Static Semantics

Total 100 points.

/accounts/classes/janikowc/submitProject/submit\_cs4280\_P3 *SubmitFileOrDirectory*

Invocation:  
> statSem [*file*]  
with *file* as before with implicit extension as before**in P2  
Wrong invocations will not be graded.**

Graded 90% execution 10% structure/standards.

**Stat Semantics Definition**

* The only static semantics we impose that can be processed by the compiler (static) are proper use of variables.
* **Variables**
  + Variables have to be defined before used first time (after satisfying syntax)
    - **data ID = Integer** defines the variable named **ID**. **Integer** is disregarded for now
    - **ID** showing up in any statement is the variable use
  + Variable name can only be defined once in a scope. The only scope is global
* **Global** option for variables
  + There is only one scope, the global scope, regardless of where variable is defined

Do not display the tree any more (**points can be deducted**).

# **P3 Suggestions**

Modify the main function so that after calling parser and receiving the tree, the main will call the static semantic function on the tree. It is just a single tree traversal.

# **P3 Global Support**

### Software support

* Use any container (called ST here)  for names such as array, list, etc. with the following interface. It shows String as the parameter, which is the ID token instance, but it could include line number or the entire token for more detailed error reporting. This contained will process identifier tokens only.
  + **insert(String)** - insert the string if not already there or error if already there (you may return fail indication or issue detailed error here and exit)
  + **Bool verify(String)** - return true if the string is already in the container variable and false otherwise (I suggest you return false indicator rather than issue detailed error here with exit but either way could possibly work if you assume that no one checks verify() unless to process variable use)

### Static semantics

* Instantiate ST as STV for variables
* Traverse the tree and perform the following (looks like preorder traversal) based on the subtree you are visiting
  + If visiting <vars> and you find identifier token (variable definition) then call **STV.insert(String)** on the token - this is variable definition
  + Otherwise (you are not in <vars>)  if you find identifier token call **STV.verify(String)** - this is variable use

# **P3 Test Files - Global good**

Create syntactically valid files, best is to use those test programs from P2 as starting point. Then introduce variables at various places and use variables. All satisfying the rules: no mult variables with same name and  a variable used must be defined prior.

begin  
 out 1 .  
end  
  
data x = 1 .  
data y = 2 .  
begin  
 out 1 .  
end  
  
data x = 1 .  
data y = 2 .  
begin  
 data z = 3 .  
 out 1 .  
end  
  
data x = 1 .  
data y = 2 .  
begin  
 data z = 3 .  
 begin   
 out 1 .  
 end   
 out z .  
end  
  
data x = 1 .  
data y = 2 .  
begin  
 in x .  
 out x + 2 / ( 2 - y ) .  
end  
  
data x = 1 .  
begin  
 in x .  
 iffy [ x < 0 ] then  
 out x . .  
end  
  
data x = 1 .  
begin  
in x .  
iffy [ x < 0 ] then  
 begin  
 data y = 2 .  
 x = y .  
 out x .  
 end .  
end  
  
data x = 1 .  
begin  
in x .  
loop [ x < 0 ]  
 out x . .  
end  
  
data x = 1 .  
begin  
in x .  
loop [ x < 0 ]  
 begin  
 data y = 2 .  
 x = y .  
 out x .  
 end .  
out y .  
end  
  
data x = 1 .  
begin  
 in x .  
 iffy [ x < 0 ] then  
 iffy [ x < > 0 ] then   
 out x . . .  
end  
  
data x = 1 .  
begin  
data y = 2 .  
in x .  
loop [ x + 2 / 3 < 0 - \* 3 ]  
 iffy [ x < > y ] then   
 out x . . .  
end

# **P3 Test Files - Global error**

Take one good program and insert one error at a time. There are two kinds of errors: multiply defined variable, and variable used before definition.

begin  
 out x .  
end  
  
data x = 1 .  
data y = 2 .  
begin  
 data y = 3 .  
 out 1 .  
end  
  
data x = 1 .  
data y = 2 .  
begin  
 data z = 3 .  
 out w .  
end  
  
data x = 1 .  
data y = 2 .  
begin  
 in x .  
 out x + 2 / ( 2 - w ) .  
end  
  
data x = 1 .  
begin  
in x .  
iffy [ x < 0 ] then  
 begin  
 data x = 2 .  
 x = 1 .  
 out x .  
 end .  
end  
  
data x = 1 .  
begin  
data y = 2 .  
in x .  
loop [ x + 2 / 3 < 0 - \* 3 ]  
 iffy [ x < > w ] then   
 out x . . .  
end