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>

using namespace std;

class Shape {

public:

virtual void draw() const {

cout << "Drawing a shape" << endl;

}

};

class Circle : public Shape {

public:

void draw() const override {

cout << "Drawing a circle" << endl;

}

};

class Rectangle : public Shape {

public:

void draw() const override {

cout << "Drawing a rectangle" << endl;

}

};

class Triangle : public Shape {

public:

void draw() const override {

cout << "Drawing a triangle" << endl;

}

};

void drawShape(const Shape\* shape) {

shape->draw();

}

int main() {

Shape\* circle = new Circle();

Shape\* rectangle = new Rectangle();

Shape\* triangle = new Triangle();

drawShape(circle);

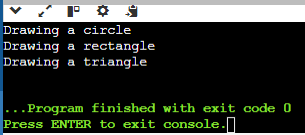
drawShape(rectangle);

drawShape(triangle);

return 0;

}

Output:



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>

using namespace std;

class Account {

private:

static int totalAccounts;

public:

Account() {

totalAccounts++;

}

~Account() {

totalAccounts--;

}

static void displayTotalAccounts() {

cout << "Total number of accounts: " << totalAccounts << endl;

}

};

int Account::totalAccounts = 0;

int main() {

Account account1;

Account account2;

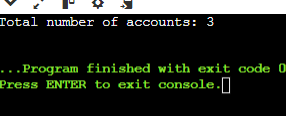
Account account3;

Account::displayTotalAccounts();

return 0;

}

Output:



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>

using namespace std;

class Box{

private:

float length;

float breath;

float height;

public:

Box() : length(0), breath(0), height(0) {}

Box(float length,float breath,float height){

this->length=length;

this->breath=breath;

this->height=height;

}

double getLength() const {

return length;

}

double getBreadth() const {

return breath;

}

double getHeight() const {

return height;

}

friend float volume(const Box& b);

};

float volume(const Box& b) {

return b.length \* b.breath \* b.height;

}

int main() {

Box box1(4.5, 5.0, 6.9);

Box box2(3.4, 5.6, 7.5);

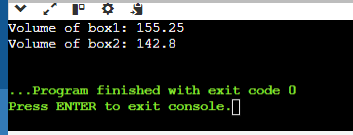
cout << "Volume of box1: " << volume(box1) << endl;

cout << "Volume of box2: " << volume(box2) << endl;

return 0;

}

Output:



Pointers:

Design a class Student with data members name and age. Create an array of Student objects dynamically using pointers. Implement functions to set and display the details of students. Also, write a function to deallocate the memory.

#include <iostream>

#include <string>

using namespace std;

class Student {

private:

string name;

int age;

public:

Student() : name(""), age(0) {}

Student(const string& name, int age) : name(name), age(age) {}

void setDetails(const string& name, int age) {

this->name = name;

this->age = age;

}

void display() const {

cout << "Name: " << name << ", Age: " << age << endl;

}

};

Student\* createStudents(int count) {

return new Student[count];

}

int main() {

int numStudents;

cout << "Enter the number of students: ";

cin >> numStudents;

Student\* students = createStudents(numStudents);

for (int i = 0; i < numStudents; ++i) {

string name;

int age;

cout << "Enter details for student " << i + 1 << endl;

cout << "Name: ";

cin >> name;

cout << "Age: ";

cin >> age;

students[i].setDetails(name, age);

}

cout << "Student details:" << endl;

for (int i = 0; i < numStudents; ++i) {

students[i].display();

}

return 0;

}

Output:



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>

using namespace std;

class Math {

public:

static int add(int a, int b) { 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(int a, int b) {

return b != 0 ? static\_cast<double>(a) / b : 0;

}

};

int main() {

cout << "Addition: " << Math::add(299, 187) << endl;

cout << "Subtraction: " << Math::subtract(56, 15) << endl;

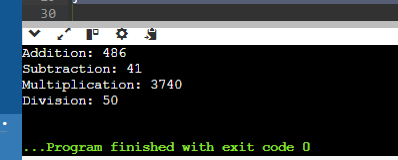
cout << "Multiplication: " << Math::multiply(110, 34) << endl;

cout << "Division: " << Math::divide(150,3) << endl;

return 0;

}

Output:



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>

using namespace std;

class Animal {

public:

virtual void sound() const = 0;

};

class Dog : public Animal {

public:

void sound() const override {

cout << "Woof!" << endl;

}

};

class Cat : public Animal {

public:

void sound() const override {

cout << "Meow!" << endl;

}

};

class Cow : public Animal {

public:

void sound() const override {

cout << "Humba!" << endl;

}

};

void makeSound(const Animal\* animal) {

animal->sound();

}

int main() {

Dog dog;

Cat cat;

Cow cow;

Animal\* animals[] = { &dog, &cat, &cow };

for (Animal\* animal : animals) {

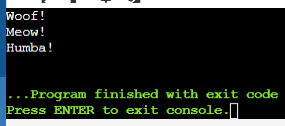
makeSound(animal);

}

return 0;

}

Output:



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 {

private:

int x;

friend class Beta;

public:

Alpha(int val = 0) : x(val) {}

void display() {

cout << "Alpha::x = " << x << endl;

}

};

class Beta {

public:

void manipulateAlpha(Alpha& alpha, int val) {

alpha.x = val;

}

void displayAlpha(const Alpha& alpha) {

cout << "Alpha::x = " << alpha.x << endl;

}

};

int main() {

Alpha alpha(40);

alpha.display();

Beta beta;

beta.manipulateAlpha(alpha, 30);

alpha.display();

beta.displayAlpha(alpha);

return 0;

}

Output:

