. Create a program using namespaces to implement a calculator with addition and multiplication functions. Ensure the namespaces are nested, and handle division by zero using control structures.

#include <iostream>

using namespace std;

namespace calculator {

namespace arithmetic {

class Calculator {

public:

static double add(double a, double b) {

return a + b;

}

static double multiply(double a, double b) {

return a \* b;

}

static double divide(double a, double b) {

if (b == 0) {

throw runtime\_error("Division by zero!");

}

return a / b;

}

};

}

}

int main() {

double num1, num2;

char op;

cout << "Enter first number: "

cin >> num1;

cout << "Enter operator (+, \*, /): ";

cin >> op;

cout << "Enter second number: ";

cin >> num2;

try {

switch (op) {

case '+':

cout << "Result: " << calculator::arithmetic::Calculator::add(num1, num2) << endl;

break;

case '\*':

cout << "Result: " << calculator::arithmetic::Calculator::multiply(num1, num2) << endl;

break;

case '/':

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

break;

default:

cout << "Invalid operator!" << endl;

}

} catch (const exception& e) {

cerr << e.what() << endl;

}

return 0;

}

2. Simulate a traffic light system using an enum to represent states (Red, Yellow, Green). Use control structures to cycle three times through states and print instructions based on the current state.

#include <iostream>

#include <chrono>

#include <thread>

using namespace std;

namespace traffic {

enum class LightState {

Red,

Yellow,

Green

};

class TrafficLight {

private:

LightState state;

public:

TrafficLight() : state(LightState::Red) {}

void changeState() {

switch (state) {

case LightState::Red:

state = LightState::Green;

break;

case LightState::Green:

state = LightState::Yellow;

break;

case LightState::Yellow:

state = LightState::Red;

break;

}

}

void printInstructions() {

switch (state) {

case LightState::Red:

cout << "Red Light: STOP!" << endl;

break;

case LightState::Green:

cout << "Green Light: GO!" << endl;

break;

case LightState::Yellow:

cout << "Yellow Light: CAUTION! Prepare to stop." << endl;

break;

}

}

};

}

int main() {

traffic::TrafficLight light;

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

cout << "Cycle " << cycle + 1 << ":" << endl;

light.printInstructions();

this\_thread::sleep\_for(chrono::seconds(2)); // Red light for 2 seconds

light.changeState();

light.printInstructions();

this\_thread::sleep\_for(chrono::seconds(3)); // Green light for 3 seconds

light.changeState();

light.printInstructions();

this\_thread::sleep\_for(chrono::seconds(1)); // Yellow light for 1 second

light.changeState();

cout << endl;

}

return 0;

}

3. Write a program to transpose a 3x3 matrix stored in a constant array. Use control structures to swap elements and print the result.

#include <iostream>

using namespace std;

int main() {

const int matrix[3][3] = {{1,2,3}, {4,5,6}, {7,8,9}};

int transpose[3][3];

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

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

transpose[j][i] = matrix[i][j];

}

}

cout << "Transposed Matrix:" << endl;

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

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

cout << transpose[i][j] << " ";

}

cout << endl;

}

return 0;

}

4. Create a program to convert temperatures between Celsius and Fahrenheit. Use typecasting to handle floating-point precision and control structures to validate input

#include <iostream>

using namespace std;

int main() {

int choice;

double temp;

cout << "1. Celsius to Fahrenheit\n2. Fahrenheit to Celsius\nEnter choice: ";

cin >> choice;

if (choice == 1) {

cout << "Enter temperature in Celsius: ";

cin >> temp;

cout << "Fahrenheit: " << (float)(temp \* 9/5 + 32) << endl;

} else if (choice == 2) {

cout << "Enter temperature in Fahrenheit: ";

cin >> temp;

cout << "Celsius: " << (float)((temp - 32) \* 5/9) << endl;

} else {

cout << "Invalid choice!" << endl;

}

return 0;

}

5. Create a program with two namespaces containing functions with the same identifier (process). Use variables to call the correct function based on user input.

