**Unit 4 code assignment**

**51. Base Class Shape with Derived Classes Circle, Rectangle, and Triangle**

cpp

CopyEdit

#include <iostream>

using namespace std;

class Shape {

public:

virtual double calculateArea() = 0; // Pure virtual function

};

class Circle : public Shape {

private:

double radius;

public:

Circle(double r) : radius(r) {}

double calculateArea() override {

return 3.14159 \* radius \* radius;

}

};

class Rectangle : public Shape {

private:

double length, width;

public:

Rectangle(double l, double w) : length(l), width(w) {}

double calculateArea() override {

return length \* width;

}

};

class Triangle : public Shape {

private:

double base, height;

public:

Triangle(double b, double h) : base(b), height(h) {}

double calculateArea() override {

return 0.5 \* base \* height;

}

};

int main() {

Shape\* shapes[3];

shapes[0] = new Circle(5);

shapes[1] = new Rectangle(4, 6);

shapes[2] = new Triangle(4, 3);

for (int i = 0; i < 3; i++) {

cout << "Area: " << shapes[i]->calculateArea() << endl;

}

return 0;

}

**52. Base Class Animal with Virtual Function speak()**

cpp

CopyEdit

#include <iostream>

using namespace std;

class Animal {

public:

virtual void speak() {

cout << "Animal speaks." << endl;

}

};

class Dog : public Animal {

public:

void speak() override {

cout << "Dog barks." << endl;

}

};

class Cat : public Animal {

public:

void speak() override {

cout << "Cat meows." << endl;

}

};

class Bird : public Animal {

public:

void speak() override {

cout << "Bird chirps." << endl;

}

};

int main() {

Animal\* animals[3];

animals[0] = new Dog();

animals[1] = new Cat();

animals[2] = new Bird();

for (int i = 0; i < 3; i++) {

animals[i]->speak();

}

return 0;

}

**53. Program Demonstrating Function Overriding (Employee, Manager, and Worker)**

cpp

CopyEdit

#include <iostream>

using namespace std;

class Employee {

public:

virtual void work() {

cout << "Employee works." << endl;

}

};

class Manager : public Employee {

public:

void work() override {

cout << "Manager manages." << endl;

}

};

class Worker : public Employee {

public:

void work() override {

cout << "Worker works on tasks." << endl;

}

};

int main() {

Employee\* employees[2];

employees[0] = new Manager();

employees[1] = new Worker();

for (int i = 0; i < 2; i++) {

employees[i]->work();

}

return 0;

}

**54. Pointer Arithmetic with an Array**

cpp

CopyEdit

#include <iostream>

using namespace std;

int main() {

int arr[] = {10, 20, 30, 40, 50};

int\* ptr = arr; // Pointer to the first element

cout << "Accessing array using pointer arithmetic:" << endl;

for (int i = 0; i < 5; i++) {

cout << \*(ptr + i) << " "; // Accessing array elements using pointer arithmetic

}

return 0;

}

**55. Dynamically Allocating Memory for an Integer Array**

cpp

CopyEdit

#include <iostream>

using namespace std;

int main() {

int size;

cout << "Enter the size of the array: ";

cin >> size;

// Dynamically allocating memory for an integer array

int\* arr = new int[size];

// Initializing the array

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

arr[i] = i \* 10; // Assigning values

}

// Displaying the array values

cout << "Array elements: ";

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

cout << arr[i] << " ";

}

cout << endl;

// Deallocating the memory

delete[] arr;

return 0;

}

Would you like me to proceed with **56 to 60**?

**You said:**

yes

**ChatGPT said:**

Here are the solutions for problems **56** to **60**:

**56. Using a Pointer to Swap the Values of Two Variables**

cpp

CopyEdit

#include <iostream>

using namespace std;

void swap(int\* a, int\* b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main() {

int x = 10, y = 20;

cout << "Before swap: x = " << x << ", y = " << y << endl;

swap(&x, &y); // Passing the addresses of x and y

cout << "After swap: x = " << x << ", y = " << y << endl;

return 0;

}

**57. Creating a Dynamic Object of a Class Student and Accessing Its Members Using Pointers**

cpp

CopyEdit

#include <iostream>

using namespace std;

class Student {

private:

string name;

int age;

public:

