

Create a program that simulates a zoo with various animals. Each animal should have a common method called "speak" that makes a sound specific to the animal type.

**Objective:**

Utilize runtime polymorphism to achieve the following:

Define an abstract base class named Animal with a method speak that doesn't have an implementation (declare it abstract).

Create subclasses for different animals like Lion, Elephant, etc., inheriting from Animal.

Override the speak method in each subclass to define the specific sound of the animal (e.g., Lion roars, Elephant trumpets).

In the main program, create an array of Animal references. Populate this array with objects of different animal subclasses.

Loop through the animal array and call the speak method on each reference. Since the references are of the base class type, runtime polymorphism will determine the actual subclass and invoke the appropriate overridden speak method.

This exercise will demonstrate runtime polymorphism by:

Highlighting the separation between declared type (reference variable type) and actual type (object type).

Showing how the method call is resolved at runtime based on the actual object.

```
#include <iostream>
```

```
#include <vector>
```

```
class Animal {                                     // Abstract base class
```

```
public:
```

```
    virtual void speak() const = 0;                // Pure virtual function
```

```
    virtual ~Animal() {}                            // Virtual destructor to ensure proper  
cleanup of derived classes
```

```
};
```

```
class Lion : public Animal {                        // Lion class
```

```
public:
```

```
    void speak() const override {
```

```

        std::cout << "Lion sounds Roar!" << std::endl;
    }
};

class Elephant : public Animal {                                // Elephant class
public:
    void speak() const override {
        std::cout << "Elephant sounds Trumpet!" << std::endl;
    }
};

class Snake : public Animal {                                    // Snake class
public:
    void speak() const override {
        std::cout << "Snake sounds Hiss!" << std::endl;
    }
};

int main() {                                                    // Main function
    std::vector<Animal*> animals = {                             // Create an array (vector) of
Animal pointers
        new Lion(),
        new Elephant(),
        new Snake()
    };

    for (const auto& animal : animals) {                        // Iterate through the array and
make each animal speak
        animal->speak();
    }
}

```

```

        for (const auto& animal : animals) {
            allocated memory // Clean up dynamically

            delete animal;

        }

        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 page displays a C++ program in a dark-themed editor. The program defines an abstract base class `Animal` with a pure virtual function `speak()` and a virtual destructor. It then defines three derived classes: `Lion`, `Elephant`, and `Snake`, each overriding the `speak()` function. The `main()` function creates a vector of `Animal` pointers, adding instances of `Lion` and `Elephant`.

```

1 #include <iostream>
2 #include <vector>
3 class Animal {
4 public:
5     virtual void speak() const = 0; // Pure virtual function
6     virtual ~Animal() {} // Virtual destructor to ensure proper cleanup of derived classes
7 };
8 class Lion : public Animal { // Lion class
9 public:
10     void speak() const override {
11         std::cout << "Lion sounds Roar!" << std::endl;
12     }
13 };
14 class Elephant : public Animal { // Elephant class
15 public:
16     void speak() const override {
17         std::cout << "Elephant sounds Trumpet!" << std::endl;
18     }
19 };
20 class Snake : public Animal { // Snake class
21 public:
22     void speak() const override {
23         std::cout << "Snake sounds Hiss!" << std::endl;
24     }
25 };
26 int main() {
27     std::vector<Animal*> animals = { // Main function
28         new Lion(), // create an array (vector) of Animal pointers
29         new Elephant()
30     };
31 }

```

The interface includes a sidebar with navigation links like 'My Projects', 'Classroom', and 'Learn Programming'. At the bottom, there's a command line for input and a banner for 'Your Alternative SAS Language Environment'.

**Virtual Function :-**

```

#include <iostream>

using namespace std;

class Base
{
    public:
    virtual void show() = 0;
};

class Derived : public Base

```

```

{
    public:
    void show()
    {
        std::cout << "Derived class is derived from the class." << std::endl;
    }
};

int main()
{
    Base *bptr;

    Derived d;

    bptr = &d;

    bptr -> show();

    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 browser has several tabs open, including 'Meeting Link C++ Programming', 'Meeting now | Microsoft To...', and 'Online C Compiler - online editor'. The compiler interface shows a file named 'main.cpp' with the following code:

```

1 #include <iostream>
2 using namespace std;
3 class Base
4 {
5     public:
6     virtual void show() = 0;
7 };
8 class Derived : public Base
9 {
10     public:
11     void show()
12     {
13         std::cout << "Derived class is derived from the class." << std::endl;
14     }
15 };
16 int main()
17 {
18     Base *bptr;
19     Derived d;
20     bptr = &d;
21     bptr -> show();
22     return 0;
23 }

```

The output window at the bottom shows the result of the program execution:

```

Derived class is derived from the class.
...Program finished with exit code 0
Press ENTER to exit console.

