### Structure

We'll create the following files:

1. main.cpp: Entry point for the application.
2. basic\_operations.cpp: Implementation of basic arithmetic operations.
3. trigonometric\_operations.cpp: Implementation of trigonometric functions.
4. exponential\_operations.cpp: Implementation of exponential and logarithmic functions.
5. scientific\_calculator.h: Header file containing declarations.

### scientific\_calculator.h

This file will declare the functions used in the other modules.

#ifndef SCIENTIFIC\_CALCULATOR\_H

#define SCIENTIFIC\_CALCULATOR\_H

// Basic Operations

double add(double a, double b);

double subtract(double a, double b);

double multiply(double a, double b);

double divide(double a, double b);

// Trigonometric Operations

double sine(double angle);

double cosine(double angle);

double tangent(double angle);

// Exponential and Logarithmic Operations

double power(double base, double exponent);

double logarithm(double value);

double exponential(double value);

#endif // SCIENTIFIC\_CALCULATOR\_H

### basic\_operations.cpp

This file implements basic arithmetic operations.

#include "scientific\_calculator.h"

double add(double a, double b) {

return a + b;

}

double subtract(double a, double b) {

return a - b;

}

double multiply(double a, double b) {

return a \* b;

}

double divide(double a, double b) {

if (b == 0) {

throw std::invalid\_argument("Division by zero is undefined.");

}

return a / b;

}

### trigonometric\_operations.cpp

This file implements trigonometric operations.

#include <cmath>

#include "scientific\_calculator.h"

double sine(double angle) {

return sin(angle);

}

double cosine(double angle) {

return cos(angle);

}

double tangent(double angle) {

return tan(angle);

}

### exponential\_operations.cpp

This file implements exponential and logarithmic functions.

#include <cmath>

#include "scientific\_calculator.h"

double power(double base, double exponent) {

return pow(base, exponent);

}

double logarithm(double value) {

if (value <= 0) {

throw std::invalid\_argument("Logarithm undefined for non-positive values.");

}

return log(value);

}

double exponential(double value) {

return exp(value);

}

### main.cpp

This file is the entry point of the program, handling user interaction and invoking the appropriate functions.

#include <iostream>

#include "scientific\_calculator.h"

void showMenu() {

std::cout << "Scientific Calculator Menu:\n";

std::cout << "1. Addition\n";

std::cout << "2. Subtraction\n";

std::cout << "3. Multiplication\n";

std::cout << "4. Division\n";

std::cout << "5. Sine\n";

std::cout << "6. Cosine\n";

std::cout << "7. Tangent\n";

std::cout << "8. Power\n";

std::cout << "9. Logarithm\n";

std::cout << "10. Exponential\n";

std::cout << "11. Exit\n";

std::cout << "Enter your choice: ";

}

int main() {

int choice;

double num1, num2;

do {

showMenu();

std::cin >> choice;

switch (choice) {

case 1:

std::cout << "Enter two numbers: ";

std::cin >> num1 >> num2;

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

break;

case 2:

std::cout << "Enter two numbers: ";

std::cin >> num1 >> num2;

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

break;

case 3:

std::cout << "Enter two numbers: ";

std::cin >> num1 >> num2;

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

break;

case 4:

std::cout << "Enter two numbers: ";

std::cin >> num1 >> num2;

try {

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

} catch (const std::invalid\_argument& e) {

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

}

break;

case 5:

std::cout << "Enter an angle in radians: ";

std::cin >> num1;

std::cout << "Result: " << sine(num1) << std::endl;

break;

case 6:

std::cout << "Enter an angle in radians: ";

std::cin >> num1;

std::cout << "Result: " << cosine(num1) << std::endl;

break;

case 7:

std::cout << "Enter an angle in radians: ";

std::cin >> num1;

std::cout << "Result: " << tangent(num1) << std::endl;

break;

case 8:

std::cout << "Enter base and exponent: ";

std::cin >> num1 >> num2;

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

break;

case 9:

std::cout << "Enter a positive number: ";

std::cin >> num1;

try {

std::cout << "Result: " << logarithm(num1) << std::endl;

} catch (const std::invalid\_argument& e) {

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

}

break;

case 10:

std::cout << "Enter a number: ";

std::cin >> num1;

std::cout << "Result: " << exponential(num1) << std::endl;

break;

case 11:

std::cout << "Exiting...\n";

break;

default:

std::cout << "Invalid choice. Please try again.\n";

}

} while (choice != 11);

return 0;

}

### Explanation

1. **Modular Design**: Each set of related functions is in its own module (file), which makes the code easier to maintain and test.
2. **Exception Handling**: The program includes basic error handling, such as preventing division by zero and calculating the logarithm of non-positive numbers.
3. **Menu-Driven Interface**: The user interacts with the calculator through a simple menu system.

### How to Compile

You can compile the program using g++:

g++ main.cpp basic\_operations.cpp trigonometric\_operations.cpp exponential\_operations.cpp -o scientific\_calculator

### Testing for DevOps

For DevOps testing, you can write automated tests, package this code into a container, and run it on different environments to ensure consistent behavior. Tools like gtest for unit testing and Docker for containerization are commonly used in DevOps pipelines.