**DAY-6:**

**1.**

#include <iostream>

using namespace std;

class Animal {

public:

virtual void speak() const = 0;

virtual ~Animal() {}

};

class Lion : public Animal {

public:

void speak() const override {

cout << "Lion roars" << endl;

}

};

class Elephant : public Animal {

public:

void speak() const override {

cout << "Elephant trumpets" << endl;

}

};

class Dog : public Animal {

public:

void speak() const override {

cout << "Dog barks" << endl;

}

};

class Cat : public Animal {

public:

void speak() const override {

cout << "Cat meows" << endl;

}

};

int main() {

//int zoosize=4;

Animal\* zoo[4];

zoo[0]=new Lion();

zoo[1]=new Elephant();

zoo[2]=new Dog();

zoo[3]=new Cat();

for(int i=0;i<4;i++)

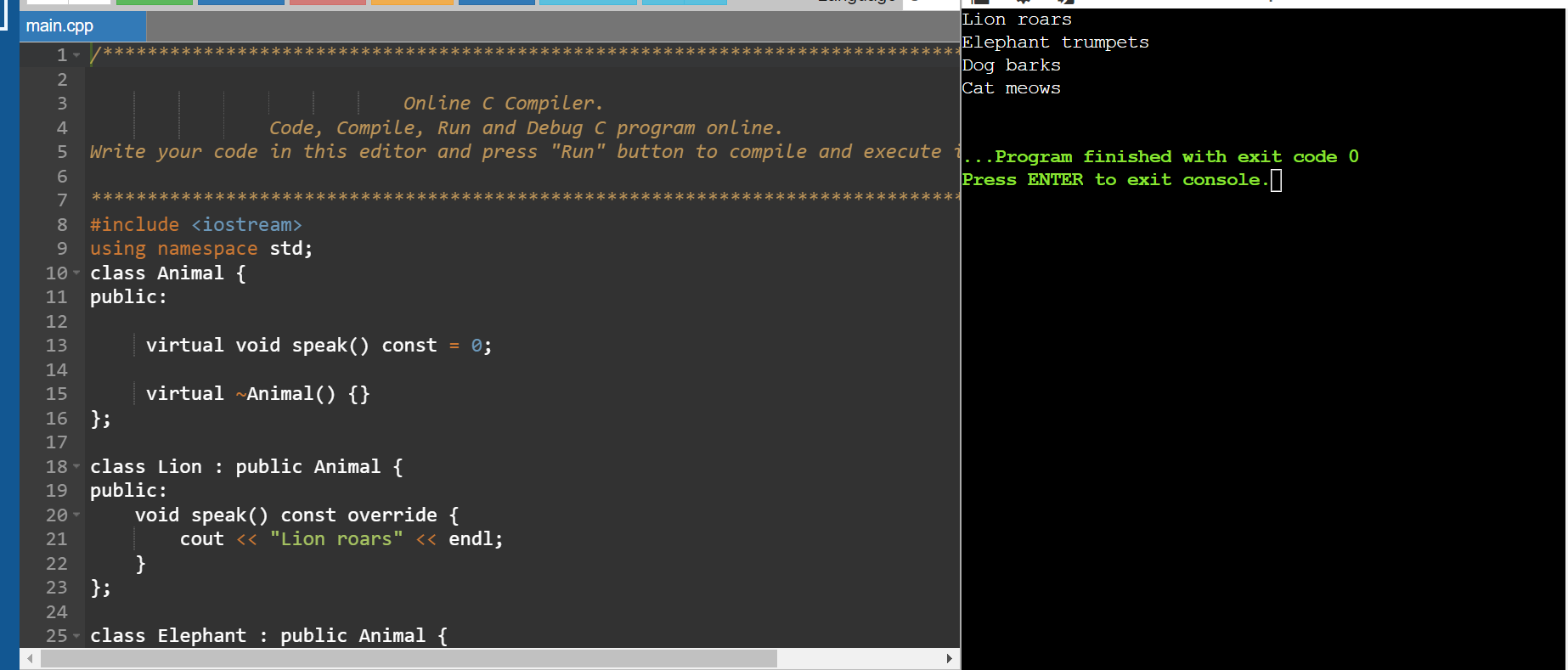
{

zoo[i]->speak();

}

return 0;

}



**2**.

#include <iostream>

using namespace std;

class base

{

public:

virtual void show()=0;

};

class derived:public base

{

public:

void show()

{

cout<<"derived class is derived from base class"<<endl;

}

};

int main()