```

The Windows taskbar at the bottom indicates the system time is 11:59 on 01-07-2024, and the weather is 27°C Cloudy.

## **DESTRUCTORS :-**

```
#include <iostream>
```

```
#include <cstring>
```

```
class String {
```

```
private:
```

```
    char* s;
```

```
    int size;
```

```
public:
```

```
    String(char*);
```

```
    ~String();
```

```
};
```

```
String::String(char* c)
```

```
{
```

```
    size = strlen(c);
```

```
    s = new char[size + 1];
```

```
    strcpy(s, c);
```

```
}
```

```
String::~~String() {delete[] s;}
```

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 browser has several tabs open, including 'Meeting Link\_C++ Programming', 'Meeting now | Microsoft To...', and 'Online C Compiler - online editor'. The main content area displays a C++ program in a dark-themed editor. The program defines a 'String' class with a private character array 's', an integer 'size', and public methods for construction, destruction, and assignment. The code is as follows:

```
1 #include <iostream>
2 #include <string>
3 class String {
4 private:
5     char* s;
6     int size;
7 public:
8     String(char*);
9     ~String();
10 };
11 String::String(char* c)
12 {
13     size = strlen(c);
14     s = new char[size + 1];
15     strcpy(s, c);
16 }
17 String::~String() {delete[] s;}
```

Below the code editor, there is a 'Compilation failed due to following error(s)' message. The error details are:

```
/usr/bin/ld: /usr/lib/gcc/x86_64-linux-gnu/11/../../../../x86_64-linux-gnu/Scrt1.o: in function `_start':
(.text+0x1b): undefined reference to `main'
collect2: error: ld returned 1 exit status
```

The bottom of the image shows a Windows taskbar with the search bar and various application icons.

## VIRTUAL DESTRUCTOR :-

```
#include<iostream>
```

```
using namespace std;
```

```
class base{
    public:
    base()
    {
        cout<<"constructing base \n";
    }
    ~base()
    {
        cout<<"destructing base \n";
    }
};
```

```
class derived: public base{
```

```
    public:
```

```

    derived(){
        cout<<"constructing derived \n";
    }

    ~derived(){
        cout<<"destructing derived \n";
    }
};

int main(void)
{
    derived*d= new derived();

    base*b = d;

    delete b;

    getchar();

    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 at the bottom shows the following text:

```

constructing base
constructing derived
destructing base

```

The browser's taskbar at the bottom shows the system clock as 12:44 on 01-07-2024, with a temperature of 29°C and a cloudy weather forecast.

## **VIRTUAL CONSTRUCTOR :-**

```
#include<iostream>

using namespace std;

class base{
    public:
    base()
    {
        cout<<"constructing base \n";
    }
    virtual~base()
    {
        cout<<"destructing base \n";
    }
};

class derived: public base{
    public:
    derived(){
        cout<<"constructing derived \n";
    }
    ~derived(){
        cout<<"destructing derived \n";
    }
};

int main(void)
{
    derived*d= new derived();

    base*b = d;

    delete b;
```



```

    getchar();

    return 0;

}

```

OUTPUT :-

```

main.cpp
1 using namespace std;
2 class base{
3 public:
4     base()
5     {
6         cout<<"constructing base \n";
7     }
8     virtual ~base()
9     {
10        cout<<"destructing base \n";
11    }
12 };
13 class derived: public base{
14 public:
15     derived(){
16         cout<<"constructing derived \n";
17     }
18     ~derived(){
19         cout<<"destructing derived \n";
20     }
21 };
22 int main(void)
23 {
24     derived* d = new derived();
25     base* b = d;
26     delete b;
27     getchar();
28     return 0;
29 }

```

Input

```

constructing base
constructing derived
destructing derived
destructing base

```

**FRIEND CLASS AND FUNCTION :-**

```
#include <iostream>
```

```
class A {
```

```
    private:
```

```
    int a;
```

```
    public:
```

```
    A() { a = 0;
```

```
    }
```

```
    friend class B;
```

```
};
```

```
class B {
```

```
    private:
```

```

        int b;

        public:

        void showA(A& x)

        {

            std::cout << "A::a=" << x.a;

        }

};

int main() {

    A a;

    B b;

    b.showA(a);

    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 following C++ code:

```

1 #include <iostream>
2 class A {
3     private:
4         int a;
5     public:
6         A() { a = 0; }
7     }
8     friend class B;
9 };
10 class B {
11     private:
12         int b;
13     public:
14         void showA(A& x)
15         {
16             std::cout << "A::a=" << x.a;
17         }
18 };
19 int main() {
20     A a;
21     B b;
22     b.showA(a);
23     return 0;
24 }

```

The output window shows the following text:

```

A::a=0
...Program finished with exit code 0
Press ENTER to exit console.

