Reviewer Name: Xuan Zhang

Reviewed Name:Xiyao Xu

Code coverage analysis:

Method Name	Code coverage	Proposed test(s) to include

Add::equals	0%	A test case is needed to check the equals method in the Add class. Two Add expressions, expr1 and expr2, can be created, and then it can be checked if they are equal.
		
		// Test case to verify the equality of two Add expressions
		// It checks if two structurally identical Add expressions are considered equal
		TEST_CASE("Verify Add Expression Equality") {
		  // Creating first Add expression
		  Add addExprOne(new Num(1), new Mult(new Num(2), new VarExpr("x")));
		
		  // Creating a second, identical Add expression
		  Add addExprTwo(new Num(1), new Mult(new Num(2), new VarExpr("x")));
		
		  // Using REQUIRE to test if both Add expressions are equal
		  REQUIRE(addExprOne.equals(&addExprTwo));
		}

Mult::interp	0%	A test case is required to test the interp method in the Mult class. A Mult expression can be created, and the interp method called to check its computational result.
		
		// Test case for validating the interpolation of a Multiplication expression
		// This specifically tests the result of multiplying two numbers
		TEST_CASE("Validate Multiplication Expression Interpolation") {
		  // Creating a Multiplication expression with two numbers
		  Mult multExpr(new Num(3), new Num(2));
		
		  // Checking if the interpolation of 3*2 equals 6
		  REQUIRE(multExpr.interp() == 6);
		}

Add::interp	0%	A test case is necessary to test the interp method in the

		Add class. In the test case, nested Add expressions can be used, and the interp method called to confirm the final computation result.
		
		// Test case to check the interpolation of nested Addition expressions
		// It verifies the correct calculation of nested Add operations
		TEST_CASE("Interpolation of Nested Addition Expressions") {
		  // Creating a nested Add expression (10+15) + (20+20)
		  Add nestedAdd(new Add(new Num(10), new Num(15)), new Add(new Num(20), new Num(20)));
		
		  // Testing if the nested Add expression correctly sums up to 65
		  REQUIRE(nestedAdd.interp() == 65);
		}

Thoughts / suggestions to improve the code or the tests:

Improvements for the Code (`expr.cpp`):

1. Memory Management:
In the current implementation, we are using raw pointers which could lead to memory leaks. Consider using smart pointers (like std::unique_ptr or std::shared_ptr) for better memory management and to avoid manual deletion of objects.

2. Optimizing subst Method:
The subst method could be optimized. For example, if the variable name doesn't match, there's no need to create a new expression; we can simply return the current object.

3. Code Redundancy:
There is some redundancy in the methods of Add, Mult, and other classes. we might consider creating a base abstract class for common functionality or using templates to reduce redundancy.

Improvements for the Tests:

1. Test Case Descriptions:
You can add more descriptive names or comments to your current test cases. This will enhance readability and make it easier for others (or your future self) to understand the purpose of each test. For example, "Complex Expression Evaluation" could provide more detailed information about which aspect of complex expression evaluation it is targeting.

2. Refactor Repeated Code:
There is a pattern of duplicated code across multiple test cases. You can refactor this by

creating utility functions or, if supported by your testing framework, setup/teardown methods. For instance, consider creating a function to compare two expressions for equality, which can be reused in multiple test cases.

Add rows when necessary.