

## Polymorphism:

**Design a class hierarchy for a simple graphic editor with base class Shape and derived classes Circle, Rectangle, and Triangle. Implement a virtual function draw() in the base class and override it in the derived classes. Write a function that takes a Shape\* and calls its draw() method.**

```
#include <iostream>
```

```
class Shape {                                     // Base class Shape
```

```
public:
```

```
    virtual void draw() const {
```

```
        std::cout << "Drawing a Shape" << std::endl;
```

```
    }
```

```
    virtual ~Shape() {}
```

```
    // Virtual destructor for
```

```
    polymorphic behavior
```

```
};
```

```
class Circle : public Shape {                     // Derived class Circle
```

```
public:
```

```
    void draw() const override {
```

```
        std::cout << "Drawing a Circle" << std::endl;
```

```
    }
```

```
};
```

```
class Rectangle : public Shape {                  // Derived class Rectangle
```

```
public:
```

```
    void draw() const override {
```

```
        std::cout << "Drawing a Rectangle" << std::endl;
```

```
    }
```

```
};
```

```
class Triangle : public Shape {                   // Derived class Triangle
```

```
public:
```

```
    void draw() const override {
```

```

        std::cout << "Drawing a Triangle" << std::endl;

    }

};

void drawShape(const Shape* shape) {                                // Function that takes
a Shape* and calls its draw() method

    shape->draw();

}

int main() {                                                        // Main function to test the
functionality

    Circle circle;

    Rectangle rectangle;

    Triangle triangle;

    drawShape(&circle);                                            // Polymorphic behavior through pointers

    drawShape(&rectangle);

    drawShape(&triangle);

    return 0;

}

```

OUTPUT :-

The screenshot shows a web browser window with the URL [https://www.onlinegdb.com/online\\_c\\_compiler#](https://www.onlinegdb.com/online_c_compiler#). The code editor displays the C++ code from the previous block. The output window shows the following text:

```

Drawing a Circle
Drawing a Rectangle
Drawing a Triangle
...Program finished with exit code 0
Press ENTER to exit console.

```

The Windows taskbar at the bottom shows the date and time as 13:50 on 04-07-2024, and the weather as 33°C Partly sunny.

