1. 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() {

Circle circle;

Rectangle rectangle;

Triangle triangle;

drawShape(&circle);

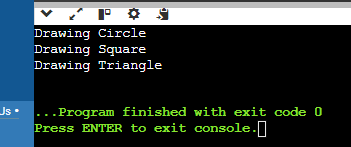
drawShape(&rectangle);

drawShape(&triangle);

return 0;

}

OUTPUT:



1. 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.

A: #include <iostream>

using namespace std;

class Account {

private:

static int totalAccounts; // Static data member to keep track of total accounts

public:

Account() {

totalAccounts++; }

static int getTotalAccounts() {

return totalAccounts; }

};

int Account::totalAccounts = 0;

void displayTotalAccounts() {

cout << "Total Accounts: " << Account::getTotalAccounts() << endl; }

int main() {

Account acc1;

Account acc2;

Account acc3;

displayTotalAccounts(); {

Account acc4;

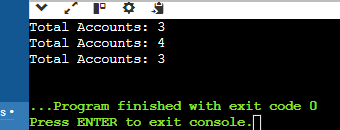
displayTotalAccounts(); }

displayTotalAccounts(); // Output: Total Accounts: 3

return 0;

}

OUTPUT:



1. 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.

A: #include <iostream>

using namespace std;

class Box;

int volume(const Box& box);

class Box {

private:

int length;

int breadth;

int height;

public:

Box(int l, int b, int h) : length(l), breadth(b), height(h) {}

friend int volume(const Box& box);

};

int volume(const Box& box) {

return box.length \* box.breadth \* box.height; }

int main() {

Box box1(3, 4, 5);

Box box2(2, 2, 2);

int vol1 = volume(box1);

int vol2 = volume(box2);

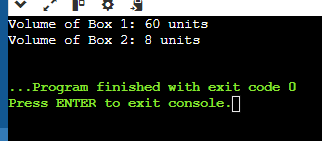
cout << "Volume of Box 1: " << vol1 << " units" << endl;

cout << "Volume of Box 2: " << vol2 << " units" << endl;

return 0;

}

OUTPUT:



1. 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.

A: #include <iostream>

#include <cassert>

using namespace std;

template <typename T>

class Array {

private:

T\* elements; // Pointer to dynamically allocated array

int capacity; // Capacity of the array (maximum elements it can hold)

int size; // Current number of elements in the array

public:

Array(int initialCapacity) : capacity(initialCapacity), size(0) {

elements = new T[capacity];

}

void add(const T& element) { // Function to add an element to the array

assert(size < capacity);

elements[size++] = element; }

void remove() {

if (size > 0) {

--size; }

}

void display() const {

cout << "Array elements:";

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

cout << " " << elements[i]; }

cout << endl; }

};

int main() {

Array<int> intArray(5);

intArray.add(10);

intArray.add(20);

intArray.add(30);

intArray.display();

Array<string> stringArray(3);

stringArray.add("Hello");

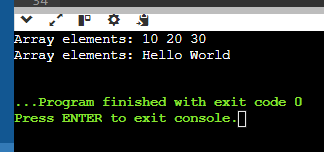
stringArray.add("World");

stringArray.display();

return 0;

}

OUTPUT:



1. 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.

A: #include <iostream>

#include <string>

using namespace std;

class Student {

private:

string name;

int age;

public:

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

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

void setDetails(const string& n, int a) {

name = n;

age = a; }

void display() const {

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

int main() {

int numStudents;

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

cin >> numStudents;

Student\* students = new Student[numStudents];

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

string name;

int age;

cout << "\nEnter details for student " << i + 1 << ":" << endl;

cout << "Name: ";

cin.ignore(); // Ignore newline left in the buffer

getline(cin, name);

cout << "Age: ";

cin >> age;

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

cout << "\nDisplaying details of students:" << endl;

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

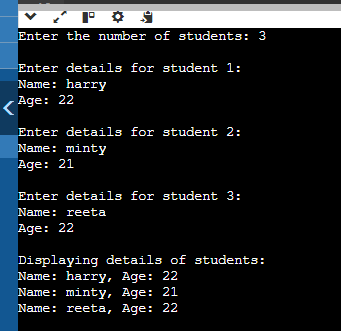
students[i].display(); }

delete[] students;

return 0;

}

OUTPUT:



1. 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.

A: #include <iostream>

using namespace std;

class Animal { // Abstract class Animal

public:

virtual void sound() const = 0; // Pure virtual function

virtual ~Animal() {} // Virtual destructor

};

class Dog : public Animal {

public:

void sound() const override {

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

};

class Cat : public Animal {

public:

void sound() const override {

cout << "Cat: Meow!" << endl; }

};

class Cow : public Animal {

public:

void sound() const override {

cout << "Cow: Moo!" << endl; }

};

int main() {

const int numAnimals = 3;

Animal\* animals[numAnimals];

animals[0] = new Dog();

animals[1] = new Cat();

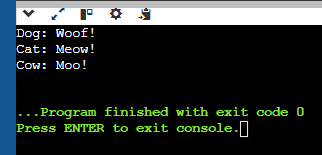
animals[2] = new Cow();

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

animals[i]->sound(); } // Polymorphic behavior

return 0;

}

OUTPUT:  


1. 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.