{

base \*bptr;

derived d;

bptr = &d;

bptr->show();

return 0;

}



**3.**

#include<iostream>

#include<cstring>

using namespace std;

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;

}

void String::print()

{

cout<<"s"<<endl;

}

int main()

{

char text[]="hello";

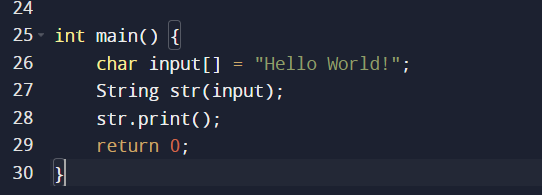
String str(text);

str.print();

return 0;

}





Output:Hello world

**4.**

#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<<"contructing derived \n";

}

~derived()

{

cout<<"Destructing derived \n";

}

};

int main(void)

{

derived \*d=new derived();

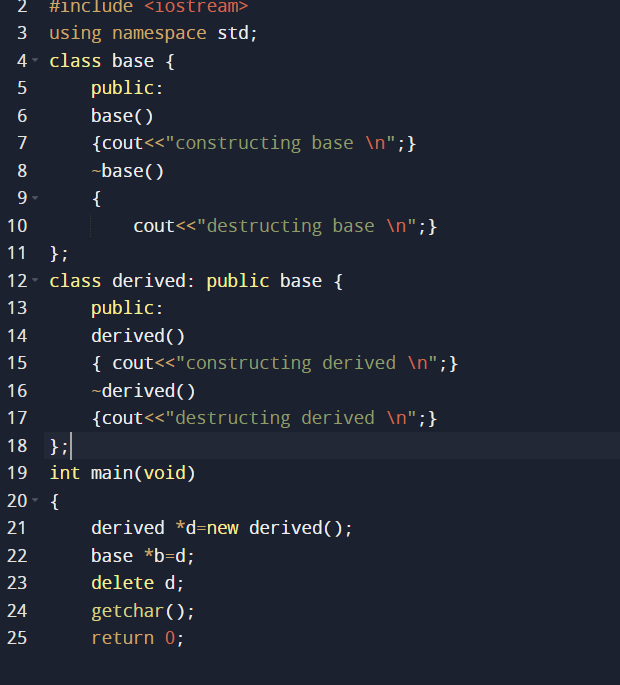
base \*b=d;

delete b;

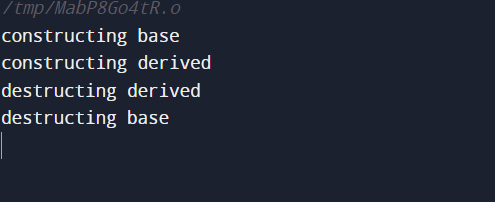
getchar();

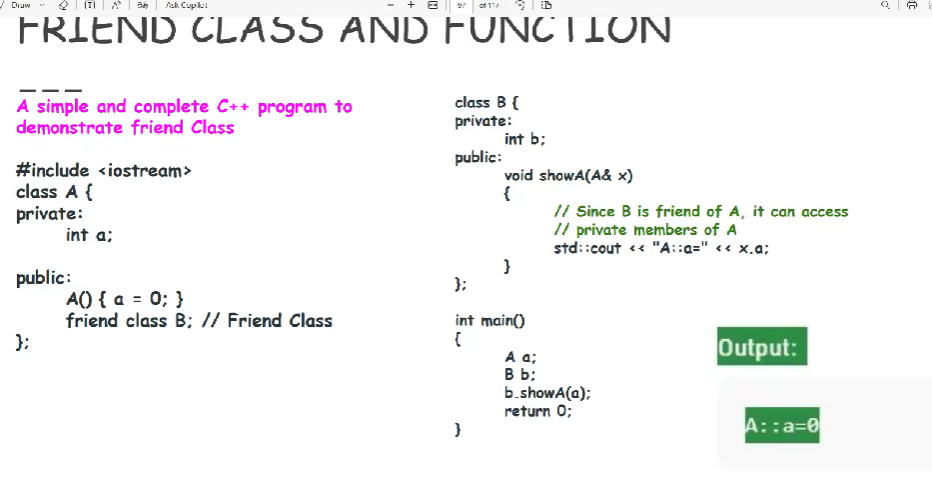
return 0;

}



Output:





**5.**

#include <iostream>

using namespace std;

class A

{

private:

int a;

public:

A()

{

a=0;

}

friend class B;

};

class B

{

private:

int b;

public:

void showA(A& x)