### Static Members:

Create a class **Account** that has a static data member **totalAccounts** to keep track of the number of accounts created. Implement necessary constructors and destructors to update **totalAccounts**. Write a function to display the total number of accounts.

```
#include <iostream>

class Account {
private:
    static int totalAccounts;           // Static data member to keep track of total accounts
    int accountNumber;

public:
    Account() {                         // Constructor
        totalAccounts++;                // Increment totalAccounts when a new object is
created
        accountNumber = totalAccounts; // Assign a unique account number
    }

    ~Account() {                        // Destructor
        totalAccounts--;                // Decrement totalAccounts when an object is
destroyed
    }

    static void displayTotalAccounts() {
        std::cout << "Total number of accounts: " << totalAccounts << std::endl;
    }

    int getAccountNumber() const {
        return accountNumber;
    }
};

int Account::totalAccounts = 0;

int main() {
```

```

Account acc1;

Account acc2;

Account acc3;

Account acc4;

Account::displayTotalAccounts();           // Display total number of accounts

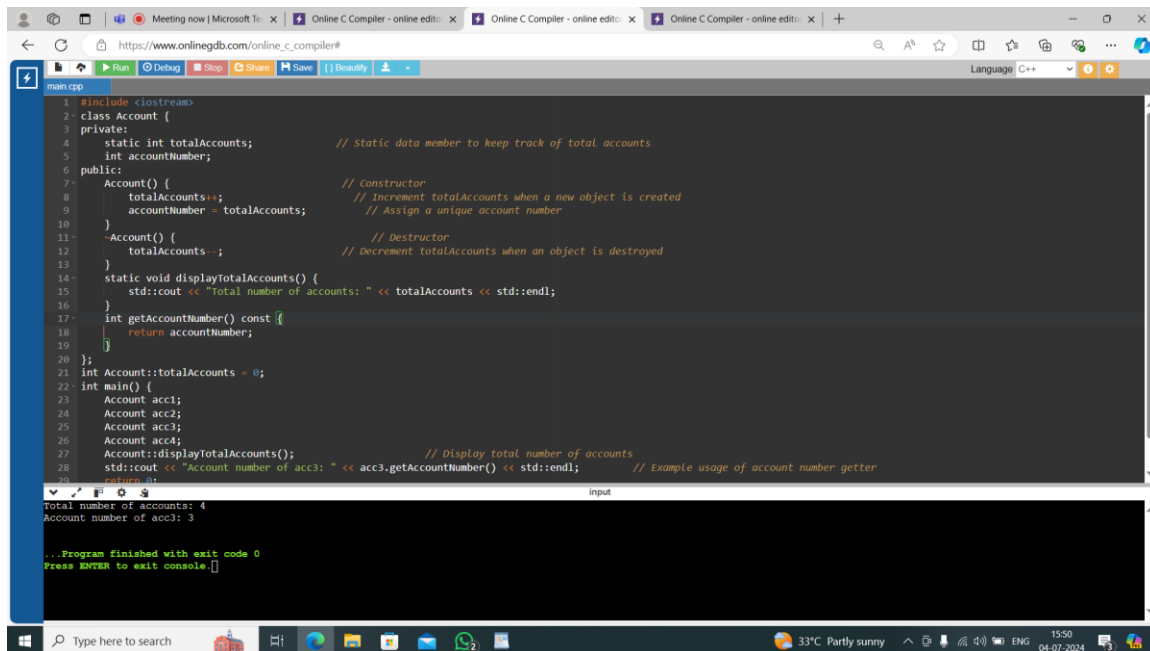
std::cout << "Account number of acc3: " << acc3.getAccountNumber() << std::endl;    //
Example usage of account number getter

return 0;

}

```

OUTPUT :-



The screenshot shows a web browser with multiple tabs of 'Online C Compiler - online editor'. The active tab displays a C++ program in a dark-themed editor. The code defines an 'Account' class with a static member 'totalAccounts', a constructor that increments 'totalAccounts', a destructor that decrements it, a static method 'displayTotalAccounts()', and a 'getAccountNumber()' method. The 'main' function creates four 'Account' objects (acc1, acc2, acc3, acc4), calls 'displayTotalAccounts()', and prints the account number of 'acc3'. The output window at the bottom shows the execution results: 'Total number of accounts: 4' and 'Account number of acc3: 3'. The program finished with exit code 0.

```

main.cpp
1 #include <iostream>
2 class Account {
3 private:
4     static int totalAccounts;           // Static data member to keep track of total accounts
5     int accountNumber;
6 public:
7     Account() {                         // Constructor
8         totalAccounts++;               // Increment totalAccounts when a new object is created
9         accountNumber = totalAccounts; // Assign a unique account number
10    }
11    ~Account() {                        // Destructor
12        totalAccounts--;               // Decrement totalAccounts when an object is destroyed
13    }
14    static void displayTotalAccounts() {
15        std::cout << "Total number of accounts: " << totalAccounts << std::endl;
16    }
17    int getAccountNumber() const {
18        return accountNumber;
19    }
20 };
21 int Account::totalAccounts = 0;
22 int main() {
23     Account acc1;
24     Account acc2;
25     Account acc3;
26     Account acc4;
27     Account::displayTotalAccounts();    // Display total number of accounts
28     std::cout << "Account number of acc3: " << acc3.getAccountNumber() << std::endl; // Example usage of account number getter
29     return 0;
30 }

```

Output:

```

Total number of accounts: 4
Account number of acc3: 3

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

