CS4318/CS5531
Assignment 3: Semantic Analyzer for `mC`
100 Points
Due: Apr 10, Friday 11:59 PM

Objective

Write a semantic analyzer for `mC`

Description

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

Building 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, in addition to `name` and `scope`, you need to store the function signature. The functions signature includes the return type, the number of parameters and the types 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.

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. 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:

- Undeclared variables and undefined functions
- Multiply declared variables and multiply defined functions
- Function declaration/call mismatch. You need to catch mismatch regarding number of arguments and type mismatch in any of the arguments.
- Indexing an array variable with a non-integer type

- Basic type mismatch. (e.g. a `char` being assigned `void`). We will use strict typing for `mC`, so a `char` being assigned to `int` would be illegal. You only need to check for mismatches when the right hand side of the assignment statement is just a variable. That is you do not need to check for expression type mismatch.

Error Handling

Your parser needs to generate meaningful error messages for the semantic errors mentioned above. In generating the error message, you should attempt to provide the line number where the error occurred.

Submission

This is a pair-programming assignment. You need to choose a partner and let me know who you will be working with. If you prefer to work by yourself let me know that too.

Create a README.txt that contains a listing of files required to build your parser. The README should also contain build instructions, special comments and known bug information. For this assignment, you also need to submit a makefile that builds your compiler. Your executable should be called `mcc`. Note, you will also need to resubmit your scanner and parser. If you have made corrections to your scanner or parser since the last submission you should submit the newer version.

Create a tar archive called assg3.lastname (lastname of person making the submission) that includes the README.txt and all files required to build your parser (.l, .y, .c, .h, makefile etc). Submit the tar archive using the drop box on the course web page by the due date.

Extra Credit : Constant Folding (15 points)

Constant folding is an optimization that can be performed at the semantic analysis phase. The basic idea is this: If the AST has two leaf nodes that are both integers and whose parent is an expression node that represents an arithmetic operation then we can perform that operation and replace the expression subtree with a single leaf node that represents the result of the operation. Of course, this same procedure can be propagated upwards, thus performing all constant operations within a program during compile time.