Alexis Delgado

Final coding project

A compiler typically translates the source code for another program language into executable code. Our interpreter doesn’t have to quite to do that, since the source from which we’re reading from isn’t a high-level programming language like Java. The sequence of machine instruction is nothing more than multiplying, dividing, subtracting and adding from whole values from a single type of primitive. During our lexical analysis, we’ve defined a statement as valid if it has the correct number of operands for the type of operation it’s performing (unary; binary) and whether the operands are numeric.

Our parse tree forms a data structure that represents our variable, type declaration, loops, expression and our case operation type. During a programs execution, a parse tree is typically constructed from into a phrase that is grammatically correct within the context of a program. Our parse tree is very simple, as it has relatively few constituents that build our tree. Thankfully, we only worked with numerical values and disregarded string literals.

Finally is our optimization, as our program evaluate constant expressions, it only optimizes away blank spaces and unreachable code (divide by zero error). Most compilers attempt to reduce some of attributes of a program either by creating a semantically equivalent program that isn’t as costly as its predecessor and takes less time to execute. Thankfully, we didn’t have to create any form of optimization algorithms.