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

void display() {

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

}

};

int main() {

// Dynamically creating an object of Student using pointer

Student\* studentPtr = new Student("John Doe", 21);

// Accessing member function using pointer

studentPtr->display();

// Deleting dynamically allocated memory

delete studentPtr;

return 0;

}

**58. Using a Pointer to an Array of Objects to Store and Display Details of Multiple Book Objects**

cpp

CopyEdit

#include <iostream>

#include <string>

using namespace std;

class Book {

private:

string title;

string author;

public:

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

void display() {

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

}

};

int main() {

// Dynamically allocating memory for an array of Book objects

Book\* books = new Book[3] {

Book("C++ Primer", "Stanley B. Lippman"),

Book("Effective C++", "Scott Meyers"),

Book("The C++ Programming Language", "Bjarne Stroustrup")

};

// Accessing and displaying details of each book using pointer arithmetic

for (int i = 0; i < 3; i++) {

(books + i)->display(); // Accessing objects using pointer arithmetic

}

// Deallocating memory

delete[] books;

return 0;

}

**59. Using a Pointer to an Object in a Class Member Function**

cpp

CopyEdit

#include <iostream>

using namespace std;

class Box {

private:

double length, width, height;

public:

Box(double l, double w, double h) : length(l), width(w), height(h) {}

void displayDetails() {

cout << "Length: " << length << ", Width: " << width << ", Height: " << height << endl;

}

void\* operator new(size\_t size) {

cout << "Allocating memory for Box object." << endl;

return ::operator new(size); // Using global new operator

}

void operator delete(void\* pointer) {

cout << "Deallocating memory for Box object." << endl;

::operator delete(pointer); // Using global delete operator

}

};

int main() {

Box\* boxPtr = new Box(5.0, 3.0, 2.0);

boxPtr->displayDetails();

// Deleting dynamically allocated memory

delete boxPtr;

return 0;

}

**60. Returning the Current Object Using the this Pointer**

cpp

CopyEdit

#include <iostream>

using namespace std;

class Box {

private:

double length, width, height;

public:

Box(double l, double w, double h) : length(l), width(w), height(h) {}

Box\* setDimensions(double l, double w, double h) {

length = l;

width = w;

height = h;

return this; // Returning the current object

}

void displayDetails() {

cout << "Length: " << length << ", Width: " << width << ", Height: " << height << endl;

}

};

int main() {

Box box(5.0, 3.0, 2.0);

// Chaining member function calls using 'this' pointer

box.setDimensions(7.0, 4.0, 3.0)->displayDetails();

return 0;

}

Would you like to proceed with **61 to 65**?

**You said:**

yes

**ChatGPT said:**

Here are the solutions for problems **61** to **65**:

**61. Using the this Pointer to Chain Member Function Calls in a Class Person**

cpp

CopyEdit

#include <iostream>

using namespace std;

class Person {

private:

string name;

int age;

public:

Person(string n, int a) : name(n), age(a) {}

// Chaining member functions using the `this` pointer

Person\* setName(string n) {

name = n;

return this;

}

Person\* setAge(int a) {

age = a;

return this;

}

void displayDetails() {

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

}

};

int main() {

Person p("John", 30);

p.setName("Alice")->setAge(25)->displayDetails(); // Chaining function calls

return 0;

}

**62. Creating a Class Counter to Compare Two Objects Using the this Pointer**

cpp

CopyEdit

#include <iostream>

using namespace std;

class Counter {

private:

int count;

public:

Counter(int c) : count(c) {}

bool compare(Counter\* other) {

return this->count == other->count;

}

void display() {

cout << "Count: " << count << endl;

}

};

int main() {

Counter c1(10), c2(10), c3(5);

cout << "Comparing c1 and c2: " << (c1.compare(&c2) ? "Equal" : "Not Equal") << endl;

cout << "Comparing c1 and c3: " << (c1.compare(&c3) ? "Equal" : "Not Equal") << endl;

return 0;

}

**63. Using Pure Virtual Functions to Create an Abstract Class Vehicle with Derived Classes Car and Bike**

cpp

CopyEdit

#include <iostream>

using namespace std;

class Vehicle {

public:

virtual void move() = 0; // Pure virtual function

};

class Car : public Vehicle {

public:

void move() override {

cout << "Car is moving." << endl;

