Lab 5

COEN 175 Compilers

Overview For Lab 5

Goal

- Create a symbol table

Submission

- Submit a tarball of your cpps and make files in folder called phase3
- Due Date: Sunday February 6th

Main Objectives for the entire phase

- You will be given a working solution for phase 2
- Modify your parser
- Write a checker
- Make Symbol, Scope, and Type Classes

Goals for this week

1. Make your Symbol class
2. Make your Scope class
3. struct definitions
4. Checker.cpp - Implement Scope functions
5. Checker.cpp - Implement Variable functions
6. Checker.cpp - Implement Function functions

Rerun your Parser

- Check your code on the examples from phase 2
- Make sure you haven’t broken your parser
- The assignment has changed slightly

1. Symbol Class

- Don’t overthink it
- Literally just contains a string name and a type

2. Scope Class

- Attributes:
  - Pointer to enclosing scope
  - Vector of symbols
- Methods
  - Insert() - insert symbol
  - Remove() - remove symbol
  - Find() - Find and return a symbol with a given name within that scope
  - Lookup() - Find and return nearest symbol with a given name in that scope and all enclosing scopes

3. struct definitions

- Struct definitions have a separate namespace
    - NOT on the symbol table
- Single global set for struct definitions
- Structure types are declared (but not complete) at open brace
- May want to write openStruct()/closeStruct()

4. Checker.cpp - Scopes

- Global variables for current scope
- openScope() - creates new scope and passes it enclosing scope
- closeScope() - pops off current scope

5. Checker.cpp - Variables

- declareVariable()
  - Find() if already declared in current scope
  - If found check for E3 or E2 depending on if current scope is global
  - If not found add to current scope
- CheckID - lookup() ID to see if given name has been declared, if not issues [E4]
- CheckIfStruct - checks to see if type is non-pointer struct type and issues [E5]
  - Functions, parameters, and callbacks

6. Checker.cpp - Functions

- declareFunction()
  - Functions always declared in global
  - Very similar to declareVariable()
- defineFunction()
  - Check for redefinition (type already has parameters)
  - Check for conflicting declarations
  - Always insert newest definition

Tips

- Only variable (non-parameter and non-callback) declarations can have a struct type, all others must have pointers to structs [E5]
- Structures must be defined before variables can have their type [E6], (though you can have pointers to struct types)
- Functions always use the most recent declaration or definition
- **READ THE SEMANTIC RULES CAREFULLY**