Lab 7

COEN 175 Compilers

Overview For Lab 7

Goal

- Create a Type checker

Submission

- Submit a tarball of your cpps and make files in folder called phase4
- Due Date: Sunday February 20th

Goals for this week

1. Finish implementing non-postfix checker
2. Modify postfix
3. Implement postfix checkers
4. Modify statement and assignment
5. Implement statement and assignment checker

1. Finish implementing non-postfix checker

- Check for error type in any operand, if so then return error

```
if (left.isError() || right.isError()) return error;
```

- May find some functions work better returning to errors early, others work better defaulting to errors at the end

2. Modify postfix

- Index
  - Return from expression is your index
- Direct/Indirect Reference
  - pass in return from identifier() to checker functions
  - careful with lvalues

2. Modify postfix - Call

- Gather up parameters in Parameters variable
- checkCall
  - If left is error then return error
  - If left is not function and not callback then report E7 and return error
  - If any parameter is not a value type then report E8 and return error
  - Get declaredArgs pointer from left type
  - If declaredArgs pointer is nullptr then return new Scalar type based on specifier and indirection
  - If declaredArgs and parameters are not the same size then report E8 and return error
  - If call isCompatibleWith on each declaredArg/parameter combo, if false for any then report E8 and return error

3. Implement postfix checkers

- Index
- Call
  - Check if parameters are value type even if function definition has no parameters
- Reference
  - Fields map in checker.cpp contains scope pointers
  - Scopes represent struct definitions
  - specifier() on left returns the struct name
  - Use fields map, struct name, and identifier to find resulting type

```
Symbol *symbol = fields[left.specifier()]->find(id);
```

4.Modify statement and assignment

- Return
  - Pass down return type (not function type) as parameter in statement(s) or global variable

```
match('{');
declarations();
statements(Scalar(typespec, indirection));
closeScope();
```

- Conditions
  - Pass in return from the expression function
- Statements/statement can still return void
- Assignment just takes in left-side lvalue and expression types

5. Implement statement and assignment functions

- Return
  - Check if return type and expression type are compatible
- Conditions
  - Just test if type a value type
- Assignment
  - Check if left is an lvalue and left and right are compatible
  - Lvalue is true for arrays, dereferences, and some function calls
- All
  - Call checker function BEFORE semicolon so line numbers match

Reminders for lvalue

- Set lvalue after every operator
- Declare a new lvalue set to false before every expression call (other than in primaryExpression)

```
static void assignment()
{
    bool lvalue = false;
    Type left = expression(lvalue);
```

Tips

- Read carefully to translate the rules from english to C++ code
  - Individually they are not complicated logic
- Recompile your code frequently to make sure it still works
- Check for error types first thing in checker functions
- pointer(pointer(incomplete)) is NOT a pointer to an incomplete type
- Run your code on CHECKSUB before submitting
- Check your code on each operator before moving on
- **READ THE SEMANTIC RULES CAREFULLY**