EECS665

Compiler Construction

Drew Davidson
Ruturaj Vaidya

Lecture: LEEP2 G415
MWF 3:00-3:50

Lab: Eaton 1005B

Project 4

Due on November 5th 11:59 PM

Accepted for 90% credit or 1 late day on 11/6 11:59 PM

Accepted for 80% credit or 2 late days on 11/7 11:59 PM

Accepted for 70% credit or 3 late day on 11/8 11:59 PM

Updates

1. The tarball of files has been uploaded and descriptions for the files you'll probably need to change have been included

Overview

For this assignment, you'll enhance the Lil' C compiler by implementing name analysis. Your goal is to implement a symbol table that is populated by traversing the AST. You'll also need to modify the AST nodes to assist in the collection of symbol table entries.

What you'll do: Alter the unparse function of the AST nodes such that every *use* of an ID has its type, in paretheses, after its name. Write error checking and reporting code for the errors described below. Make any changes in the code necessary to support the above tasks.

How we'll grade: We will run the new P4 driver class with various test-case input files, then diff the expected results from the output of your code. We'll use test files that we expect to unparse correctly, and test files that have type errors that we expect to report type errors.

Unparse Output Format

Since we'll be grading with diff, we'll require a very specific format for the unparsed data. The following format should be used for the type data in parentheses after the ID:

- For names of functions, the information should be of the form

  ```
  param1Type,param2Type,...,paramNType -> returnType
  ```

- For names of global variables, parameters, and local variables of a non-struct type, output `int` or `bool`. For a global or local that is of `struct` type, the information should be the name of the struct type.

▶ **Example** *(click the dang triangle to expand)*

Error Reporting

Your name analysis should find all of the errors described in the table given below; it should report the specified position of the error, and it should give exactly the specified error message (each message should appear on a single line, rather than how it is formatted in the following table). Error messages should have the same format as in the scanner and parser.

Type of Error	Error Message Description	Position to Report
More than one declaration of an identifier in a		The first character of the

given scope (note: includes identifier associated with a struct definition)	Multiply declared identifier	ID in the duplicate declaration
Use of an undeclared identifier	Undeclared identifier	The first character of the undeclared identifier
Bad struct access (LHS of dot-access is not of a struct type)	Dot−access of non−struct type	The first character of the ID corresponding to the LHS of the dot-access.
Bad struct access (RHS of dot-access is not a field of the appropriate a struct)	Invalid struct field name	The first character of the ID corresponding to the RHS of the dot-access.
Bad declaration (variable or parameter of type void)	Non-function declared void	The first character of the ID in the bad declaration.
Bad declaration (attempt to declare variable of a bad struct type)	Invalid name of struct type	The first character of the ID corresponding to the struct type in the bad declaration.

Note that the names themselves should *not* be printed as part of the error messages.

During name analysis, if a function name is multiply declared you *should* still process the formals and the body of the function; don't add a new entry to the current symbol table for the function, but do add a new hashtable to the front of the SymTable's list for the names declared in the body (i.e., the parameters and other local variables of the function).

If you find a bad variable declaration (a variable of type void or of a bad struct type), give an error message and add nothing to the symbol table.

If a declaration is both "bad" (e.g., a non-function declared void) and is a declaration of a name that has already been declared in the same scope, you should give two error messages (first the "bad" declaration error, then the "multiply declared" error).

Skeleton Files

You can use the code from your previous project, enhanced with the new main file P4.cpp. If you'd prefer to start fresh, you may use the tarball here: p4.tgz

▶ **Description of each file** *(click the dang triangle)*

Advice

This section is optional reading for those looking for more guidance on the project. It does not contain any additional requirements.

Don't panic.