CA448 - Compiler Construction

Assignment 2: A Top-Down parser with semantic analysis for the simpL language.

Ian Duffy - 13356066

Introduction

This document describes my approach towards creating a top-down parser with semantic analysis for the simpL language. Throughout this document I describe how I approached the task and why I did things in certain ways.

The submitted code fulfils all the requirements of the assignment that is, it generates an AST and symbol table along with preforming the following semantic checks:

- Is every identifier declared within scope before it is used?
- Is no identifier declared more than once in the same scope?
- Is the left-hand side of an assignment a variable or the correct type?
- Are the arguments of an arithmetic operator the interger or real, or integer or real constants?
- Are the arguments of a boolean operator boolean variables or boolean constants?
- Is there a function for every invoked identifier?
- Does every function call have the correct number of arguments?
- Is every variable both written to and read from?
- Is every function called?

While I wouldn't say the AST I have generated is fully comprehensive for the simpL language it has enough to meet all the semantic checks of this assignment.

My semantic check is done by walking the AST using a visitor. The symbol table is generated during the visiting.

My symbol table is a HashMap containing key String and value HashMap. The key String represents scope. The HashMap represents a scoped symbol table, this is made up of key identifier, value Symbol table child. Unique scopes are created for global (anthing not in main or a function), every function and main.

A symbol table child is made up of:

- Token identifier: The identifier of this variable/function
- DataType Kind: What kind is this? variable? const? function?

- String scope: What scope is this present in?
- Token type: What type is this identifier? String? Void? Boolean? Int? Real?
- Map data: Extra data that may be associated with this variable e.g. value, parameters, isWrittenTo, etc.

Lets discuss some of its nodes and why they exist:

Program

This represents the root node. Effectively it only exists as a container for all other nodes.

It does the following:

- Creates the hashmap for global scope.
- visits all child nodes.
- Prints out final semantic checks i.e. Functions that were called/not-called, variables that were written to/read from.
- Prints the symbol table.

VarDecl

This node represents the declaration of variables.

VarDecl has two children nodes one is IdentList the other is Type. Visiting IdentList returns a list of identifiers, I loop through this list and create a new entry in the symbol table for the current scope. Should an entry already be found in the symbol table for the current scope an error is printed.

ConstDecl

This node represents the declaration of constant variables.

A ConstDecl has many children nodes. Lets look at them as a set of three, identifier, type, expression. I loop through all children nodes as sets of three and create a new entry in the symbol table for the current scope. Should an entry already be found in the symbol table for the current scope an error is printed.

FunctionDecl

This node represents the declaration of a function.

a FunctionDecl is made up of three nodes, type, identifier and param list.

In the global scope symbol table I create a new entry for the function identifier. Should an entry already be found in the global scope symbol table I print an error. I also add the function identifier to a LinkedList called functionDeclerations. This list is used on parsing finish to check if all declared functions were called.

I switch the scope to function-scope-<counter>

For each of the params in the param list I create a new entry in the functions scope symbol table.

FunctionBody

This node represents the body of a function. I just visit all children. When it ends I reset the scope back to whatever it was before changing. (Always global for the simpL language)

ParamList

This node represents a param list which is passed to a function.

A param list can be empty, contain one entry of type and identifier or multiple entries seperated by comma of type and identifier.

I loop through all these child nodes and place them into a list.

This list is returned to whoever visits it.

Type

Just returns node.jjtGetValue();

Main

changes the scope to main. Visits all child nodes.

Assignment

This represents the assignment of a variable.

An assignment is made up of an identifier and an expression.

An expression can be many things, an identifier, a number, a true/false, a AddExpr, SubExpr, MultExpr, DivExpr, ModExpr, etc.

This is represented as a list…. Lets explain by examples.

x := y; identifier = x, value = [y]

x := y + 1 + 1; identifier = x, value = [y, [1, 1]]

On an assignment I add written as data to the STC in the symbol table.

Type checking occurs at this stage too. This is done via a recursive function.

The function takes the STC of the variable and the List of assignment values. It loops through

the list, if the item is a list it calls the check type function with itself, if its a token I check the type.

If the token is an identifier I loop up the scoped symbol table to get the identifier's decleration. If nothing is found in the scoped symbol table I look in the global symbol table.

If something is found in either I compare the identifer's decleration type with the type of the variable decleration. If they don't match I print an error message.

if the token is a non-identifier(number, etc.) I just compare the token kind directly with the variable decleration. If they don't match I print an error message.

Function Call

This represents the call to a function.

A functionCall is made up of an identifier and arguments.

I look the identifier up global symbol table, get the function decleration. if the function is not found I print an error. I add this function call to a LinkedList calledFunctions. This list is used on parsing finish to check if all declared functions were called.

I then look at the data for the function decleration, if it contains a paramList I get the paramList size and compare it with the amount of user supplied arguments. If they differ I print an error message.

AddExpr, SubExpr, DivExpr, MultExpr and ModExpr

These nodes represent math operations e.g. 1 + 1, 3 - 1, 2 / 2, 4 % 2 and 2 * 2.

I check to see if the arguments are:

- int and const
- int and variable
- real and const
- real and variable

If they are I print an message expressing this information.

I return a list of both number arguments e.g. 1+1 would return [1, 1].

Condition

This node represents a boolean expression e.g. true OR true.

I check to see if the arguments are:

- boolean and const
- boolean and variable

IdentList

This node represents a list of identifiers. This is used during VarDecl. e.g. real ident1, ident2, ident3 : int

The ident1, ident2, ident3 part is the IdentList.

I loop through all identifiers and get their values and return a list.

Number

This node represents a number e.g. 1 or 1.1

If a number contains a "." it is considered a real otherwise it is an int.

Boolean

This node represents a boolean e.g. true or false.

It just sets the type.

Identifier

This node represents identifiers e.g. a, b, c, etc.

I look for the identifier in both current scope symbol table and global scope symbol table. If it isn’t in either I print an error saying the variable isn’t delcared.

If the variable is declared and found I mark it as being read from.