```

## Friend Functions:

Implement a class Box that has private data members length, breadth, and height. Write a friend function volume() that calculates and returns the volume of the box. Create objects of Box and use the friend function to compute their volumes.

```
#include <iostream>
```

```
class Box {
```

```
private:
```

```

float length;

float breadth;

float height;

public:

    Box(float l, float b, float h) {                // Constructor

        length = l;

        breadth = b;

        height = h;

    }

    friend double volume(Box b);                    // Friend function declaration

};

double volume(Box b) {                             // Definition of the friend
function volume

    return b.length * b.breadth * b.height;

}

int main() {                                        // Main function to demonstrate
the usage

    Box box1(3.0, 8.0, 5.0);                        // Creating objects of Box

    Box box2(5.0, 5.0, 5.0);

    float vol1 = volume(box1);                      // Calculating volumes using the
friend function

    float vol2 = volume(box2);

    std::cout << "Volume of box1: " << vol1 << std::endl;    // Displaying the volumes

    std::cout << "Volume of box2: " << vol2 << std::endl;

    return 0;

}

```

OUTPUT :-

```
1 #include <iostream>
2 class Box {
3 private:
4     float length;
5     float breadth;
6     float height;
7 public:
8     Box(float l, float b, float h) {           // constructor
9         length = l;
10        breadth = b;
11        height = h;
12    }
13    friend double volume(Box b);               // friend function declaration
14 };
15 double volume(Box b) {                       // Definition of the friend function volume
16     return b.length * b.breadth * b.height;
17 }
18 int main() {
19     Box box1(3.0, 8.0, 5.0);                  // Main function to demonstrate the usage
20     Box box2(5.0, 5.0, 5.0);                  // Creating objects of Box
21
22     float vol1 = volume(box1);                // Calculating volumes using the friend function
23     float vol2 = volume(box2);
24
25     std::cout << "Volume of box1: " << vol1 << std::endl;    // Displaying the volumes
26     std::cout << "Volume of box2: " << vol2 << std::endl;
27     return 0;
28 }
```

Volume of box1: 120  
Volume of box2: 125

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

## Templates:

Write a template class Array that can store an array of any data type. Include member functions to perform operations like adding an element, removing an element, and displaying the array. Demonstrate the functionality with different data types.

```
#include <iostream>
```

```
#include <stdexcept>
```

```
template <typename T>
```

```
class Array {
```

```
private:
```

```
    T *elements;
```

```
    int size;
```

```
    int capacity;
```

```
public:
```

```
    Array(int initialCapacity = 10) {
```

```
        capacity = initialCapacity;
```

```
        size = 0;
```

```
        elements = new T[capacity];
```

```

    }

    ~Array() {
        delete[] elements;
    }

    void addElement(const T& element) {
        if (size >= capacity) {
            int newCapacity = capacity * 2; // Resize the
array if it's full

            T *newElements = new T[newCapacity];

            for (int i = 0; i < size; ++i) {
                newElements[i] = elements[i];
            }

            delete[] elements;

            elements = newElements;

            capacity = newCapacity;
        }

        elements[size++] = element;
    }

    void removeElement(int index) {
        if (index < 0 || index >= size) {
            throw std::out_of_range("Index out of range");
        }

        for (int i = index; i < size - 1; ++i) {
            elements[i] = elements[i + 1];
        }

        size--;
    }

    void display() const {

```

```

        std::cout << "[";
        for (int i = 0; i < size; ++i) {
            std::cout << elements[i];

            if (i < size - 1) {
                std::cout << ", ";
            }
        }

        std::cout << "]" << std::endl;
    }
};

