Clayton Green (230089109)

kgreen1@unbc.ca

October 31, 2013

CPSC 425 - Compiler Project Phase IV

Introduction

Included is an incomplete semantic analyzer. Declarations are correctly added to the symbol table in the init traversal and the full traversal phases. Missing is linking variables to their declarations and type checking, along with other small checks.

Participation

Clayton Green - All

Project Status

Scanner - Completed
Parser Basic - Complete
Parser Full - Not Complete
+ Add error recovery (synch)
Semantic Analyzer - Not Complete
+ Finish scope analysis (full traversal)
+ Add type checking

TODO
+ Add functionality for --out/-o
+ Use makefile instead of shell script
+ Update previous documents
+ Comment source files
+ Fix command line arguments (-q should be default yet it's required)

Architecture and Design

The administrator runs the semantic analyzer. First the AST, root, is built for the source program using the parser. If this phase completed without error, then another AST, new root, containing the built in functions (readint, writeint, readbool, writebool) is created. The new root AST

appends the root AST to itself. This accomplishes adding the built in functions to the original AST. Finally the semantic analyzer executes init traversal, full traversal then type analysis. All of the AST traversals are implemented using the visitor pattern.

The init visitor starts from the root of the AST and only visits the global declarations of functions and variables. The full visitor skips the global declarations and traverses into the global functions adding identifiers to the symbol table.

Implementation

Building and Use

Extract:

$ tar zxf green-parser-basic.tar.gz -- Compressed tar
$ tar xf green-parser-basic.tar -- Standard tar

This extracts the project into a directory named green-parser-basic.

Build:

$ cd green-parser-basic
$ ./build.sh -- make sure build.sh can be ran as an executable

This will build the compiler and put the executable in the folder
$ */green-parser-basic/executables/Compiler*

Usage:

$ ./Compiler {-l | -p | -s | -t | -c} {-q | -v} -I <filepath> { -I <filepath> } [-o]*

Option Listing:

*-h	--help*	*Display usage info and exits*
*-v	--version*	*Display version and exit*
-l	--lex	Process up to the Lexer phase
-p	--parse	Process up to the Parser phase
-s	--sem	Process up to the Semantic Analyser phase
-t	--tup	Process up to the Tuple phase
-c	--compile	Process all phases and compile (default)
-q	--quite	Only display error messages (default)
-v	--verbose	Display all trace messages
-I	--input	File to be compiled
-e	--err	Error file (defaults to STDOUT)
-o	--out	Output file (defaults to STDOUT)

Examples:

$ ./Compiler -s -v -I ./../Tests/input/parser/correct.cs13
-I ./../Tests/input/parser/incorrect.cs13 -e ./../Tests/results/parser/out.txt

This will run the files *correct.cs13* and *incorrect.cs13* through the compiler up to the semantic analyzer phase (*-s*) with trace messages (-v). Then output the trace messages and error messages to *out.txt*. (Note: this is when running Compiler from the executables directory).

Code

Semantic Analyzer Class

You can create a Semantic Analyzer by

SemanticAnalyzer sem(ASTroot, filename, administrator)

where ASTroot is the builtin function AST with the source code AST appended to it, filename is the name of the current source file, and administrator is a pointer to the calling administrator for access to the messenger. Semantic analysis can be executed in phases and is started by calling sem.InitTraversal();, sem.FullTraversal();, and sem.ScopeAnalyze(). The semantic analyzer also has the capability of printing the symbol table for debugging purposes.

ASTNodeInitVisitor Class

InitVisitor is run by calling

```
ASTNode *node;
node->Accept(new ASTNodeInitVisitor(&symbol_table, filename, administrator);
```

where the Semantic Analyzer's symbol table, filename, and pointer to calling administrator are all passed as arguments. The init visitor traverses the first level of the AST only adding global declarations to the symbol table.

ASTNodeFullVisitor Class

FullVisitor is run by calling

```
ASTNode *node;
node->Accept(new ASTNodeFullVisitor(&symbol_table, filename, administrator);
```

where the Semantic Analyzer's symbol table, filename, and pointer to calling administrator are all passed as arguments. The full visitor traverses the tree into function declarations and links variables to their declarations.

SymbolTable

The symbol table is a struct contained in the semantic analyzer and contains the access table and the identifier table.

Tests and Observations

Parse.cs13 is a skeleton program that only has declarations (from document) and is correct and properly builds the symbol table. Parse2.cs13 incorrectly defines readint twice and neglects int main(void) and both errors are reported.