**Reviewer Name**:

**Whitney Kenner**

**Reviewed Name**:

**Mack Tawa**

**Code coverage analysis**:

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Code coverage** | **Proposed test(s) to include** |
| Num::equals(Expr \*e) | return false; line coverage  branch coverage | Testing num expression equality where they are not equal  CHECK ( new num(2)->equals(new num(4)) == false);  CHECK (new num(1)->equals(new add(2, 4)) == false |
| Num::has\_variable() | return false;  line coverage | Testing if a num has a variable  CHECK(new num(2)->hasVariable() == **false**); |
| Var::has\_variable() | return true;  line coverage | Testing if a vraible has a variable  CHECK(new var(“W”)->hasVariable() == **true**); |
| Var::equals(Expr \*e) | return false;  line coverage  branch coverage | Testing variable equality where the result is false  CHECK(new var(“x”)->equals(new var(“c”) == false);  CHECK(new var(“c”)->equals(new num(2) == fasle); |
| Var::interp() | throw std::runtime\_error( "no value for variable" );  line coverage | Testing that trying to interpret a variable throws an error  CHECK\_THROWS\_WITH( (**new** Variable("x"))->interp(), "Invalid call of interp- cannot interpet variable" ); |
| Add::has\_variable() | return (this->lhs->has\_variable() || this->rhs->has\_variable());  line and branch coverage | Testing that an add expression has a variable. This should be tested for both the left hand and right hand side with and without variables (all 4 branches)  CHECK(**new** Add(Num\* num2 = **new** Num(2), n Num\* num2 = **new** Num(2)) ->hasVariable() == **false**);  CHECK(**new** Add(Num\* num2 = **new** var(“x”), n Num\* num2 = **new** Num(2)) ->hasVariable() == true);  CHECK(**new** Add(Num\* num2 = **new** Num(2), n Num\* num2 = **new** var(“b”)) ->hasVariable() == true);  CHECK(**new** Add(Num\* num2 = **new** var(“j”), n Num\* num2 = **new** var(“b”)) ->hasVariable() == true); |
| Mult::has\_variable() | return (this->lhs->has\_variable() || this->rhs->has\_variable());  line and branch coverage | Testing that an mult expression has a variable. This should be tested for both the left hand and right hand side with and without variables (all 4 branches)  CHECK(**new** Mult (Num\* num2 = **new** Num(2), n Num\* num2 = **new** Num(2)) ->hasVariable() == **false**);  CHECK(new Mult((Num\* num2 = **new** var(“x”), n Num\* num2 = **new** Num(2)) ->hasVariable() == true);  CHECK(new Mult(Num\* num2 = **new** Num(2), n Num\* num2 = **new** var(“b”)) ->hasVariable() == true);  CHECK( new Mult (Num\* num2 = **new** var(“j”), n Num\* num2 = **new** var(“b”)) ->hasVariable() == true); |
| Mult::equals(Expr \*e) | return false;  line coverage  branch coverage | Test that a multiplication equality returns false. Test this for both sides of the mult expr and inequality of comparison  CHECK(mult2->equals(num6) == **false**);  CHECK(num6->equals(mult2) == false);  Num\* num1 = **new** Num(2);  Num\* num2 = **new** Num(2);  Num\* num3 = **new** Num(3);  Mult\* mult2 = **new** Mult(num2, num3);  Mult\* mult3 = **new** Mult(num3, num1);  CHECK(mult3->equals(mult2) == **false**); |
|  |  |  |

**Thoughts / suggestions to improve the code or the tests**:

|  |
| --- |
| Make sure to test false return statements for each method |
| For equality methods be sure to check all branching options |
| Make sure to check invalid comparisons |

Add rows when necessary.