```

**FRIEND CLASS AND FUNCTION :-**

```
#include<iostream>
```

```
class B;

class A{
    public:
        void showB(B&);
};

class B{
    private:
        int b;
    public:
        B(){ b=0;}
        friend void A::showB(B& x);
};

void A::showB(B& x)
{
    std::cout<<"B::b="<<x.b;
}

int main(){
    A a;
    B x;
    a.showB(x);
    return 0;
}
```

OUTPUT :-

```
1 #include<iostream>
2 class B;
3 class A{
4     public:
5     void showB(B& x);
6 };
7 class B{
8     private:
9     int b;
10    public:
11    B(){ b=0;}
12    friend void A::showB(B& x);
13 };
14 void A::showB(B& x)
15 {
16     std::cout<<"B::b="<<x.b;
17 }
18 int main(){
19     A a;
20     B x;
21     a.showB(x);
22     return 0;
23 }
```

B::b=0

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

## DEMONSTRATE FRIEND CLASS :-

```
#include<iostream>
```

```
class A{
```

```
    private:
```

```
    int a;
```

```
    public:
```

```
    A(){ a = 0;}
```

```
    friend class B;
```

```
};
```

```
class B{
```

```
    private:
```

```
    int b;
```

```
    public:
```

```
    void showA(A& x){
```

```
        std::cout<<"A::a="<<x.a;
```

```

}

};

int main(){

    A a;

    B b;

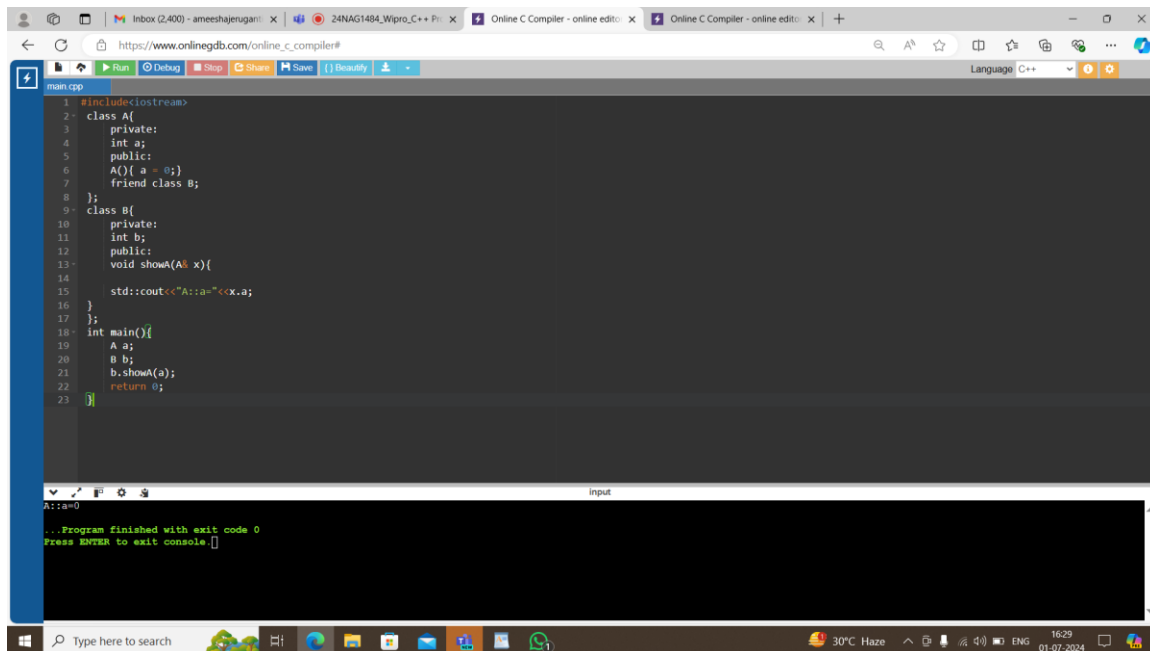
    b.showA(a);

    return 0;

}

```

OUTPUT :-



The screenshot shows a web browser window with an online C++ compiler. The code is as follows:

```

1 #include<iostream>
2 class A{
3     private:
4         int a;
5     public:
6         A(){ a = 0; }
7         friend class B;
8 };
9 class B{
10     private:
11         int b;
12     public:
13         void showA(A& x){
14             std::cout<<"A::a="<<x.a;
15         }
16 };
17 int main(){
18     A a;
19     B b;
20     b.showA(a);
21     return 0;
22 }

```

The output of the program is:

```

A::a=0
...Program finished with exit code 0
Press ENTER to exit console.

