : public Animal {                              // Derived class Dog

public:

    std::string makeSound() const override {

        return "woof!";
}
};
class Cat : public Animal {                                         // Derived class Cat
public:
    std::string makeSound() const override {
        return "Meow!";
    }
};
class Bird : public Animal {                                           // Derived class Bird
public:
    std::string makeSound() const override {
        return "Chirp!";
    }
};
void playAnimalSound(const Animal* animal) {                    // Function to play the sound of an animal
    std::cout << animal->makeSound() << std::endl;
}
int main() {
    Animal* animals[] = { new Dog(), new Cat(), new Bird() };           // Create an array of Animal pointers
    for (const auto& animal : animals) {                                       // Play each animal's sound polymorphically
        playAnimalSound(animal);
    }
    for (const auto& animal : animals) {
        delete animal;                                                      // Clean up allocated memory
    }
    return 0;
```

}

OUTPUT :-



Inbox (2,442) - ameeshajerugant
Meeting now | Microsoft Te
Online C Compiler - online edito
ChatGPT
Online C Compiler - online edito
https://www.onlinegdb.com/online_c_compiler#
Run
Debug
Stop
Share
Save
{ } Beautify
Language C++
main.cpp
input
Type here to search
26°C Mostly cloudy
ENG
15:09
28-06-2024

```cpp
        return "Woof!";
    }
};
class Cat : public Animal {                              // Derived class Cat
public:
    std::string makeSound() const override {
        return "Meow!";
    }
};
class Bird : public Animal {                              // Derived class Bird
public:
    std::string makeSound() const override {
        return "Chirp!";
    }
};
void playAnimalSound(const Animal* animal) {              // Function to play the sound of an animal
    std::cout << animal->makeSound() << std::endl;
}
int main() {
    Animal* animals[] = { new Dog(), new Cat(), new Bird() };         // Create an array of Animal pointers
    for (const auto& animal : animals) {                               // Play each animal's sound polymorphically
        playAnimalSound(animal);
    }
    for (const auto& animal : animals) {
        delete animal;                                    // Clean up allocated memory
    }
    return 0;
}
```

```
Woof!
Meow!
Chirp!


...Program finished with exit code 0
Press ENTER to exit console.
```