```

```

int main() {
    Array<int> intArray;                                // Example usage with different data types
    intArray.addElement(90);                             // Integer array
    intArray.addElement(20);
    intArray.addElement(50);
    intArray.display();
    intArray.removeElement(2);
    intArray.display();

    Array<double> doubleArray;                           // Double array
    doubleArray.addElement(3.14);
    doubleArray.addElement(2.68);
    doubleArray.display();
    doubleArray.removeElement(1);
    doubleArray.display();

    Array<std::string> stringArray;                      // String array
    stringArray.addElement("Hello");
    stringArray.addElement("World");
    stringArray.display();
}

```



```

return 0;

}

```

OUTPUT :-

The screenshot shows a web browser with multiple tabs of 'Online C Compiler'. The active tab displays a C++ program in a dark-themed editor. The program defines a template class 'Array' with private attributes 'elements', 'size', and 'capacity'. It includes methods for initialization, deletion, and adding elements, with a logic to resize the array if it's full. The output window shows the results of running the program with various inputs: '[90, 20, 50]', '[90, 20]', '[3.14, 2.68]', '[3.14]', and '[Hello, World]'. The program finished with exit code 0.

```

main.cpp
1 #include <iostream>
2 #include <stdexcept>
3 template <typename T>
4 class Array {
5 private:
6     T elements;
7     int size;
8     int capacity;
9 public:
10     Array(int initialCapacity = 10) {
11         capacity = initialCapacity;
12         size = 0;
13         elements = new T[capacity];
14     }
15     ~Array() {
16         delete[] elements;
17     }
18     void addElement(const T& element) {
19         if (size >= capacity) {
20             int newCapacity = capacity * 2;
21             T* newElements = new T[newCapacity];
22             for (int i = 0; i < size; ++i) {
23                 newElements[i] = elements[i];
24             }
25             delete[] elements;
26             elements = newElements;
27             capacity = newCapacity;
28         }
29         elements[size++] = element;
30     }
31 };
32
33 int main() {
34     Array<int> arr(10);
35     arr.addElement(90);
36     arr.addElement(20);
37     arr.addElement(50);
38     Array<int> arr2(5);
39     arr2.addElement(90);
40     arr2.addElement(20);
41     Array<double> arr3(5);
42     arr3.addElement(3.14);
43     arr3.addElement(2.68);
44     Array<double> arr4(5);
45     arr4.addElement(3.14);
46     Array<string> arr5(5);
47     arr5.addElement("Hello, World");
48     return 0;
49 }

```

input

```

[90, 20, 50]
[90, 20]
[3.14, 2.68]
[3.14]
[Hello, World]

```

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

## Polymorphism with Abstract Classes:

Create an abstract class **Animal** with a pure virtual function **sound()**. Derive classes **Dog**, **Cat**, and **Cow** from **Animal** and override the **sound()** function in each derived class. Write a program to demonstrate polymorphism using these classes.

```
#include <iostream>
```

```
#include <string>
```

```
using namespace std;
```

```
class Animal {
```

```
// Abstract class Animal
```

```
public:
```

```
    virtual void sound() const = 0;
```

```
// Pure virtual function sound
```

```
};
```

```
class Dog : public Animal {
```

```
// Derived class Dog from Animal
```

```
public:
```

```
    void sound() const override {
```

```

        cout << "Dog: Woof!" << endl;
    }
};

class Cat : public Animal {                // Derived class Cat from Animal
public:
    void sound() const override {
        cout << "Cat: Meow!" << endl;
    }
};

class Cow : public Animal {                // Derived class Cow from Animal
public:
    void sound() const override {
        cout << "Cow: Moo!" << endl;
    }
};

int main() {
    Animal* animals[3];                    // Declare an array of Animal pointers
    Dog dog;                               // Instantiate objects of Dog, Cat, and
    Cow cat;                               //
    Cat cat;
    Cow cow;

    animals[0] = &dog;                     // Assign addresses of objects to Animal
    pointers
    animals[1] = &cat;
    animals[2] = &cow;

    for (int i = 0; i < 3; ++i) {          // Use polymorphism to call the sound()
    function
        animals[i]->sound();
    }
}