#include <iostream>

using namespace std;

namespace A {

void process() { cout << "Namespace A process()" << endl; }

}

namespace B {

void process() { cout << "Namespace B process()" << endl; }

}

int main() {

int choice;

cout << "Enter 1 for A::process or 2 for B::process: ";

cin >> choice;

if (choice == 1)

A::process();

else if (choice == 2)

B::process();

else

cout << "Invalid choice!" << endl;

return 0;

}

6. Generate the first N Fibonacci numbers, where N is a constant. Use control structures to compute and store the sequence in an array.

#include <iostream>

using namespace std;

#define N 10

int main() {

int fib[N] = {0, 1};

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

fib[i] = fib[i-1] + fib[i-2];

}

cout << "First " << N << " Fibonacci numbers: ";

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

cout << fib[i] << " ";

cout << endl;

return 0;

}

7. Create a function to calculate the total cost of event tickets with a default discount rate of 10%. Allow the user to specify a custom discount or use the default.

#include <iostream>

using namespace std;

float totalCost(float price, int qty, float discount = 10) {

float total = price \* qty;

return total - (total \* discount / 100);

}

int main() {

float price;

int qty;

float discount;

char choice;

cout << "Enter price and quantity: ";

cin >> price >> qty;

cout << "Do you want to enter custom discount? (y/n): ";

cin >> choice;

if (choice == 'y') {

cout << "Enter discount: ";

cin >> discount;

cout << "Total Cost: " << totalCost(price, qty, discount) << endl;

} else {

cout << "Total Cost (10% discount): " << totalCost(price, qty) << endl;

}

return 0;

}

8. Define macros to compute the square and cube of a number. Write a function to compare the results and determine which is larger, using the macros for calculations.

#include <iostream>

using namespace std;

#define SQUARE(x) ((x)\*(x))

#define CUBE(x) ((x)\*(x)(x))

void compare(int num) {

int sq = SQUARE(num);

int cb = CUBE(num);

cout << "Square: " << sq << ", Cube: " << cb << endl;

if (sq > cb)

cout << "Square is larger." << endl;

else if (cb > sq)

cout << "Cube is larger." << endl;

else

cout << "Both are equal." << endl;

}

int main() {

int num;

cout << "Enter a number: ";

cin >> num;

compare(num);

return 0;

}

9. Create a BankAccount class with multiple constructors to initialize an account with different details (account number only, or account number and balance). Include a function to display account details.

#include <iostream>

using namespace std;

class BankAccount {

int accNumber;

float balance;

public:

BankAccount(int acc) : accNumber(acc), balance(0.0f) {}

BankAccount(int acc, float bal) : accNumber(acc), balance(bal) {}

void display() {

cout << "Account: " << accNumber << ", Balance: " << balance << endl;

}

};

int main() {

BankAccount a1(12345);

BankAccount a2(54321, 5000);

a1.display();

a2.display();

return 0;

}

10. Create a multilevel inheritance chain: Vehicle → Car → ElectricCar. Each class adds specific attributes, and the final class displays all details.

#include <iostream>

using namespace std;

class Vehicle {

protected:

string brand;

public:

Vehicle(string b) : brand(b) {}

};

class Car : public Vehicle {

protected:

int seats;

public:

Car(string b, int s) : Vehicle(b), seats(s) {}

};

class ElectricCar : public Car {

int battery;

public:

ElectricCar(string b, int s, int bat) : Car(b, s), battery(bat) {}

void display() {

cout << "Brand: " << brand << ", Seats: " << seats << ", Battery: " << battery << " kWh" << endl;

}

};

int main() {

ElectricCar e("Tesla", 5, 100);

e.display();

return 0;

}

11. Create a base class Shape with a function to draw. Derive Circle and Rectangle classes using hierarchical inheritance, overloading the draw function to handle different parameters.

