**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 ~Shape() {}

virtual void draw() const

{

std::cout << "Drawing a generic Shape\n";

}

};

class Circle : public Shape

{

public:

void draw() const override

{

std::cout << "Drawing a Circle\n";

}

};

class Rectangle : public Shape

{

public:

void draw() const override

{

std::cout << "Drawing a Rectangle\n";

}

};

class Triangle : public Shape

{

public:

void draw() const override

{

std::cout << "Drawing a Triangle\n";

}

};

void drawShape(const Shape\* shape)

{

shape->draw();

}

int main()

{

Circle circle;

Rectangle rectangle;

Triangle triangle;

drawShape(&circle);

drawShape(&rectangle);

drawShape(&triangle);

return 0;

}

Output:

Drawing a Circle

Drawing a Rectangle

Drawing a Triangle

**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()

{

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

}

};

int Account::totalAccounts = 0;

int main()

{

Account acc1;

Account acc2;

Account::displayTotalAccounts();

Account acc3;

Account::displayTotalAccounts(); // Output: Total accounts: 3

return 0;

}

Output:

**Total accounts: 2**

**Total accounts: 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:

Box(double l, double b, double h)

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

friend double volume(const Box& box);

};

double volume(const Box& box)

{

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

}

int main()

{

Box box1(3.0, 4.0, 5.0);

Box box2(1.5, 2.5, 3.5);

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

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

return 0;

}

**Output:**

**Volume of box1: 60**

**Volume of box2: 13.125**

**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>

Using namespace std;

template <typename T>

class Array

{

private:

std::vector<T> elements;

public:

Array() {}

~Array() {}

void addElement(const T& element)

{

elements.push\_back(element);

}

void removeElement(int index)

{

if (index >= 0 && index < elements.size())

{

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

}

else

{

std::cout << "Invalid index. Element removal failed.\n";

}

}

void display() const

{

std::cout << "Array elements:";

for (const auto& elem : elements)

{

std::cout << " " << elem;

}

std::cout << std::endl;

}

};

int main()

{

Array<int> intArray;

Array<double> doubleArray;

Array<std::string> stringArray;

intArray.addElement(10);

intArray.addElement(20);

intArray.addElement(30);

doubleArray.addElement(1.5);

doubleArray.addElement(2.5);

doubleArray.addElement(3.5);

stringArray.addElement("Hello");

stringArray.addElement("Mouni");

intArray.display();

doubleArray.display();

stringArray.display();

intArray.removeElement(1);

doubleArray.removeElement(2);

stringArray.removeElement(0);

intArray.display();

doubleArray.display();

stringArray.display();

return 0;

}

**Output:**

**Array elements: 10 20 30**

**Array elements: 1.5 2.5 3.5**

**Array elements: Hello Mouni**

**Array elements: 10 30**

**Array elements: 1.5 2.5**

**Array elements: Mouni**

**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:

std::string name;

int age;

public:

Student() {}

void setName(const std::string& name)

{

this->name = name;

}

void setAge(int age)

{

this->age = age;

}

std::string getName() const

{

return name;

}

int getAge() const

{

return age;

}

};

void setDetails(Student\* students, int size)

{

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

{

std::string name;

int age;

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

std::cin >> name;

students[i].setName(name);

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

std::cin >> age;

students[i].setAge(age);

}

}

void displayDetails(const Student\* students, int size)

{

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

{

std::cout << "Student " << i + 1 << " - Name: " << students[i].getName() << ", Age: " << students[i].getAge() << std::endl;

}

}

void deallocateMemory(Student\* students)

{

delete[] students;

}

int main()

{

int numStudents;

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

std::cin >> numStudents;

Student\* students = new Student[numStudents];

setDetails(students, numStudents);

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

displayDetails(students, numStudents);

deallocateMemory(students);

return 0;

}

**Output:**

**Enter number of students: 3**

**Enter name for student 1: Alice**

**Enter age for student 1: 20**

**Enter name for student 2: Bob**

**Enter age for student 2: 21**

**Enter name for student 3: Carol**

**Enter age for student 3: 22**

**Student Details:**

**Student 1 - Name: Alice, Age: 20**

**Student 2 - Name: Bob, Age: 21**

**Student 3 - Name: Carol, Age: 22**

**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>

#include <vector>

using namespace std;

class Animal

{

public:

virtual void sound() const = 0;

};

class Dog : public Animal

{

public:

void sound() const override

{

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

}

};

class Cat : public Animal

{

public:

void sound() const override

{

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

}

};

class Cow : public Animal

{

public:

void sound() const override

{

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

}

};

int main()

{

std::vector<Animal\*> animals;

Dog dog;

Cat cat;