}

};

class Bike : public Vehicle {

public:

void move() override {

cout << "Bike is moving." << endl;

}

};

int main() {

Vehicle\* v1 = new Car();

Vehicle\* v2 = new Bike();

v1->move();

v2->move();

delete v1;

delete v2;

return 0;

}

**64. Demonstrating Runtime Polymorphism Using a Virtual Function in a Base Class Shape and Derived Classes Circle and Square**

cpp

CopyEdit

#include <iostream>

using namespace std;

class Shape {

public:

virtual void draw() {

cout << "Drawing shape" << endl;

}

};

class Circle : public Shape {

public:

void draw() override {

cout << "Drawing circle" << endl;

}

};

class Square : public Shape {

public:

void draw() override {

cout << "Drawing square" << endl;

}

};

int main() {

Shape\* shape1 = new Circle();

Shape\* shape2 = new Square();

shape1->draw(); // Runtime polymorphism

shape2->draw(); // Runtime polymorphism

delete shape1;

delete shape2;

return 0;

}

**65. Creating a Class Account with a Pure Virtual Function calculateInterest() and Derived Classes SavingsAccount and CurrentAccount**

cpp

CopyEdit

#include <iostream>

using namespace std;

class Account {

public:

virtual void calculateInterest() = 0; // Pure virtual function

};

class SavingsAccount : public Account {

private:

double balance;

public:

SavingsAccount(double bal) : balance(bal) {}

void calculateInterest() override {

double interest = balance \* 0.04;

cout << "Interest for Savings Account: " << interest << endl;

}

};

class CurrentAccount : public Account {

private:

double balance;

public:

CurrentAccount(double bal) : balance(bal) {}

void calculateInterest() override {

double interest = balance \* 0.02;

cout << "Interest for Current Account: " << interest << endl;

}

};

int main() {

Account\* acc1 = new SavingsAccount(1000);

Account\* acc2 = new CurrentAccount(1000);

acc1->calculateInterest();

acc2->calculateInterest();

delete acc1;

delete acc2;

return 0;

}

Would you like me to proceed with **66 to 70**?

**You said:**

yes

**ChatGPT said:**

Here are the solutions for problems **66** to **70**:

**66. Demonstrating Polymorphism Using a Base Class Media and Derived Classes Book and DVD**

cpp

CopyEdit

#include <iostream>

using namespace std;

class Media {

public:

virtual void display() = 0; // Pure virtual function

};

class Book : public Media {

private:

string title;

public:

Book(string t) : title(t) {}

void display() override {

cout << "Book: " << title << endl;

}

};

class DVD : public Media {

private:

string title;

public:

DVD(string t) : title(t) {}

void display() override {

cout << "DVD: " << title << endl;

}

};

int main() {

Media\* media1 = new Book("The C++ Programming Language");

Media\* media2 = new DVD("Inception");

media1->display();

media2->display();

delete media1;

delete media2;

return 0;

}

**67. Implementing a Class Hierarchy with a Base Class Appliance and Derived Classes WashingMachine, Refrigerator, and Microwave**

cpp

CopyEdit

#include <iostream>

using namespace std;

class Appliance {

public:

virtual void functionality() = 0; // Pure virtual function

};

class WashingMachine : public Appliance {

public:

void functionality() override {

cout << "Washing clothes" << endl;

}

};

class Refrigerator : public Appliance {

public:

void functionality() override {

cout << "Preserving food" << endl;

}

};

class Microwave : public Appliance {

public:

void functionality() override {

cout << "Heating food" << endl;

}

};

int main() {

Appliance\* appliances[3];

appliances[0] = new WashingMachine();

appliances[1] = new Refrigerator();

appliances[2] = new Microwave();

for (int i = 0; i < 3; i++) {

appliances[i]->functionality();

delete appliances[i];

}

return 0;

}

**68. Using Polymorphism to Calculate the Area of Different Geometric Shapes Using a Base Class Shape and Derived Classes Circle and Rectangle**

cpp

CopyEdit

#include <iostream>

using namespace std;

class Shape {

public:

virtual double area() = 0; // Pure virtual function

};

class Circle : public Shape {

private:

double radius;

public:

Circle(double r) : radius(r) {}

double area() override {

return 3.14 \* radius \* radius;

}

};

class Rectangle : public Shape {

private:

double length, width;

public:

Rectangle(double l, double w) : length(l), width(w) {}

double area() override {

return length \* width;

}

};

int main() {

Shape\* shape1 = new Circle(5);

Shape\* shape2 = new Rectangle(4, 6);

cout << "Area of Circle: " << shape1->area() << endl;

cout << "Area of Rectangle: " << shape2->area() << endl;

delete shape1;

delete shape2;

return 0;

}

**69. Creating an Abstract Class Employee with Pure Virtual Functions calculateSalary() and displayDetails(). Implement Derived Classes Manager and Engineer**

cpp

CopyEdit

#include <iostream>

using namespace std;

class Employee {

public:

virtual void calculateSalary() = 0; // Pure virtual function

virtual void displayDetails() = 0; // Pure virtual function

};

class Manager : public Employee {

private:

double salary;

public:

Manager(double sal) : salary(sal) {}

void calculateSalary() override {

salary += 1000; // Adding bonus

}

void displayDetails() override {

cout << "Manager Salary: " << salary << endl;

}

};

class Engineer : public Employee {

private:

double salary;

public:

Engineer(double sal) : salary(sal) {}

void calculateSalary() override {

salary += 500; // Adding bonus

}

void displayDetails() override {

cout << "Engineer Salary: " << salary << endl;

}

};

int main() {

Employee\* e1 = new Manager(5000);

Employee\* e2 = new Engineer(4000);

e1->calculateSalary();

e2->calculateSalary();

e1->displayDetails();

e2->displayDetails();

delete e1;

delete e2;

return 0;

}

**70. Implementing a Pure Virtual Function processPayment() in an Abstract Class Payment with Derived Classes CreditCardPayment and DebitCardPayment**

cpp

CopyEdit

#include <iostream>

using namespace std;

class Payment {

public:

virtual void processPayment() = 0; // Pure virtual function

};

class CreditCardPayment : public Payment {

public:

void processPayment() override {

cout << "Processing Credit Card Payment" << endl;

}

};

class DebitCardPayment : public Payment {

public:

void processPayment() override {

cout << "Processing Debit Card Payment" << endl;

}

};

int main() {

Payment\* payment1 = new CreditCardPayment();

Payment\* payment2 = new DebitCardPayment();

payment1->processPayment();

payment2->processPayment();

delete payment1;

delete payment2;

return 0;

}

Would you like me to proceed with **71 to 75**?

**You said:**

yes

**ChatGPT said:**

Here are the solutions for problems **71** to **75**:

**71. Creating an Abstract Class Device with a Pure Virtual Function turnOn(). Implement Derived Classes Laptop and Smartphone**

cpp

CopyEdit

#include <iostream>

using namespace std;

class Device {

public:

virtual void turnOn() = 0; // Pure virtual function

};

class Laptop : public Device {

public:

void turnOn() override {

cout << "Laptop is turning on." << endl;

}

};

class Smartphone : public Device {

public:

void turnOn() override {

cout << "Smartphone is turning on." << endl;

}

};

int main() {

Device\* d1 = new Laptop();

Device\* d2 = new Smartphone();

d1->turnOn();

d2->turnOn();

delete d1;

delete d2;

return 0;

}

**72. Handling Division by Zero Using Exception Handling**

cpp

CopyEdit

#include <iostream>

#include <stdexcept>

using namespace std;

double divide(int a, int b) {

if (b == 0) {

throw runtime\_error("Division by zero is not allowed.");

}

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

}

int main() {

try {

int num1 = 10, num2 = 0;

cout << "Result: " << divide(num1, num2) << endl;

} catch (const runtime\_error& e) {

cout << "Error: " << e.what() << endl;

}

return 0;

}

**73. Using Multiple Catch Blocks to Handle Different Types of Exceptions**

cpp

CopyEdit

#include <iostream>

#include <stdexcept>

using namespace std;

void testExceptions(int choice) {

if (choice == 1) {

throw runtime\_error("Runtime error occurred.");

} else if (choice == 2) {

throw logic\_error("Logic error occurred.");

} else {

throw exception();

}

}

int main() {

try {

testExceptions(1);

} catch (const runtime\_error& e) {

cout << "Caught runtime error: " << e.what() << endl;

} catch (const logic\_error& e) {

cout << "Caught logic error: " << e.what() << endl;

