CS4318: Compiler Constructions
Project Phase 3: Semantic Analyzer for mC
100 Points

Submission Deadline: 18th April, Friday. 11:59 PM

Objectives

Write a semantic Analyzer for mC

Description

Your task for this project is to build a static semantic analyzer that augments the symbol table and AST (built by the parser) with appropriate semantic information. The semantic analyzer will utilize the symbol table and AST to detect errors in the program related to scoping rules and type mismatch.

Extending the Symbol Table

In this phase, you need to classify the identifiers into two categories: identifiers that refer to variables and identifiers that refer to functions. For each variable the symbol table should contain three types of information : name, type and scope. For function identifiers, the type is represented as the functions signature. The function signature includes the return type, the number of parameters and the type of each parameter.

Types in mC

A variable identifier in mC can have the following basic types : int, char or void. A variable identifier can also be an array of each of the basic types. For this assignment you do not have to do any type promotion. You only perform type checking based on the types provided in the original source. For e.g., if int a = 5; float b = 5.0; then the types of a and b will be considered different.

Scoping Rules in mC

There are only two scopes possible for a variable identifier: global and local to a function. Any variable declared outside a function is considered global and any variable declared in a function is local to that function. However, you can have multiple local variables with the same name in different functions. A local declaration shadows the global declaration. All functions are considered global.

Semantic Error Checking

The semantic analyzer needs to be able to detect the following errors:

1. **Undeclared variables and undefined functions.** A variable or function is considered undeclared if it does not have a declaration in the current scope.
2. **Multiply declared variables and multiply defined functions.** A variable or function is considered multiply declared if it has more than one declaration in the current scope. Note, if you have two declarations of the form **`int a; char a;`** in the current scope then a should be considered multiply defined.
3. **Function declaration/call mismatch.** You need to catch mismatch regarding number of arguments and type mismatch in any of the arguments.
4. Indexing an array variable with a non-integer type
5. Indexing an array with an out-of-bounds integer literal
6. **Type mismatch in assignments.** If the right hand side and the left hand side of an assignment statement is different then your semantic analyzer should generate an error message. For example, trying to assign a void value to an int causes an error, trying to assign a **`char`** to a **`void`** causes an error, etc. You only need to perform this check when the right hand side is a simple expression (i.e, a variable or literal) or a function call.

Error Handling

Two folders containing the cases and the expected results for each of the tests are given for you. You can test your program using these test cases and comparing your results with the expected results provided.

Implementation and Testing Instructions

- You are provided the following files - `driver.c, parser.y, scanner.l, strtab.c, strtab.h, tree.c, tree.h,` and the `makefile.`
- Several test cases are provided for you to test your code. It is only a subset of the cases that will be used to test your code for grading. Please think about edge cases where your program may fail. Testing your code is equally important as writing the code itself.
- Your program will be tested on Zeus, a university provided computing environment.

Writeup

You need to write a short document named `writeup.txt`.
In it, you should include the following (numbered) sections.

1. Your name, course id.
2. If a group project, then who contributed in which part.
3. Provide two examples of how you will test the correctness of your code.
   These examples should be different from the ones provided in this document.
4. Extra grading instructions if you choose not to use makefile.

Compilation, Testing, and Submission

- Your program should be compiled using the commands `make clean; make.`
- You are allowed to do otherwise but make sure to add proper instructions in the `writeup.txt` file.
- The makefile for building your scanner is included.
- If you have made corrections to your previous files, make sure to include those.

Category	**Points**
Symbol Table (type scope, function info)	20
Undeclared/ Multiply Declared Variables	10
Undeclared/ Multiply Declared Functions	10
Function call mismatch	10
Array indexing	10
Type Mismatch	10
Write up	10
Documentation	10
Compile & Run	10

Table 1: Rubric for Project Phase 2

Submission Deadline: 18th April, Friday. 11:59 PM

Late Amount	**Points Deducted**
1 Day	10%
2 Days	20%
3 Days	30%
4 Days	40%
5 Days	50%
> 5 Days	No Points