**Date-04/07/2024**

**Exam Questions:**

Q.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>

// Base class

class Shape {

public:

virtual void draw() const = 0;

virtual ~Shape() = default;

};

// Derived class Circle

class Circle : public Shape {

public:

void draw() const override {

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

}

};

// Derived class Rectangle

class Rectangle : public Shape {

public:

void draw() const override {

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

}

};

class Triangle : public Shape {

public:

void draw() const override {

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

}

};

void drawShape(const Shape\* shape) {

shape->draw();

}

int main() {

Circle circle;

Rectangle rectangle;

Triangle triangle;

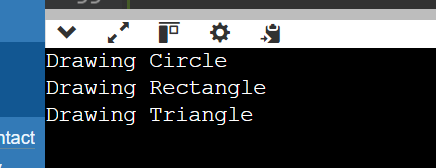
drawShape(&circle);

drawShape(&rectangle);

drawShape(&triangle);

return 0;

}



Q.2 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 {

public:

static int totalAccounts;

// Constructor

Account() {

totalAccounts++;

}

~Account() {

totalAccounts--;

}

static void displayTotalAccounts() {

std::cout << "Total Accounts: " << totalAccounts << std::endl;

}

};

int Account::totalAccounts = 0;

int main() {

Account a1;

Account::displayTotalAccounts();

{

Account a2, a3;

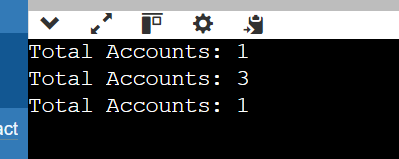
Account::displayTotalAccounts();

}

Account::displayTotalAccounts();

return 0;

}



Q.3 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:

double length;

double breadth;

double height;

public:

// Constructor to initialize the dimensions of the box

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

// Friend function to calculate and return the volume of the box

friend double volume(const Box& b);

};

// Friend function definition

double volume(const Box& b) {

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

}

int main() {

Box box1(3, 4, 5);

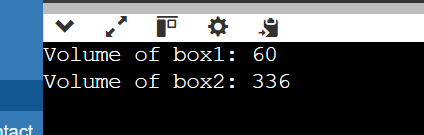
Box box2(6, 7, 8);

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

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

return 0;

}



Q.4 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 <vector>

template <typename T>

class Array {

private:

std::vector<T> elements;

public:

// Function to add an element to the array

void addElement(const T& element) {

elements.push\_back(element);

}

// Function to remove an element from the array by index

void removeElement(size\_t index) {

if (index < elements.size()) {

elements.erase(elements.begin() + index);

} else {

std::cout << "Index out of bounds" << std::endl;

}

}

// Function to display the array

void display() const {

for (const auto& element : elements) {

std::cout << element << " ";

}

std::cout << std::endl;

}

};

int main() {

// Demonstrate the functionality with int data type

Array<int> intArray;

intArray.addElement(1);

intArray.addElement(2);

intArray.addElement(3);

std::cout << "Int array: ";

intArray.display();

intArray.removeElement(1);

std::cout << "Int array after removing element at index 1: ";

intArray.display();

// Demonstrate the functionality with double data type

Array<double> doubleArray;

doubleArray.addElement(1.1);

doubleArray.addElement(2.2);

doubleArray.addElement(3.3);

std::cout << "Double array: ";

doubleArray.display();

doubleArray.removeElement(0);

std::cout << "Double array after removing element at index 0: ";

doubleArray.display();

// Demonstrate the functionality with std::string data type

Array<std::string> stringArray;

stringArray.addElement("Hello");

stringArray.addElement("World");

stringArray.addElement("!");

std::cout << "String array: ";

stringArray.display();

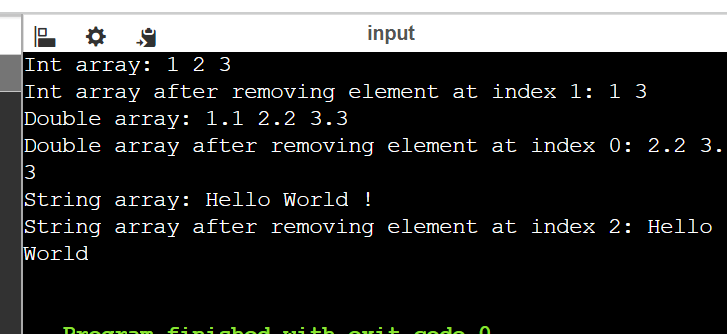
stringArray.removeElement(2);

std::cout << "String array after removing element at index 2: ";

stringArray.display();

return 0;

}



Q.5 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>

class Student {

private:

std::string name;

int age;

public:

// Function to set the details of the student

void setDetails(const std::string& studentName, int studentAge) {

name = studentName;

age = studentAge;

}

// Function to display the details of the student

void display() const {

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

}

};

int main() {

int numberOfStudents;

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

std::cin >> numberOfStudents;

// Dynamically allocate an array of Student objects

Student\* students = new Student[numberOfStudents];