} catch (const exception& e) {

cout << "Caught a general exception." << endl;

}

return 0;

}

**74. Creating a Custom Exception Class InvalidAgeException to Handle Invalid Age Input**

cpp

CopyEdit

#include <iostream>

#include <stdexcept>

using namespace std;

class InvalidAgeException : public exception {

public:

const char\* what() const noexcept override {

return "Invalid age input! Age must be greater than 0.";

}

};

void checkAge(int age) {

if (age <= 0) {

throw InvalidAgeException();

}

cout << "Age is valid: " << age << endl;

}

int main() {

try {

checkAge(-5); // Invalid age

} catch (const InvalidAgeException& e) {

cout << "Error: " << e.what() << endl;

}

return 0;

}

**75. Using Exception Handling to Manage File Input/Output Errors**

cpp

CopyEdit

#include <iostream>

#include <fstream>

#include <stdexcept>

using namespace std;

void readFile(const string& filename) {

ifstream file(filename);

if (!file) {

throw runtime\_error("Failed to open file.");

}

string line;

while (getline(file, line)) {

cout << line << endl;

}

}

int main() {

try {

readFile("nonexistentfile.txt"); // File does not exist

} catch (const runtime\_error& e) {

cout << "File error: " << e.what() << endl;

}

return 0;

}

Would you like me to proceed with **76 to 80**?

**You said:**

yes

**ChatGPT said:**

Here are the solutions for problems **76** to **80**:

**76. Implementing a Program that Demonstrates the Use of the finally Block to Release Resources in Exception Handling**

In C++, there is no explicit finally block as in languages like Java. However, you can achieve similar functionality by using destructors or RAII (Resource Acquisition Is Initialization). Here's an example using destructors to release resources:

cpp

CopyEdit

#include <iostream>

#include <stdexcept>

using namespace std;

class Resource {

public:

Resource() {

cout << "Resource allocated." << endl;

}

~Resource() {

cout << "Resource released." << endl;

}

};

void process() {

Resource r;

cout << "Processing resource..." << endl;

throw runtime\_error("An error occurred during processing.");

}

int main() {

try {

process();

} catch (const runtime\_error& e) {

cout << "Error: " << e.what() << endl;

}

// Destructor will automatically release the resource

return 0;

}

In this example, the resource is released when the Resource object goes out of scope (destructor is called).

**77. Writing a Function Template to Perform a Linear Search on an Array of Any Data Type**

cpp

CopyEdit

#include <iostream>

using namespace std;

template <typename T>

int linearSearch(T arr[], int size, T key) {

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

if (arr[i] == key) {

return i;

}

}

return -1; // Not found

}

int main() {

int arrInt[] = {1, 2, 3, 4, 5};

double arrDouble[] = {1.1, 2.2, 3.3, 4.4, 5.5};

cout << "Index of 3 in int array: " << linearSearch(arrInt, 5, 3) << endl;

cout << "Index of 3.3 in double array: " << linearSearch(arrDouble, 5, 3.3) << endl;

return 0;

}

**78. Implementing a Class Template Stack with Member Functions to Push, Pop, and Display Elements**

cpp

CopyEdit

#include <iostream>

#include <vector>

using namespace std;

template <typename T>

class Stack {

private:

vector<T> stack;

public:

void push(T value) {

stack.push\_back(value);

}

void pop() {

if (!stack.empty()) {

stack.pop\_back();

} else {

cout << "Stack is empty!" << endl;

}

}

void display() {

if (stack.empty()) {

cout << "Stack is empty!" << endl;

return;

}

for (T value : stack) {

cout << value << " ";

}

cout << endl;

}

};

int main() {

Stack<int> intStack;

intStack.push(10);

intStack.push(20);

intStack.push(30);

cout << "Stack elements: ";

intStack.display();

intStack.pop();

cout << "Stack after pop: ";

intStack.display();

return 0;

}

**79. Creating a Function Template to Find the Maximum of Two Values of Any Data Type**

cpp

CopyEdit

#include <iostream>

using namespace std;

template <typename T>

T findMax(T a, T b) {

return (a > b) ? a : b;

}

int main() {

cout << "Max of 3 and 7: " << findMax(3, 7) << endl;

cout << "Max of 3.14 and 2.71: " << findMax(3.14, 2.71) << endl;

cout << "Max of 'a' and 'z': " << findMax('a', 'z') << endl;

return 0;