#include <iostream>

using namespace std;

class Shape {

public:

void draw() {

cout << "Drawing a generic shape." << endl;

}

};

class Circle : public Shape {

public:

void draw(float radius) {

cout << "Drawing a circle with radius " << radius << endl;

}

};

class Rectangle : public Shape {

public:

void draw(float length, float width) {

cout << "Drawing a rectangle of " << length << " x " << width << endl;

}

};

int main() {

Circle c;

Rectangle r;

c.draw(); // Base class draw

c.draw(5.0); // Circle-specific draw

r.draw(); // Base class draw

r.draw(4.0, 6.0); // Rectangle-specific draw

return 0;

}

---

12. Create a base class Book with attributes for title and author. Derive two classes, FictionBook and NonFictionBook, using hierarchical inheritance. Each derived class should have specific attributes (e.g., genre for fiction, subject for non-fiction). Implement overloaded functions to display book details in short and detailed formats. Use constructors to initialize objects.

#include <iostream>

using namespace std;

class Book {

protected:

string title, author;

public:

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

};

class FictionBook : public Book {

string genre;

public:

FictionBook(string t, string a, string g) : Book(t, a), genre(g) {}

void display(bool detailed = false) {

cout << "Fiction - Title: " << title;

if (detailed)

cout << ", Author: " << author << ", Genre: " << genre;

cout << endl;

}

};

class NonFictionBook : public Book {

string subject;

public:

NonFictionBook(string t, string a, string s) : Book(t, a), subject(s) {}

void display(bool detailed = false) {

cout << "Non-Fiction - Title: " << title;

if (detailed)

cout << ", Author: " << author << ", Subject: " << subject;

cout << endl;

}

};

int main() {

FictionBook fb("The Alchemist", "Paulo Coelho", "Adventure");

NonFictionBook nfb("Brief History of Time", "Stephen Hawking", "Physics");

fb.display();

fb.display(true);

nfb.display();

nfb.display(true);

return 0;

}

---

13.Create a base class Shape and derived classes Circle and Rectangle. Use virtual functions to calculate and display areas, demonstrating runtime polymorphism.

#include <iostream>

#include <cmath>

using namespace std;

class Shape {

public:

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

};

class Circle : public Shape {

float radius;

public:

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

void area() {

cout << "Circle area: " << 3.14 \* radius \* radius << endl;

}

};

class Rectangle : public Shape {

float length, width;

public:

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

void area() {

cout << "Rectangle area: " << length \* width << endl;

}

};

int main() {

Shape\* s1;

Circle c(5);

Rectangle r(4, 6);

s1 = &c;

s1->area();

s1 = &r;

s1->area();

return 0;

}

---

14. Write a program to reverse an integer array using pointer arithmetic. Validate the array size to ensure it’s positive.

#include <iostream>

using namespace std;

int main() {

int size;

cout << "Enter size of array: ";

cin >> size;

if (size <= 0) {

cout << "Invalid size!" << endl;

return 0;

}

int arr[size];

cout << "Enter " << size << " elements: ";

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

cin >> arr[i];

int\* start = arr;

int\* end = arr + size - 1;

while (start < end) {

int temp = \*start;

\*start = \*end;

\*end = temp;

start++;

end--;

}

cout << "Reversed array: ";

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

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

cout << endl;

return 0;

}

---

15. Create an Employee class with name and salary. Use pointers to objects and the this pointer to chain a method that increases salary.

#include <iostream>

using namespace std;

class Employee {

string name;

float salary;

public:

Employee(string n, float s) : name(n), salary(s) {}

Employee\* increaseSalary(float amount) {

salary += amount;

return this;

}

void display() {

cout << "Name: " << name << ", Salary: " << salary << endl;

}

};

int main() {

Employee e1("John", 5000);

Employee\* ptr = &e1;

ptr->increaseSalary(1000)->increaseSalary(500); // Chaining

ptr->display();

return 0;

}

---

16. Define an abstract class Vehicle with a pure virtual function describe. Derive Car and Bike classes to implement the function.

