1.) #include <iostream>

using namespace std;

// Base class

class Shape {

public:

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

virtual ~Shape() {} // Virtual destructor

};

// 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;

}

};

// Derived class Triangle

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 c;

Rectangle r;

Triangle t;

Shape\* shapes[] = { &c, &r, &t };

for (const auto& shape : shapes)

{

drawShape(shape);

}

return 0;

}

2.) #include <iostream>

using namespace std;

class Account {

private:

// Static data

static int totalAccounts;

public:

// Constructor

Account() {

++totalAccounts;

}

// Destructor

~Account() {

--totalAccounts;

}

// to display all accounts

static void printallAccounts() {

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

}

};

// Initialize static data member

int Account::totalAccounts = 0;

int main() {

// Creating Account objects

Account a1,a2,a3;

Account::printallAccounts(); // Total Accounts: 1

{

Account a2, a5, a7,a9;

Account::printallAccounts(); // Total Accounts: 3

} // a2,a5,a7,a9 are destroyed

Account::printallAccounts(); // Total Accounts: 1

return 0;

}

3.) #include <iostream>

using namespace std;

// Class Box

class Box {

private:

int length;

int breadth;

int height;

public:

// Constructor

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

// Friend function declaration

friend int volume(const Box& b);

};

// Friend function definition

int volume(const Box& b) {

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

}

5.)#include <iostream>

using namespace std;

class Student {

private:

string name;

int age;

public:

// Constructor

Student() : name("Haritha"), age(23) {}

// Function to set the details of the student

void Details(const std::string& stuName, int stuAge) {

name = stuName;

age = stuAge;

}

// Function to display the details of the student

void printDetails() const {

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

}

};

// Function to deallocate memory for the array of students

void deallocateMemory(Student\* students) {

delete[] students;

}

int main() {

int numStudents;

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

std::cin >> numStudents;

// Dynamically create an array of Student objects

Student\* students = new Student[numStudents];

// Set the details for each student

for (int i = 0; i < numStudents; ++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].Details(name, age);

}

// Display the details of each student

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

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

students[i].printDetails();

}

// Deallocate memory

deallocateMemory(students);

return 0;

}

int main() {

// Create Box objects

Box box1(7.0, 1.0, 7.0);

Box box2(8.0, 3.0, 8.0);

// to print volume of the boxes

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

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

return 0;

}

6.) #include <iostream>

#include <string>

using namespace std;

class Animal { // Abstract class Animal

public:

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

};

class Dog : public Animal { // Derived class Dog from Animal

public:

void sound() const override {

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

}

};

class Cat : public Animal { // Derived class Cat from Animal

public:

void sound() const override {

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

}

};

class Cow : public Animal { // Derived class Cow from Animal

public:

void sound() const override {

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

}

};

int main() {

Animal\* animals[3]; // Declare an array of Animal pointers

Dog dog; // Instantiate objects of Dog, Cat, and Cow

Cat cat;

Cow cow;

animals[0] = &dog; // Assign addresses of objects to Animal pointers

animals[1] = &cat;

animals[2] = &cow;

for (int i = 0; i < 3; ++i) { // Use polymorphism to call the sound() function

animals[i]->sound();

}

return 0;

}

7.) #include <iostream>

class Math {

public:

static int add(int a, int b) { // Static member functions for basic math operations

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) {

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

return 0;

}

return a / b;

}

};

int main() {

std::cout << "Addition: " << Math::add(10,5) << std::endl; // Using static member functions without creating an object of the class

std::cout << "Subtraction: " << Math::subtract(10, 5) << std::endl;

std::cout << "Multiplication: " << Math::multiply(20, 5) << std::endl;

std::cout << "Division: " << Math::divide(20.0, 5.0) << std::endl;

return 0;

}

8.) #include <iostream>

using namespace std;

class Beta;

class Alpha {

private:

int data;

public:

// Constructor to initialize data

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

void display() const {

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

}

// Friend class declaration

friend class Beta;

};

class Beta {

public:

void setData(Alpha& a, int value) {

a.data = value;

}

void doubleData(Alpha& a) {

a.data \*= 2;

}

};

int main() {

Alpha a(25);

a.display(); // Alpha data: 10

Beta b;

b.setData(a, 25);

a.display(); // Alpha data: 20

b.doubleData(a);

a.display(); // Alpha data: 40

return 0;

}

9.) #include <iostream>

using namespace std;

template <typename T1, typename T2>

class Pair {

private:

T1 first;

T2 second;

public:

// Constructor

Pair(const T1& firstValue, const T2& secondValue) : first(firstValue), second(secondValue) {}

// Default constructor

Pair() : first(T1()), second(T2()) {}

// to set the values

void setFirst(const T1& firstValue) {

first = firstValue;

}

void setSecond(const T2& secondValue) {

second = secondValue;

}

T1 getFirst() const {

return first;

}

T2 getSecond() const {

return second;

}

// to display the values

void display() const {

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

}

};

int main() {

Pair<int, double> intDoublePair(1, 2.5);

intDoublePair.display(); // First: 1, Second: 2.5

intDoublePair.setFirst(3);

intDoublePair.setSecond(4.5);

intDoublePair.display();

Pair<std::string, char> stringCharPair("Hello", 'A');

stringCharPair.display();

stringCharPair.setFirst("World");

stringCharPair.setSecond('B');

stringCharPair.display();

Pair<float, bool> floatBoolPair(1.5f, true);

floatBoolPair.display();

floatBoolPair.setFirst(2.5f);

floatBoolPair.setSecond(false);

floatBoolPair.display();

return 0;

}

10.) #include <iostream>

#include <string>

// Class Book

class Book {

private:

std::string title;

std::string author;

public:

// Constructor

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

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

title = t;

author = a;

}

void displayDetails() const {

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

}

std::string getTitle() const {

return title;

}

};

// Function to input details for each book

void inputDetails(Book\* books[], int size) {

std::string title, author;

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

std::cout << "Enter details for book " << (i + 1) << ":" << std::endl;

std::cout << "Title: ";

std::cin.ignore();

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

std::cout << "Author: ";

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

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

}

}

// Function to display details of all books

void displayAllBooks(Book\* books[], int size) {

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

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

books[i]->displayDetails();

}

}

void searchBookByTitle(Book\* books[], int size, const std::string& title) {

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

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

std::cout << "Book found:\n";

books[i]->displayDetails();

return;

}

}

std::cout << "Book not found.\n";

}

void deallocateMemory(Book\* books[], int size) {

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

delete books[i];

}

}

int main() {

int numBooks;

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

std::cin >> numBooks;

Book\* books[numBooks];

inputDetails(books, numBooks);

// Display details of all books

displayAllBooks(books, numBooks);

std::string titleToSearch;

std::cout << "\nEnter the title of the book : ";

std::cin.ignore();

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

searchBookByTitle(books, numBooks, titleToSearch);

deallocateMemory(books, numBooks);

return 0;

}