}

**80. Writing a Class Template LinkedList with Member Functions to Insert, Delete, and Display Nodes**

cpp

CopyEdit

#include <iostream>

using namespace std;

template <typename T>

class LinkedList {

private:

struct Node {

T data;

Node\* next;

};

Node\* head;

public:

LinkedList() : head(nullptr) {}

void insert(T value) {

Node\* newNode = new Node{value, head};

head = newNode;

}

void deleteValue(T value) {

Node\* temp = head;

Node\* prev = nullptr;

while (temp != nullptr && temp->data != value) {

prev = temp;

temp = temp->next;

}

if (temp == nullptr) {

cout << "Value not found in list!" << endl;

return;

}

if (prev == nullptr) {

head = temp->next;

} else {

prev->next = temp->next;

}

delete temp;

}

void display() {

Node\* temp = head;

while (temp != nullptr) {

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

~LinkedList() {

Node\* temp = head;

while (temp != nullptr) {

Node\* toDelete = temp;

temp = temp->next;

delete toDelete;

}

}

};

int main() {

LinkedList<int> list;

list.insert(10);

list.insert(20);

list.insert(30);

cout << "Linked List: ";

list.display();

list.deleteValue(20);

cout << "After deleting 20: ";

list.display();

return 0;

}

**81. Implementing a Function Template to Perform Bubble Sort on an Array of Any Data Type**

cpp

CopyEdit

#include <iostream>

using namespace std;

template <typename T>

void bubbleSort(T arr[], int size) {

for (int i = 0; i < size - 1; i++) {

for (int j = 0; j < size - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

T temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

template <typename T>

void displayArray(T arr[], int size) {

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

cout << arr[i] << " ";

}

cout << endl;

}

int main() {

int arrInt[] = {5, 2, 9, 1, 5, 6};

double arrDouble[] = {3.1, 2.4, 1.8, 4.7, 5.2};

cout << "Before sorting integer array: ";

displayArray(arrInt, 6);

bubbleSort(arrInt, 6);

cout << "After sorting integer array: ";

displayArray(arrInt, 6);

cout << "Before sorting double array: ";

displayArray(arrDouble, 5);

bubbleSort(arrDouble, 5);

cout << "After sorting double array: ";

displayArray(arrDouble, 5);

return 0;

}

**82. Creating a Class Template Queue with Member Functions to Enqueue, Dequeue, and Display Elements**

cpp

CopyEdit

#include <iostream>

#include <queue>

using namespace std;

template <typename T>

class Queue {

private:

queue<T> q;

public:

void enqueue(T value) {

q.push(value);

}

void dequeue() {

if (!q.empty()) {

q.pop();

} else {

cout << "Queue is empty!" << endl;

}

}

void display() {

if (q.empty()) {

cout << "Queue is empty!" << endl;

return;

}

queue<T> temp = q;

while (!temp.empty()) {

cout << temp.front() << " ";

temp.pop();

}

cout << endl;

}

};

int main() {

Queue<int> intQueue;

intQueue.enqueue(10);

intQueue.enqueue(20);

intQueue.enqueue(30);

cout << "Queue elements: ";

intQueue.display();

intQueue.dequeue();

cout << "Queue after dequeue: ";

intQueue.display();

return 0;

}

**83. Writing a Program that Uses Polymorphism to Create a Menu-Driven Application for Managing Different Types of Bank Accounts**

cpp

CopyEdit

#include <iostream>

#include <vector>

using namespace std;

class BankAccount {

public:

virtual void displayBalance() = 0;

virtual void deposit(double amount) = 0;

virtual void withdraw(double amount) = 0;

};

class SavingsAccount : public BankAccount {

private:

double balance;

public:

SavingsAccount(double initialBalance) : balance(initialBalance) {}

void displayBalance() override {

cout << "Savings Account Balance: $" << balance << endl;

}

void deposit(double amount) override {

balance += amount;

cout << "Deposited: $" << amount << endl;

}

void withdraw(double amount) override {

if (balance >= amount) {

balance -= amount;

cout << "Withdrew: $" << amount << endl;

} else {

cout << "Insufficient funds!" << endl;