```

```

return 0;

}

```

OUTPUT :-

```

1 #include <iostream>
2 #include <string>
3 using namespace std;
4 class Animal {           // Abstract class Animal
5 public:
6     virtual void sound() const = 0;           // Pure virtual function sound
7 };
8 class Dog : public Animal {           // Derived class Dog from Animal
9 public:
10     void sound() const override {
11         cout << "Dog: Woof!" << endl;
12     }
13 };
14 class Cat : public Animal {           // Derived class Cat from Animal
15 public:
16     void sound() const override {
17         cout << "Cat: Meow!" << endl;
18     }
19 };
20 class Cow : public Animal {           // Derived class Cow from Animal
21 public:
22     void sound() const override {
23         cout << "Cow: Mool!" << endl;
24     }
25 };
26 int main() {
27     Animal* animals[];           // Declare an array of Animal pointers
28     Dog dog;           // Instantiate objects of Dog, Cat, and Cow
29     Cat cat;
30
31     dog.sound();
32     cat.sound();
33     Cow cow;
34     cow.sound();
35
36     ...Program finished with exit code 0
37     Press ENTER to exit console.

```

## Static Member Functions:

Implement a class Math that has static member functions for basic mathematical operations like addition, subtraction, multiplication, and division. Demonstrate the use of these functions without creating an object of the class.

```
#include <iostream>
```

```
class Math {
```

```
public:
```

```

    static int add(int a, int b) {           // Static member functions
for basic math operations

```

```
    return a + b;
```

```
}
```

```
    static int subtract(int a, int b) {
```

```
    return a - b;
```

```

    }

    static int multiply(int a, int b) {

        return a * b;

    }

    static double divide(double a, double b) {

        if (b == 0) {

            std::cerr << "Error: Division by zero!" << std::endl;

            return 0;

        }

        return a / b;

    }

};

int main() {

    std::cout << "Addition: " << Math::add(20,5) << std::endl;
    // Using static member functions without creating an object of the class

    std::cout << "Subtraction: " << Math::subtract(20, 5) << std::endl;

    std::cout << "Multiplication: " << Math::multiply(20, 5) << std::endl;

    std::cout << "Division: " << Math::divide(20.0, 5.0) << std::endl;

    return 0;

}

```

OUTPUT :-

```
1 #include <iostream>
2 class Math {
3 public:
4     static int add(int a, int b) { // Static member functions for basic math operations
5         return a + b;
6     }
7     static int subtract(int a, int b) {
8         return a - b;
9     }
10    static int multiply(int a, int b) {
11        return a * b;
12    }
13    static double divide(double a, double b) {
14        if (b == 0) {
15            std::cerr << "Error: Division by zero!" << std::endl;
16            return 0;
17        }
18        return a / b;
19    }
20 };
21 int main() {
22     std::cout << "Addition: " << Math::add(20,5) << std::endl; // Using static member functions without creating an object of the class
23     std::cout << "Subtraction: " << Math::subtract(20, 5) << std::endl;
24     std::cout << "Multiplication: " << Math::multiply(20, 5) << std::endl;
25     std::cout << "Division: " << Math::divide(20.0, 5.0) << std::endl;
26     return 0;
27 }
28
```

Output:

```
Addition: 25
Subtraction: 15
Multiplication: 100
Division: 4
...Program finished with exit code 0
Press ENTER to exit console.
```