```

You have a TemperatureSensor class that measures temperature in Celsius. You want a separate DisplayTemperature function to print the temperature in Fahrenheit. However, the conversion formula requires accessing the private celsius member.

Create a TemperatureSensor class with a private celsius member and a public constructor.

Implement a friend function DisplayTemperature that takes a TemperatureSensor object and prints the temperature in Fahrenheit (conversion formula provided).

Write a main function to demonstrate how to use the classes.

```
#include <iostream>
```

```

class TempSensor {
private:
    double c;                // Private member to store temperature in Celsius

public:
    TempSensor(int tempC) : c(tempC) {}                // Constructor to initialize temperature
in Celsius

    friend void DisplayTemp(const TempSensor& sensor);    // Friend function
declaration (friend function can access private members)

};

void DisplayTemp(const TempSensor& sensor) {            // Friend function
definition to display temperature in Fahrenheit

    double fahrenheit = sensor.c * 9.0 / 5.0 + 32.0;    // Convert Celsius to Fahrenheit

    std::cout << "Temperature in Fahrenheit: " << fahrenheit << " F" << std::endl;    //
Print the temperature in Fahrenheit

}

int main() {

    TempSensor sensor(65.0);                // Create a TemperatureSensor object with a
temperature                                of 65 degrees Celsius

    DisplayTemp(sensor);                    // Display the temperature in Fahrenheit using the
friend function

    return 0;

}

```

OUTPUT :-

```
1 #include <iostream>
2 class TempSensor {
3 private:
4     double c; // Private member to store temperature in Celsius
5 public:
6     TempSensor(int tempC) : c(tempC) {} // Constructor to initialize temperature in Celsius
7     friend void DisplayTemp(const TempSensor& sensor); // Friend function declaration (friend function can access private members)
8 };
9 void DisplayTemp(const TempSensor& sensor) { // Friend function definition to display temperature in Fahrenheit
10     double fahrenheit = sensor.c * 9.0 / 5.0 + 32.0; // Convert Celsius to Fahrenheit
11     std::cout << "Temperature in Fahrenheit: " << fahrenheit << " F" << std::endl; // Print the temperature in Fahrenheit
12 }
13 int main() {
14     TempSensor sensor(65.0); // Create a TempSensor object with a temperature of 65 degrees Celsius
15     DisplayTemp(sensor); // Display the temperature in Fahrenheit using the friend function
16     return 0;
17 }
18
```

Temperature in Fahrenheit: 149 F

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

### Friend Class for Stream Insertion:

**Scenario:** You have a Point class with private members for x and y coordinates. You want to define a way to easily print Point objects to output streams like cout.

Create a Point class with private x and y members and a public constructor.

Design a friend class PointOutputStream that has an overloaded << operator to format and insert Point objects into output streams.

In main, demonstrate creating Point objects and printing them using cout.

```
#include <iostream>
```

```
class Point; //declaration of point class
```

```
class PointOutputStream { // Friend class declaration
```

```
public:
```

```
    friend std::ostream& operator<<(std::ostream& os, const Point& point);
```

```
};
```

```
class Point {
```

```
// Point class declaration
```

```
private:
```

```
    int x;
```

```

        int y;

public:
        Point(int x, int y) : x(x), y(y) {}

        friend std::ostream& operator<<(std::ostream& os, const Point& point);           //
        Declaring PointOutputStream as friend to access private members

};

std::ostream& operator<<(std::ostream& os, const Point& point) {                       //
    Overloaded << operator for Point class (defined outside the class)

        os << "(" << point.x << ", " << point.y << ")";

        return os;

}

int main() {

    Point p1(6, 8);

    Point p2(-1, 5);

    std::cout << "Point p1: " << p1 << std::endl;

    std::cout << "Point p2: " << p2 << std::endl;

    return 0;

}

```

OUTPUT :-



```
1 #include <iostream>
2 class Point; //declaration of point class
3 class PointOutputStream { // Friend class declaration
4 public:
5     friend std::ostream& operator<<(std::ostream& os, const Point& point);
6 };
7 class Point { // Point class declaration
8 private:
9     int x;
10    int y;
11 public:
12    Point(int x, int y) : x(x), y(y) {}
13    friend std::ostream& operator<<(std::ostream& os, const Point& point); // Declaring PointOutputStream as friend to access private members
14 };
15 std::ostream& operator<<(std::ostream& os, const Point& point) { // Overloaded << operator for Point class (defined outside the class)
16     os << "(" << point.x << ", " << point.y << ")";
17     return os;
18 }
19 int main() {
20     Point p1(6, 8);
21     Point p2(-1, 5);
22     std::cout << "Point p1: " << p1 << std::endl;
23     std::cout << "Point p2: " << p2 << std::endl;
24     return 0;
25 }
26
```

Point p1: (6, 8)  
Point p2: (-1, 5)

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