// Set details for each student

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

std::string name;

int age;

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

std::cout << "Name: ";

std::cin >> name;

std::cout << "Age: ";

std::cin >> age;

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

}

// Display the details of each student

std::cout << "\nStudent Details:" << std::endl;

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

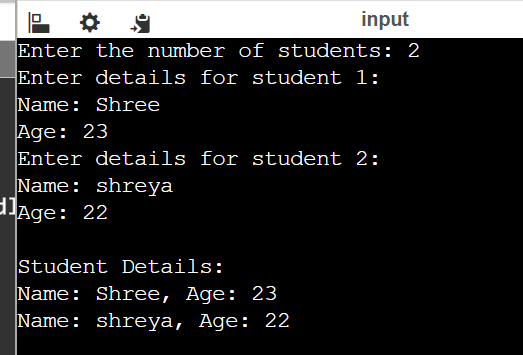
students[i].display();

}

delete[] students;

return 0;

}



Q.6 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>

class Animal {

public:

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

virtual ~Animal() = default; // Virtual destructor

};

class Dog : public Animal {

public:

void sound() const override {

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

}

};

class Cat : public Animal {

public:

void sound() const override {

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

}

};

class Cow : public Animal {

public:

void sound() const override {

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

}

};

void makeSound(const Animal& animal) {

animal.sound();

}

int main() {

Dog dog;

Cat cat;

Cow cow;

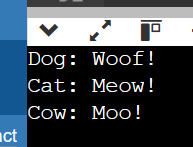
makeSound(dog);

makeSound(cat);

makeSound(cow);

return 0;

}



Q.7 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 member functions for basic mathematical operations

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(double a, double b) {

if (b != 0) {

return a / b;

} else {

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

return 0;

}

}

};

int main() {

// Demonstrate the use of static member functions without creating an object of the class

int a = 11, b = 55;

std::cout << "Addition: " << Math::add(a, b) << std::endl; // Output: 15

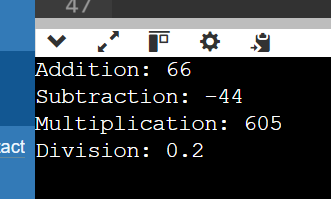
std::cout << "Subtraction: " << Math::subtract(a, b) << std::endl; // Output: 5

std::cout << "Multiplication: " << Math::multiply(a, b) << std::endl; // Output: 50

std::cout << "Division: " << Math::divide(a, b) << std::endl; // Output: 2

return 0;

}



Q.8 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>

class Beta;

class Alpha {

private:

int privateData;

public:

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

// Friend declaration for Beta class

friend class Beta;

};

// Class Beta

class Beta {

public:

void modifyAlphaData(Alpha& alpha, int newData) {

alpha.privateData = newData;

}

int accessAlphaData(const Alpha& alpha) const {

return alpha.privateData;

}

};

int main() {

Alpha alphaObject(10);

Beta betaObject;

// Accessing and modifying private data of Alpha using Beta

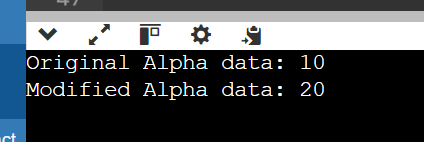
std::cout << "Original Alpha data: " << betaObject.accessAlphaData(alphaObject) << std::endl;

betaObject.modifyAlphaData(alphaObject, 20);

std::cout << "Modified Alpha data: " << betaObject.accessAlphaData(alphaObject) << std::endl;

return 0;

}



Q.9 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.

#include <iostream>

#include <string>

class Book {

private:

std::string title;

std::string author;

public:

// Constructor to initialize title and author

Book(const std::string& t, const std::string& a) : title(t), author(a) {}

// Function to set the details of the book

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

title = t;

author = a;

}

// Function to display the details of the book

void display() const {

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

}

// Function to get the title of the book

std::string getTitle() const {

return title;

}

};

int main() {

const int numBooks = 3; // Number of books

// Create an array of pointers to Book objects

Book\* books[numBooks];

// Input details for each book

std::string title, author;

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

std::cout << "Enter details for Book " << i + 1 << ":" << std::endl;

std::cout << "Title: ";

std::getline(std::cin >> std::ws, title); // Use getline to handle spaces in title

std::cout << "Author: ";

std::getline(std::cin >> std::ws, author); // Use getline to handle spaces in author

books[i] = new Book(title, author);

}

// Display details of all books

std::cout << "\nDetails of all books:" << std::endl;

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

books[i]->display();

}

// Search for a book by title

std::string searchTitle;

std::cout << "\nEnter the title to search: ";

std::getline(std::cin >> std::ws, searchTitle); // Use getline to handle spaces in searchTitle

bool found = false;

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

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

std::cout << "Book found:" << std::endl;

books[i]->display();

found = true;

break;

}

}

if (!found) {

std::cout << "Book with title \"" << searchTitle << "\" not found." << std::endl;

}

// Deallocate memory

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

delete books[i];

}

return 0;

}

