Project P3: Static Semantics
Due April 18, 2022 at 11:59 pm

Total 50 points.

- Check static semantics assuming global scope.
- Use a symbol table to track variables and check semantics.
- Invocation:
  > statSem [*file*]

  Failure to compile or wrong invocations will not be graded.
- ***Program must compile and run on clark.rnet.missouri.edu***

- Graded 90% execution 10% structure/standards.

Static Semantics Definition

- The only static semantics we impose that can be processed by the compiler (static) are proper use of variables.

- **Variables**
  - Variables are defined using '**`Name Indentifier`**' and '**`Spot Indentifier`**'.
  - Variables have to be defined before used first time (must satisfy syntax too)
  - Variable name can only be defined once

- Modify the main function so that after calling parser and receiving the tree, main will call the static semantic function on the tree.
- If an error is encountered, state the type of error (using an undefined variable or redefining a variable) and the variable name
  - You may terminate when you find the first error.
- If there are no errors, output the symbol table.

Suggestions

Software support

- Use any container for storing identifiers such as array, list, etc. with the following interface. Below we show '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.

  - 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 (suggest you return false indicator rather than issue detailed error with exit, but either way could possibly work if you assume that no one checks verify() unless to process variable use)

Static semantics

- Instantiate container (referred to as 'CONTNR' below)
- Traverse the tree and perform the following based on the subtree you are visiting
    - If visiting <X> and you find identifier token then call CONTNR.insert(String) // this is variable definition
    - Otherwise (you are not in <X>) if you find any identifier token call CONTNR.verify(String)

Optional Extra Credit (5 points):

- Optionally include a test for identifiers to be introduced in alphabetical order
- Assume that numbers come before letters in the sorted order
- Examples of orders of identifier declarations that are allowed:
    - bc2, hc3, hc33, hc3c, hcb, zyx
    - a, a2a, a3, aa, aa5, ab3, xy
- Examples of orders that are not allowed:
    - cd, c2
    - d3, d2

Test Files

Good file example:

```
Name prog1
Spot prog2
Place
Name id1
Home
Show prog1
```

Bad file example:

```
Name prog1
Spot prog2
Place
Name id1
Home
Show prog3
```

Another bad file example:

```
Name prog1
Spot prog2
```

```
Place
Name prog1
Home
Show prog1
```