}

}

};

class CurrentAccount : public BankAccount {

private:

double balance;

public:

CurrentAccount(double initialBalance) : balance(initialBalance) {}

void displayBalance() override {

cout << "Current Account Balance: $" << balance << endl;

}

void deposit(double amount) override {

balance += amount;

cout << "Deposited: $" << amount << endl;

}

void withdraw(double amount) override {

if (balance >= amount) {

balance -= amount;

cout << "Withdrew: $" << amount << endl;

} else {

cout << "Insufficient funds!" << endl;

}

}

};

int main() {

vector<BankAccount\*> accounts;

accounts.push\_back(new SavingsAccount(1000));

accounts.push\_back(new CurrentAccount(500));

int choice;

do {

cout << "1. Deposit\n2. Withdraw\n3. Display Balance\n4. Exit\nEnter your choice: ";

cin >> choice;

if (choice == 1) {

int accountType;

double amount;

cout << "Choose account type (1: Savings, 2: Current): ";

cin >> accountType;

cout << "Enter amount to deposit: ";

cin >> amount;

if (accountType == 1) {

accounts[0]->deposit(amount);

} else if (accountType == 2) {

accounts[1]->deposit(amount);

}

} else if (choice == 2) {

int accountType;

double amount;

cout << "Choose account type (1: Savings, 2: Current): ";

cin >> accountType;

cout << "Enter amount to withdraw: ";

cin >> amount;

if (accountType == 1) {

accounts[0]->withdraw(amount);

} else if (accountType == 2) {

accounts[1]->withdraw(amount);

}

} else if (choice == 3) {

int accountType;

cout << "Choose account type (1: Savings, 2: Current): ";

cin >> accountType;

if (accountType == 1) {

accounts[0]->displayBalance();

} else if (accountType == 2) {

accounts[1]->displayBalance();

}

}

} while (choice != 4);

// Cleanup

for (auto account : accounts) {

delete account;

}

return 0;

}

**84. Implementing a Program that Demonstrates the Use of Smart Pointers for Dynamic Memory Management**

cpp

CopyEdit

#include <iostream>

#include <memory>

using namespace std;

class Resource {

public:

Resource() {

cout << "Resource acquired." << endl;

}

~Resource() {

cout << "Resource released." << endl;

}

};

int main() {

// Using smart pointer to automatically manage memory

{

unique\_ptr<Resource> resPtr = make\_unique<Resource>();

cout << "Using resource inside scope." << endl;

} // Resource is automatically released here when the unique\_ptr goes out of scope

return 0;

}

**85. Creating a Program that Uses Exception Handling and Templates to Implement a Safe Array Class**

cpp

CopyEdit

#include <iostream>

#include <stdexcept>

using namespace std;

template <typename T>

class SafeArray {

private:

T\* arr;

int size;

public:

SafeArray(int s) : size(s) {

arr = new T[size];

}

~SafeArray() {

delete[] arr;

}

void set(int index, T value) {

if (index < 0 || index >= size) {

throw out\_of\_range("Index out of range!");

}

arr[index] = value;

}

T get(int index) {

if (index < 0 || index >= size) {

throw out\_of\_range("Index out of range!");

}

return arr[index];

}

};

int main() {

try {

SafeArray<int> arr(5);

arr.set(2, 10);

cout << "Element at index 2: " << arr.get(2) << endl;

arr.set(6, 15); // Will throw an exception

} catch (const out\_of\_range& e) {

cout << "Error: " << e.what() << endl;

}

return 0;

}

**86. Implementing a Program that Demonstrates the Use of Virtual Inheritance to Avoid the Diamond Problem in Multiple Inheritance**

cpp

CopyEdit

#include <iostream>

using namespace std;

class A {

public:

A() {

cout << "Class A constructor" << endl;

}

};

class B : virtual public A {

public:

B() {

cout << "Class B constructor" << endl;

}

};

class C : virtual public A {

public:

C() {

cout << "Class C constructor" << endl;

}

};

class D : public B, public C {

public:

D() {

cout << "Class D constructor" << endl;

}

};

int main() {

D obj; // Creates an object of class D

return 0;

}

In this case, B and C both inherit virtually from A. This ensures that the constructor of A is called only once, avoiding the "diamond problem" in multiple inheritance.

