CSE 340 Principles of Programming Languages
Fall 2014

Programming Assignment 3
Due on November 6, 2014 by 11:59 PM

Abstract

Create a semantic analyzer for the programming language described in class. Incorporating it with the lexical and syntax analyzer (developed in assignments #1 and #2) as a cohesive project.

INSTRUCTIONS

1. Assure that your Lexer (assignment #1) and Parser (assignment #2) work properly.

2. Download and set up the source code published on Blackboard. You will need to include a Lexer and a Parser to that code. Review and understand the code, mainly the following

   In the `SemanticAnalyzer.java` file:

   a) The declaration of the symbol table as a HashTable.

   b) The declaration of the stack that you are going to use to calculate types.

   c) In the `Gui.java` file, a new tab has been included to show the symbol table
   A new method `writeSymbolTable` allow you to print the symbol table in the new tab. You will need to call this method in your parser.java (in the last line of the method `run()`.The method `run()` will looks like this:

```
public static DefaultMutableTreeNode run(Vector<Token> t, Gui g) {
    gui = g;
    tokens = t;
    currentToken = 0;
    root = new DefaultMutableTreeNode("program");
    rule_program(root);
    gui.writeSymbolTable(SemanticAnalizer.getSymbolTable());
    return root;
}
```

3. In the class SemanticAnalyzer create cube of types. Fill it with the information shown in the tables in the appendix A at the end of this document.

4. You are required to complete in this class the definition of: (a) a function that reviews declaration and unicity of variables; (b) a function that reviews type matching in assignation; (c) a function that reviews Boolean type for “if” and “while” conditions; (d) functions, that uses the Cube and the Stack to calculate the resultant type for binary operators.

5. Inside your Parser file, add to each of the methods that represent your rules (program, variable, assignment, etc.) the needed code to: (a) store a variable in the Symbol Table; and (b) review for semantic errors. Lectures 17 to 19 provide a description of the code to be added.

6. The semantic analyzer must report the errors and the number of the line (obtained from your current list of Tokens) in which the errors occur. Your semantic analyzer should be able to recognize the following type of errors:

   a) All variables are declared and have a unique name.
   b) Types of variables match the values assigned to them.
   c) Conditions. The conditons have a boolean value.

   The error method provided in the source code published on Blackboard includes a method error that should be call to report the errors. It works similarly to how the error method in the parser class works.

Read and actively participate in the discussion board. Take advantage of the discussion content to improve your implementation.

Create a zip file, using the following naming convention: **Firstname_Lastname_P3.zip**. This file should contain your source code for:

SemanticAnalyzer.java

And should include a compiled version of your application:

CSE340.jar.

GRADING

The assignment will be graded in a scale of 0-100 considering the following:

- Convention followed for java implementation: one class SemanticAnalizer with static methods for each validation as described before.
- Symbol table implemented an working correctly (all variables included)
- Definition and uniqueness of variables is validated
- Type matching validation is working
- Conditions in IF and WHILE as boolean values is working

Be aware that:

- No credit will be given to programs that do not compile.

- No credit will be given if the project does not follow the format described above (input, output, number and structure of classes, attributes, and methods).

If you have any question, contact us (me or the TAs) to clarify them. And remember, actively participate in the discussion board.

Appendix A.

This are the semantic rules to be coded in your cube of types:

OP → {- (binary), *, /}

OP	int	float	char	string	boolean	void	error
int	int	float	error	error	error	error	error
float	float	float	error	error	error	error	error
char	error	error	error	error	error	error	error
string	error	error	error	error	error	error	error
boolean	error	error	error	error	error	error	error
void	error	error	error	error	error	error	error
error	error	error	error	error	error	error	error

OP → {+}

+	int	float	char	string	boolean	void	error
int	int	float	error	error	error	error	error
float	float	float	error	error	error	error	error
char	error	error	error	error	error	error	error
string	error	error	error	string	error	error	error
boolean	error	error	error	error	error	error	error
void	error	error	error	error	error	error	error
error	error	error	error	error	error	error	error

OP → {- (unary)}

-	int	float	char	string	boolean	void	error
	int	float	error	error	error	error	error

OP → {>, <}

OP	int	float	char	string	boolean	void	error
int	boolean	boolean	error	error	error	error	error
float	boolean	boolean	error	error	error	error	error
char	error	error	error	error	error	error	error
string	error	error	error	error	error	error	error
boolean	error	error	error	error	error	error	error
void	error	error	error	error	error	error	error
error	error	error	error	error	error	error	error

OP → {!=, ==}

OP	int	float	char	string	boolean	void	error
int	boolean	boolean	error	error	error	error	error
float	boolean	boolean	error	error	error	error	error
char	error	error	boolean	error	error	error	error
string	error	error	error	boolean	error	error	error
boolean	error	error	error	error	boolean	error	error
void	error	error	error	error	error	error	error
error	error	error	error	error	error	error	error

OP → {&, !}

OP	int	float	char	string	boolean	void	error
int	error	error	error	error	error	error	error
float	error	error	error	error	error	error	error
char	error	error	error	error	error	error	error
string	error	error	error	error	error	error	error
boolean	error	error	error	error	boolean	error	error
void	error	error	error	error	error	error	error
error	error	error	error	error	error	error	error

OP → {!}

!	int	float	char	string	boolean	void	error
	error	error	error	error	boolean	error	error