Stat Semantics Definition

- The only static semantics we impose that can be processed by the compiler (static) are proper definition and use of variables.
- Variable
    - A variable has to be defined anywhere in the program to be used (must satisfy syntax)
        - variable definition
            - any identifier 'id' listed in any <varList> is definition
    - 
        - 
        - variable use - identifier showing up in any statement
    - Variable name can only be defined once in a scope but can be reused in another scope.

Global option for all variables (100 total)

- There is only one scope, the global scope, regardless of where a variable is defined.

**Do not display the tree any more in P3.**

# P3 Suggestions

Modify the main function from P2 so that after calling the parser and receiving the tree, the main will not call any more the P2 tester but will call the static semantic function on the tree. It is just a tree traversal. You may need to perform two traversals - do one call from main to staticSemantics() function and there perform all needed traversals. Place the new functions in a separate file.

# P3 Global Support

## Software support

- Use any container (called **ST** here) for strings (identifiers) such as array, list, etc. with the following interface. It shows a 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 container will process identifier tokens only. Provide the following API
    - **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 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 to collect all definitions using **insert()** into STV.
- Traverse again using STV but skipping over <vars> subtrees.
    - if you find identifier token in any node, this is variable use, call **STV.verify(string)** on the string token

If there are no errors after a complete traversal, static semantics are good (global option). If there is an error, you report and exit.

# 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.

#p3gg1

```
let aa = 1
ab = 2
ac = 3 .
main
  print 1 .
end
```

#p3gg2

```
let aa = 1
ab = 2
ac = 3 .
main
  print aa .
  start
    let ad = 4
    ae = 5 .
    print ab .
    start
      let af = 6 .
      print 2 .
    stop
```

```
  stop
end
```

#p3gg3

```
let aa = 1 .
main
  print ad .
  start
    let ad = 4
    ae = 5 .
    print af .
    start
      let af = 6 .
      aa ~ ad + 1 .
      print aa .
    stop
  stop
end
```

# P3 Test Files - Global bad

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.

#p3gb1 mult definition

```
let aa = 1
ab = 2
aa = 3 .
main
  print 1 .
end
```

#p3gb2 mult definitions

```
let aa = 1
ab = 2
ac = 3 .
main
  print aa .
  start
    let ad = 4
    ae = 5 .
    print ab .
    start
```

```
      let ab = 6 .
      print 2 .
    stop
  stop
end
```

#p3gb3 undefined

```
let aa = 1 .
main
  print ad .
  start
    let ad = 4
    ae = 5 .
    print af .
    start
      let af = 6 .
      af ~ ad + ag .
      print aa .
    stop
  stop
end
```