#include <iostream>

using namespace std;

class Vehicle {

public:

virtual void describe() = 0; // Pure virtual

};

class Car : public Vehicle {

public:

void describe() {

cout << "This is a car." << endl;

}

};

class Bike : public Vehicle {

public:

void describe() {

cout << "This is a bike." << endl;

}

};

int main() {

Car c;

Bike b;

Vehicle\* v;

v = &c;

v->describe();

v = &b;

v->describe();

return 0;

}

---

17. Write a program to divide two numbers, handling division by zero and invalid input exceptions using custom exception classes

#include <iostream>

using namespace std;

class DivisionByZero : public exception {

public:

const char\* what() const throw() {

return "Error: Division by zero!";

}

};

int main() {

int a, b;

cout << "Enter two numbers: ";

cin >> a >> b;

try {

if (b == 0)

throw DivisionByZero();

cout << "Result: " << a / b << endl;

} catch (DivisionByZero& e) {

cout << e.what() << endl;

}

return 0;

}

---

18. Create a template function to find the maximum of two values, supporting integers, doubles, and strings. Handle comparison for custom types.

#include <iostream>

#include <string>

using namespace std;

template <typename T>

T maximum(T a, T b) {

return (a > b) ? a : b;

}

int main() {

cout << "Max(3, 7): " << maximum(3,7) << endl;

cout << "Max(3.5, 2.5): " << maximum(3.5,2.5) << endl;

cout << "Max(\"apple\", \"banana\"): " << maximum(string("apple"), string("banana")) << endl;

return 0;

}

---

19. Create a base class Payment with a virtual process function. Derive CreditCard and Cash classes to handle different payment methods.

#include <iostream>

using namespace std;

class Payment {

public:

virtual void process() = 0;

};

class CreditCard : public Payment {

public:

void process() {

cout << "Processing credit card payment." << endl;

}

};

class Cash : public Payment {

public:

void process() {

cout << "Processing cash payment." << endl;

}

};

int main() {

CreditCard cc;

Cash cash;

Payment\* p;

p = &cc;

p->process();

p = &cash;

p->process();

return 0;

}

---

20. Create an abstract class Resource with a pure virtual use function and a virtual destructor. Derive FileResource and NetworkResource to demonstrate proper cleanup.

#include <iostream>

using namespace std;

class Resource {

public:

virtual void use() = 0;

virtual ~Resource() { cout << "Resource destroyed" << endl; }

};

class FileResource : public Resource {

public:

void use() {

cout << "Using file resource." << endl;

}

~FileResource() { cout << "FileResource destroyed" << endl; }

};

class NetworkResource : public Resource {

public:

void use() {

cout << "Using network resource." << endl;

}

~NetworkResource() { cout << "NetworkResource destroyed" << endl; }

};

int main() {

Resource\* res = new FileResource();

res->use();

delete res;

res = new NetworkResource();

res->use();

delete res;

return 0;

}

---

21. Print a number pattern where each row contains numbers from 1 to the row number, but only odd-numbered rows are printed in reverse. For example, for 5 rows:

1

2 1

3

4 3 2 1

5

#include <iostream>

using namespace std;

int main() {

int rows = 5;

for (int i = 1; i <= rows; i++) {

if (i % 2 == 0) {

for (int j = i; j >= 1; j--)

cout << j << " ";

} else {

for (int j = 1; j <= i; j++)

cout << j << " ";

}

cout << endl;

}

return 0;

}

22. Compute the sum of the series 1 - 2 + 3 - 4 + ... up to n terms using a loop. Ensure the sign alternates correctly.

#include <iostream>

using namespace std;

int main() {

int n, sum = 0;

cout << "Enter number of terms: ";

cin >> n;

for (int i = 1; i <= n; i++) {

if (i % 2 == 0)

sum -= i;

else

sum += i;

}

cout << "Sum of series: " << sum << endl;

return 0;

}