{

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

}

};

int main()

{

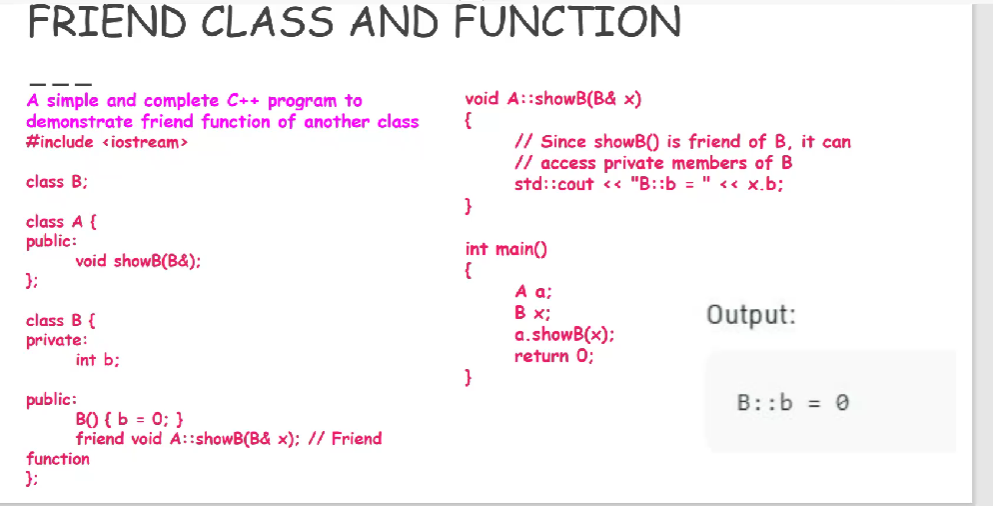
A a;

B b;

b.showA(a);

return 0;

}



**6.**

#include <iostream>

using namespace std;

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)

{

cout<<"B::b="<<x.b;

}

int main()

{

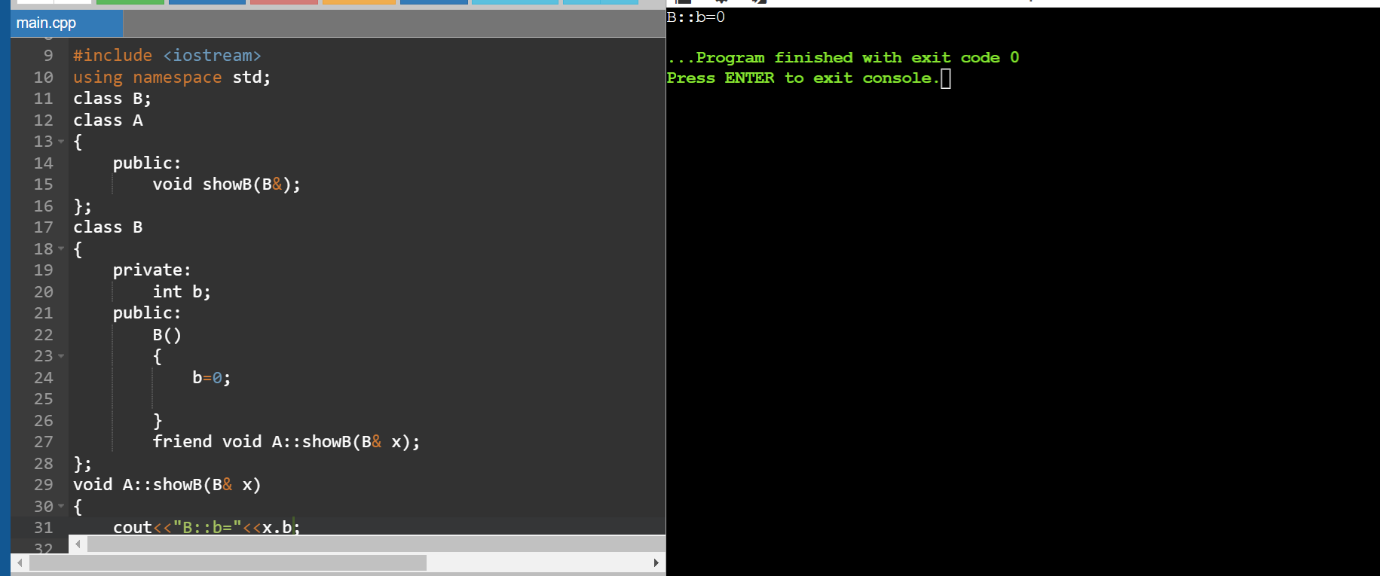
A a;