## Class Templates with Multiple Parameters:

Write a class template `Pair` that can store a pair of values of any two data types. Include member functions to set and get the values. Demonstrate the usage of this template with different data types.

```
#include <iostream>
```

```
template <typename T1, typename T2>
```

```
class Pair {
```

```
private:
```

```
    T1 first;
```

```
    T2 second;
```

```
public:
```

```
    Pair(const T1& f, const T2& s) : first(f), second(s) {}
```

the pair

// Constructor to initialize

```
    void setPair(const T1& f, const T2& s) {
```

pair

// Function to set values of the

```
        first = f;
```

```
        second = s;
```

```
    }
```

```

        T1 getFirst() const {                                // Function to get the first
value of the pair

            return first;

        }

        T2 getSecond() const {                             // Function to get the second
value of the pair

            return second;

        }
};

int main() {

    Pair<int, double> pair1(8, 3.14);                        // Example usage of Pair class
template

    Pair<std::string, bool> pair2("Hello", true);

    std::cout << "Pair 1: " << pair1.getFirst() << ", " << pair1.getSecond() << std::endl;

    pair1.setPair(10, 2.71);

    std::cout << "Pair 1 (after setting): " << pair1.getFirst() << ", " << pair1.getSecond() << std::endl;

    std::cout << "Pair 2: " << pair2.getFirst() << ", " << pair2.getSecond() << std::endl;

    return 0;

}

```

OUTPUT :-

```
1 #include <iostream>
2 template <typename T1, typename T2>
3 class Pair {
4 private:
5     T1 first;
6     T2 second;
7 public:
8     Pair(const T1& f, const T2& s) : first(f), second(s) {} // Constructor to initialize the pair
9     void setPair(const T1& f, const T2& s) { // Function to set values of the pair
10         first = f;
11         second = s;
12     }
13     T1 getFirst() const { // Function to get the first value of the pair
14         return first;
15     }
16     T2 getSecond() const { // Function to get the second value of the pair
17         return second;
18     }
19 };
20 int main() {
21     Pair<int, double> pair1(8, 3.14); // Example usage of Pair class template
22     Pair<std::string, bool> pair2("Hello", true);
23     std::cout << "Pair 1: " << pair1.getFirst() << ", " << pair1.getSecond() << std::endl;
24     pair1.setPair(10, 2.71);
25     std::cout << "Pair 1 (after setting): " << pair1.getFirst() << ", " << pair1.getSecond() << std::endl;
26     std::cout << "Pair 2: " << pair2.getFirst() << ", " << pair2.getSecond() << std::endl;
27     return 0;
28 }
```

Pair 1: 8, 3.14  
Pair 1 (after setting): 10, 2.71  
Pair 2: Hello, 1

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

## Friend Classes:

Create two classes Alpha and Beta. Make Beta a friend class of Alpha so that it can access private data members of Alpha. Implement functions in Beta to manipulate the private data of Alpha.

```
#include <iostream>
```

```
using namespace std;
```

```
class Beta;
```

```
class Alpha { // Alpha class
```

```
private:
```

```
    int privateData;
```

```
public:
```

```
    Alpha(int data) : privateData(data) {}
```

```
    friend class Beta; // Friend declaration for Beta class
```

```
    void displayPrivateData() { // Function to display privateData
```

```
        cout << "Alpha's privateData: " << privateData << endl;
```

```
    }
```

```

};

class Beta {                                // Beta class

public:

    void modifyAlphaData(Alpha& alpha, int newData) {                // Function to modify
privateData of Alpha

        alpha.privateData = newData;                                // Beta can access
privateData directly

    }

};

int main() {

    Alpha alphaObj(68);

    alphaObj.displayPrivateData();                                // Output: Alpha's privateData: 68

    Beta betaObj;

    betaObj.modifyAlphaData(alphaObj, 99);

    alphaObj.displayPrivateData();                                // Output: Alpha's privateData: 99

    return 0;

}

```

OUTPUT :-

The screenshot shows a web browser window with the URL [https://www.onlinegdb.com/online\\_c\\_compiler#](https://www.onlinegdb.com/online_c_compiler#). The code editor contains the C++ code from the previous block. The output window at the bottom shows the following text:

```

Alpha's privateData: 68
Alpha's privateData: 99

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

```

The Windows taskbar at the bottom indicates a temperature of 33°C, mostly sunny weather, and the date 04-07-2024.



