CS3201: OBJECT ORIENTED PROGRAMMING LABORATORY

Topic: Function templates, class templates, namespaces and exception handling

Date: April 22, 2025

Spot Questions 4 x 10 = 40

1. Design and implement a matrix calculator that operates within a namespace called MathLib and supports both real and complex number operations using templates. Inside this namespace, define a class template Matrix<T> that stores a 2D matrix of any numeric type, including int, double, and std::complex<double>. Implement a function template MatrixMultiply that performs matrix multiplication and returns the resulting matrix. To handle errors related to invalid matrix dimensions, create a custom exception class DimensionMismatchError inside a nested namespace MathLib::Errors, inheriting from std::exception. This exception should be thrown whenever two matrices are not compatible for multiplication. Ensure your implementation can handle complex numbers correctly by utilizing the <complex> library and allowing std::complex<T> as a valid type for the matrix elements. In the main() function, demonstrate multiplication of both integer and complex matrices, print the results in a readable format, and handle any thrown exceptions gracefully using try-catch blocks.
2. Within a CalculatorLib namespace, define a function template Power<T> that computes $base^{exponent}$ for numeric types and throws a NegativeExponentError if the exponent is negative and the base is non-floating-point. Implement a class template ScientificCalculator<T> that includes this function along with other operations like square root and logarithm. Place all error types in a CalculatorLib::Exceptions namespace.
3. Write a C++ program that simulates a type-safe expression evaluator within a namespace called SafeEval. The program should include a function template Evaluate<T>(T a, T b, char op) that performs arithmetic operations (+, -, *, /) on the inputs a and b. You should define three custom exception classes: InvalidOperatorError, DivisionByZeroError, and TypeMismatchError, all inheriting from std::exception, and place them in a nested namespace SafeEval::Errors. The Evaluate function should throw these exceptions under appropriate conditions — for example, if an unsupported operator is passed, if a division by zero is attempted (for non-floating-point types), or if a string type is passed where arithmetic is not meaningful. In your main() function, call Evaluate with various inputs to deliberately trigger each of these exceptions. Use multiple catch blocks to catch each specific error type and print

a custom error message. Conclude with a final catch(...) block that handles any unexpected exception types and displays a generic error message.

4. Design a templated Stack<T> class inside a Containers namespace that supports push, pop, top, and size operations. Implement custom exception handling in a nested namespace Containers::Errors, with an EmptyStackException that is thrown when attempting to pop or access the top of an empty stack. Write a function template that accepts a Stack<T> and prints all elements without modifying the original stack. Finally, demonstrate usage of this class with both int and string types in a main function, using try-catch blocks to gracefully handle exceptions.