Cow cow;

animals.push\_back(&dog);

animals.push\_back(&cat);

animals.push\_back(&cow);

for (const auto& animal : animals)

{

animal->sound();

}

return 0;

}

**Output:**

**Dog says: Woof!**

**Cat says: Meow!**

**Cow says: Moo!**

**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(double a, double b)

{

if (b != 0)

{

return a / b;

}

else

{

std::cerr << "Error: Division by zero\n";

return 0;

}

}

};

int main()

{

int num1 = 10, num2 = 5;

std::cout << "Addition: " << Math::add(num1, num2) << std::endl;

std::cout << "Subtraction: " << Math::subtract(num1, num2) << std::endl;

std::cout << "Multiplication: " << Math::multiply(num1, num2) << std::endl;

std::cout << "Division: " << Math::divide(num1, num2) << std::endl;

return 0;

}

**Output:**

**Addition: 15**

**Subtraction: 5**

**Multiplication: 50**

**Division: 2**

**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 data;

public:

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

friend class Beta;

};

class Beta

{

public:

void setData(Alpha& obj, int value)

{

obj.data = value;

}

void displayData(const Alpha& obj)

{

std::cout << "Data in Alpha: " << obj.data << std::endl;

}

};

int main()

{

Alpha alphaObj(100);

Beta betaObj;

betaObj.displayData(alphaObj);

betaObj.setData(alphaObj, 200);

betaObj.displayData(alphaObj);

return 0;

}

**Output:**

**Data in Alpha: 100**

**Data in Alpha: 200**

**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.**

#include <iostream>

using namespace std;

template <typename T1, typename T2>

class Pair

{

private:

T1 first;

T2 second;

public:

Pair(const T1& f, const T2& s) : first(f), second(s) {}

void setFirst(const T1& f)

{

first = f;

}

void setSecond(const T2& s) {

second = s;

}

T1 getFirst() const {

return first;

}

T2 getSecond() const {

return second;

}

};

int main()

{

Pair<int, double> pair1(10, 3.14);

pair1.setFirst(20);

pair1.setSecond(6.28);

std::cout << "Pair1 - First: " << pair1.getFirst() << ", Second: " << pair1.getSecond() << std::endl;

Pair<std::string, char> pair2("Hello", 'C');

pair2.setFirst("Goodbye");

pair2.setSecond('D');

std::cout << "Pair2 - First: " << pair2.getFirst() << ", Second: " << pair2.getSecond() << std::endl;

return 0;

}

**Output:**

**Pair1 - First: 20, Second: 6.28**

**Pair2 - First: Goodbye, Second: D**

**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>

using namespace std;

class Book

{

private:

std::string title;

std::string author;

public:

Book() : title(""), author("") {}

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

std::string getTitle() const

{

return title;

}

std::string getAuthor() const

{

return author;

}

void setTitle(const std::string& t)

{

title = t;

}

void setAuthor(const std::string& a)

{

author = a;

}

void inputDetails()

{

std::cout << "Enter title: ";

std::getline(std::cin, title);

std::cout << "Enter author: ";

std::getline(std::cin, author);

}

void displayDetails() const

{

std::cout << "Title: " << title << std::endl;

std::cout << "Author: " << author << std::endl;

}

};

int main()

{

const int numBooks = 3;

Book\* library[numBooks];

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

{

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

library[i] = new Book();

library[i]->inputDetails();

}

std::cout << "\nLibrary Contents:" << std::endl;

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

{

std::cout << "Book " << i + 1 << ":" << std::endl;

library[i]->displayDetails();

std::cout << std::endl;

}

std::string searchTitle;

std::cout << "Enter the title to search for: ";

std::getline(std::cin, searchTitle);

bool found = false;

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

{

if (library[i]->getTitle() == searchTitle)

{

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

library[i]->displayDetails();

found = true;

break;

}

}

if (!found)

{

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

}

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

{

delete library[i];

}

return 0;

}

**Output:**

**Enter details for Book 1:**

**Enter title: The Great Gatsby**

**Enter author: F. Scott Fitzgerald**

**Enter details for Book 2:**

**Enter title: To Kill a Mockingbird**

**Enter author: Harper Lee**

**Enter details for Book 3:**

**Enter title: 1984**

**Enter author: George Orwell**

**Library Contents:**

**Book 1:**

**Title: The Great Gatsby**

**Author: F. Scott Fitzgerald**

**Book 2:**

**Title: To Kill a Mockingbird**

**Author: Harper Lee**

**Book 3:**

**Title: 1984**

**Author: George Orwell**

**Enter the title to search for: To Kill a Mockingbird**

**Book found:**

**Title: To Kill a Mockingbird**

**Author: Harper Lee**