Sami Blevens

Project Specification

Constant variables and function arguments

Implement the addition of constant variables and function arguments. This will need to be done by altering all aspects of the MyPL code generated for the homework assignments.

1. Update TokenType.java to include the new token type CONST which will be for the word "constant". Update the nextToken function within Lexer.java to handle the new CONST token. An example of this TokenType being added is:
   new Token(TokenType.CONST, "const", line, column).

2. Update the grammar language to include the optional constant type at the beginning of the variable declaration. Below is the updated grammar language, and an example of this grammar in MyPL:
   <vdecl_stmt> ::= (CONST | E) VAR (<dtype> | E) ID ASSIGN <expr>
   CONST var x = 43

3. Update the ASTParser.java and the VarDeclStmt Class (and subsequently can update the Parse.java). Add a boolean const variable to the VarDeclStmt Class:
   public Boolean const = false;
   Update the vdecl_stmt() method to handle the optional const token type and fill out the AST.

4. Create a Tuple class in order to utilize tuples.

5. Update SymbolTable.java and TypeInfo.java to use tuples to include a variable for the constant type – because variables are being stored here, and it is important to know both what type the variable is and if it is constant.
   List<Map<String,tuple<String,Boolean>>> environments = new ArrayList<>();

6. Update StaticChecker.java to check every single variable access whether the variable is constant and if it is, make sure the variable is not being changed.

7. Deal with function arguments. You have to decide the best way to accomplish this. One way is to require the function declaration to show CONST in the parameter declaration if constants are allowed. Then, in callExpr, when making sure that the arguments are the same type, you can also check that if the parameter is constant, then the argument is constant. This however does not work well if you want to be able to send in both constant and non-constant variables – that it doesn't have to be declared in the function declaration.

One full example of how the construct will work:

A variable declaration of

CONST var x = 43

will first be tokenized by the Lexer as:

(CONST, “const”), (VAR, “var”), (ID, “x”), (ASSIGN, “=”), (INT_VAL, “43”)

Then, within the vdecl_stmt() method in the Parser, will fill in the VarDeclStmt node as:

const = true; typeName = null; varName = x;

expr = Expr with firstTerm as simple term 43

Within StaticChecker, when adding the variables to the symbolTable and infoTable, add as:

symbolTable.add(“x”,new Tuple<”int”,true>);

Then, for AssignStmt, make sure that the lhs is not constant. If it is, error.

For function parameters, when adding them to the symbolTable, have to check if

Initial test cases that will be used for testing:

1. Simple type declaration.

```
type Node {
   CONST var x = 0
}
```

2. Simple function declaration.

```
Fun void f(){
  CONST var x = 0
}
```

3. Simple function call

```
fun void f(int x) {
  print(x)
}
fun void main(){
  CONST var y = 0
  f(y)
```

```
}
```

4. Function call with changes to parameter variable. – should fail

```
fun void f(int x) {
  x = x +1
}
fun void main() {
  CONST var y = 0
  f(y)
}
```