**87. Implementing a Class Polynomial with Member Functions to Add and Multiply Polynomials Using Operator Overloading**

cpp

CopyEdit

#include <iostream>

#include <vector>

using namespace std;

class Polynomial {

private:

vector<int> coeffs;

public:

Polynomial(vector<int> c) : coeffs(c) {}

// Operator overloading for addition

Polynomial operator + (const Polynomial& other) {

int size = max(coeffs.size(), other.coeffs.size());

vector<int> result(size, 0);

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

result[i] += coeffs[i];

}

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

result[i] += other.coeffs[i];

}

return Polynomial(result);

}

// Operator overloading for multiplication

Polynomial operator \* (const Polynomial& other) {

int size = coeffs.size() + other.coeffs.size() - 1;

vector<int> result(size, 0);

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

for (int j = 0; j < other.coeffs.size(); j++) {

result[i + j] += coeffs[i] \* other.coeffs[j];

}

}

return Polynomial(result);

}

void display() {

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

cout << coeffs[i] << "x^" << i;

if (i < coeffs.size() - 1) {

cout << " + ";

}

}

cout << endl;

}

};

int main() {

Polynomial p1({1, 2, 3}); // 1 + 2x + 3x^2

Polynomial p2({4, 5}); // 4 + 5x

Polynomial sum = p1 + p2;

Polynomial product = p1 \* p2;

cout << "Sum of polynomials: ";

sum.display();

cout << "Product of polynomials: ";

product.display();

return 0;

}

**88. Create a Program that Uses Function Pointers to Implement a Callback Mechanism**

cpp

CopyEdit

#include <iostream>

using namespace std;

// Define a callback function type

typedef void (\*Callback)(int);

void greet(int num) {

cout << "Hello, " << num << " people!" << endl;

}

void farewell(int num) {

cout << "Goodbye, " << num << " people!" << endl;

}

// Function that takes a callback

void executeCallback(Callback cb, int num) {

cb(num);

}

int main() {

executeCallback(greet, 5); // Call greet function

executeCallback(farewell, 3); // Call farewell function

return 0;

}

In this example, we have a Callback function pointer type and two functions (greet and farewell). The executeCallback function accepts a callback function and calls it with the given argument.

**89. Write a Program that Uses Class Templates and Exception Handling to Implement a Generic and Robust Data Structure**

cpp

CopyEdit

#include <iostream>

#include <stdexcept>

using namespace std;

template <typename T>

class SafeArray {

private:

T\* arr;

int size;

public:

SafeArray(int s) : size(s) {

arr = new T[size];

}

~SafeArray() {

delete[] arr;

}

void set(int index, T value) {

if (index < 0 || index >= size) {

throw out\_of\_range("Index out of range!");

}

arr[index] = value;

}

T get(int index) {

if (index < 0 || index >= size) {

throw out\_of\_range("Index out of range!");

}

return arr[index];

}

};

int main() {

try {

SafeArray<int> arr(5);

arr.set(2, 10);

cout << "Element at index 2: " << arr.get(2) << endl;

arr.set(6, 15); // Will throw an exception

} catch (const out\_of\_range& e) {

cout << "Error: " << e.what() << endl;

}

return 0;

}

This example uses class templates and exception handling to implement a safe array class, which throws exceptions if accessed out of bounds.

**90. Implementing a Program that Demonstrates the Use of Virtual Destructors in a Class Hierarchy**

cpp

CopyEdit

#include <iostream>

using namespace std;

class Base {

public:

virtual ~Base() { // Virtual destructor

cout << "Base destructor called" << endl;

}

};

class Derived : public Base {

public:

~Derived() override { // Overriding destructor

cout << "Derived destructor called" << endl;

}

};

int main() {

Base\* ptr = new Derived(); // Creating derived class object

delete ptr; // Proper destruction order (Derived then Base)

return 0;

}

In this example, we use virtual destructors to ensure proper cleanup when deleting a derived class object via a base class pointer. This is important to avoid undefined behavior due to incomplete destruction of derived class objects.

cpp

CopyEdit

#include <iostream>

using namespace std;

int main() {

int num1, num2;

cout << "Enter two integers: ";

cin >> num1 >> num2;

cout << "The sum is: " << num1 + num2 << endl;

return 0;

}

This program reads two integers from the user and calculates their sum using cin and cout.