B x;

a.showB(x);

return 0;

}



7.#include <iostream>

using namespace std;

class Example {

private:

int value;

public:

// Default constructor

Example() : value(0) {

cout << "Default constructor value is " << value << endl;

}

// Parameterized constructor

Example(int v) : value(v) {

cout << "Parameterized constructor Value is " << value << endl;

}

// Copy constructor

Example(const Example &other) : value(other.value) {

cout << "Copy constructor Value is " << value << endl;

}

// Destructor

~Example() {

cout << "Destructor Value was " << value << endl;

}

void display() const {

cout << "Value: " << value << endl;

}

};

int main() {

Example e1; // Using default constructor

e1.display();

Example e2(70);// Using parameterized constructor

e2.display();

Example e3 = e2; // Using copy constructor

e3.display();

return 0;

}

8.#include <iostream>

using namespace std;

class A

{

private:

int a;

public:

A()

{

a=0;

}

friend class B;

};

class B

{

private:

int b;

public:

void showA(A& x)

{

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

}

};

int main()

{

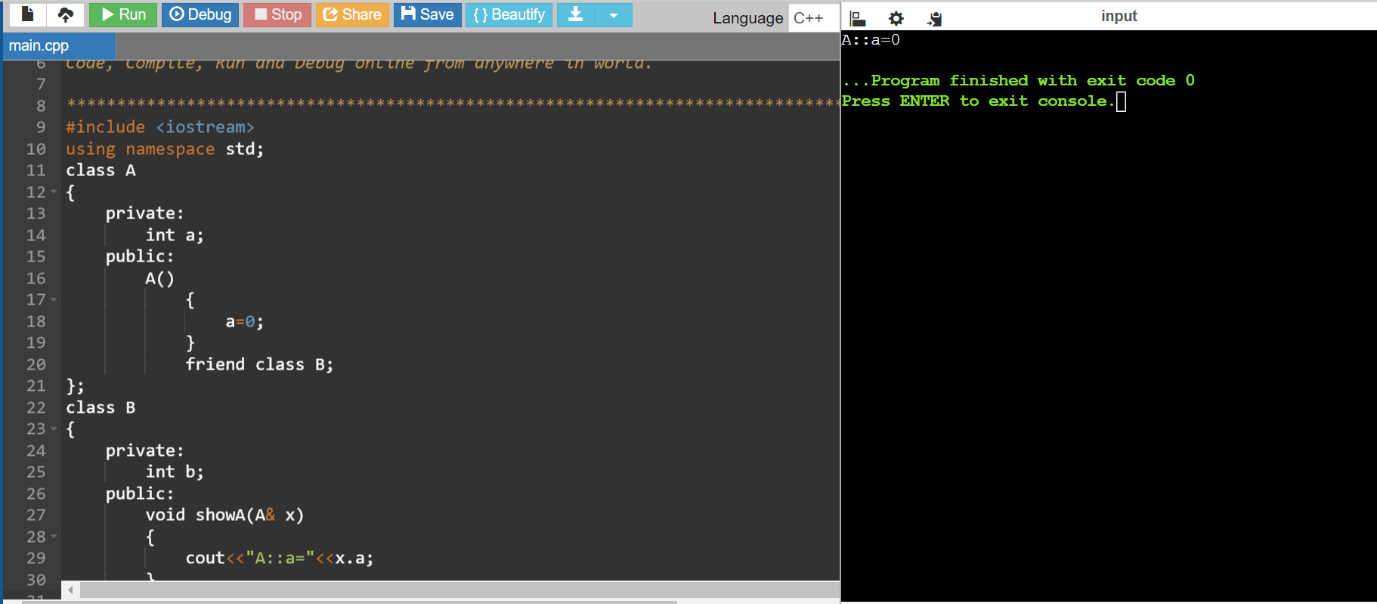
A a;

B b;

b.showA(a);

return 0;

}



9.

#include <iostream>

using namespace std;

class TempSensor {

private:

double celsius;

public:

TempSensor(double temp) : celsius(temp) {}

friend void Display(const TempSensor&);

};

void Display(const TempSensor& sensor)

{

double ft = (sensor.celsius \* 9.0 / 5.0) + 32.0;

cout << "Temperature in Fahrenheit: " << ft << endl;

}

int main() {

TempSensor s(45);

Display(s);

return 0;

}