A: #include <iostream>

using namespace std;

class Math {

public:

static int addition(int a, int b) {

return a + b; }

static int subtraction(int a, int b) {

return a - b; }

static int multiplication(int a, int b) {

return a \* b; }

static double division(int a, int b) {

if (b == 0) {

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

return 0.0; // Error case }

return static\_cast<double>(a) / b; }

};

int main() {

int x = 9, y = 5; // Using static member functions without creating an object

cout << "Addition: " << Math::addition(x, y) << endl;

cout << "Subtraction: " << Math::subtraction(x, y) << endl;

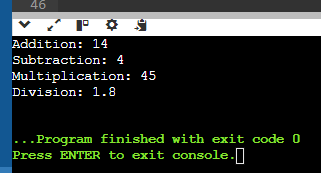
cout << "Multiplication: " << Math::multiplication(x, y) << endl;

cout << "Division: " << Math::division(x, y) << endl;

return 0;

}

OUTPUT:



1. 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.

A: #include <iostream>

using namespace std;

class Beta; // Forward declaration of Beta

class Alpha {

private:

int data;

public:

Alpha(int value) : data(value) {}

void display() const {

cout << "Alpha data: " << data << endl; }

friend class Beta; // Make Beta a friend class of Alpha

};

class Beta {

public:

void setData(Alpha& a, int value) {

a.data = value; }

void addData(Alpha& a, int value) {

a.data += value; }

};

int main() {

Alpha a(10);

a.display();

Beta b;

b.setData(a, 20);

cout << "After setting data to 20:" << endl;

a.display();

b.addData(a, 5);

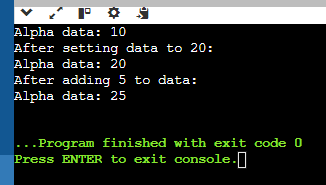
cout << "After adding 5 to data:" << endl;

a.display();

return 0;

}

OUTPUT:



1. 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.

A: #include <iostream>

#include <string>

using namespace std;

template <typename T1, typename T2>

class Pair {

private:

T1 first;

T2 second;

public:

void setValues(const T1& f, const T2& s) {

first = f;

second = s; }

T1 getFirst() const {

return first; }

T2 getSecond() const {

return second; }

void display() const {

cout << "First: " << first << ", Second: " << second << endl; }

};

int main() {

Pair<int, double> p1; // Pair of int and double

p1.setValues(42, 3.14);

cout << "Pair1: " << endl;

p1.display();

Pair<string, int> p2; // Pair of string and int

p2.setValues("Age", 25);

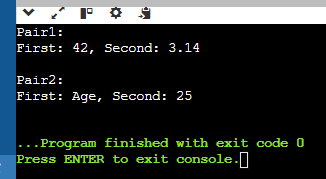
cout << "\nPair2:" << endl;

p2.display();

return 0;

}

OUTPUT:



1. Pointer to Objects:

Define a class Book with data members title and author. Create an array of pointers to Book objects. Write functions to input details for each book, display the details, and search for a book by title.

A: #include <iostream>

#include <string>

using namespace std;

class Book {

private:

string title;

string author;

public:

void setDetails(const string& t, const string& a) {

title = t;

author = a; }

string getTitle() const {

return title; }

void display() const {

cout << "Title: " << title << ", Author: " << author << endl; }

};

void inputDetails(Book\* books[], int n) { // Function to input details for each book

string title, author;

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

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

cout << "Title: ";

getline(cin, title);

cout << "Author: ";

getline(cin, author);

books[i] = new Book();

books[i]->setDetails(title, author);

}

} void displayDetails(Book\* books[], int n) { // Function to display details of all books

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

books[i]->display();

}

}

Book\* searchByTitle(Book\* books[], int n, const string& title) {

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

if (books[i]->getTitle() == title) {

return books[i];

} }

return nullptr;

}

int main() {

int n;

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

cin >> n;

cin.ignore();

Book\* books[n];

inputDetails(books, n);

cout << "\nDisplaying book details:\n";

displayDetails(books, n);

string title;

cout << "\nEnter the title of the book to search: ";

getline(cin, title);

Book\* foundBook = searchByTitle(books, n, title);

if (foundBook) {

cout << "\nBook found:\n";

foundBook->display();

} else {

cout << "\nBook not found.\n";

}

return 0; }

OUTPUT:  
