COP 4020: Programming Languages

Fall 2018

Project 2

"The reCALC Interpreter Variant B"

Instructions

Submission Deadline: Friday, November 16th 2018, 11:59 PM, before which you should submit your recalc.y file to Canvas.

Purpose

This project is intended to give you experience in using a parser generator (YACC) a parser, writing a syntax specification (grammar) for a language, performing parsing and semantic analysis (attribute grammar).

Summary

Your task is to write an interpreter for a simple program whose programming language, reCALC, contains basic language constructs such as variables, assignment statements, simple control flow, and output. The interpreter will be written using a compiler generator (YACC). The program should call the lexical analyzer (Lex) for the next token, parse the token stream, report grammatical errors, perform static and dynamic semantic checks.

You will need to construct a YACC specification file recalc.y which will be use in conjuction with the provided lexer.l to create your interpreter. The executable can be built (and tested with 0.test, for example) in the following way:

```
$ yacc -d recalc.y
$ flex -I lexer.l
$gcc lex.yy.c y.tab.c -lfl
$ ./a.out < tests/0.test
```

The included proj2.tar contains the needed lexer.l file,a sample executable, and a directory of test files.**Do not change the lexer.l file to suit your needs – it will be used as is to test your recalc.y file.**

Syntax Specification

The syntax of the reCALC language is described by a set of syntax diagrams in syntax.pdf. A syntax diagram is a directed graph with one entry and one exit. Each path through the graph defines an allowable sequence of symbols. For example, the structure of a valid reCALC program is defined by the first syntax diagram. The occurrence of the name of another diagram such as declaration and compound statement indicate that any sequence of symbols defined by the other diagram may occur at that point. The following is an example valid reCALC program, called ex.recalc.

```
program test is
{
val1, val2 :: int;
farewell :: string;
val1 = 10;
farewell = “Farewell”;
print “The number “ . val1 . “ is held in val1.”;
print farewell . “!”;
end
```

which yields the following result when run through the reCALC interpreter:

```
$ ./a.out < ex.recalc
The number 10 is held in a.
Farewell!
```

Your first task in this assignment is to develop a context free grammar for the reCALC language from the syntax diagrams.

YACC Specification

Your second task in this assignment is to express your grammar as a YACC specification. You will want to run your specification through YACC to ensure that the grammar produces no parsing conflicts (compiled with yacc -v). If conflicts are indicated by YACC, you should alter your grammar to eliminate them without changing the language accepted by your grammar or ensure that YACCs handling of the conflict agrees with thereCALC language specification. It is suggested that you first develop a parser that merely prints out ACCEPT for syntactically correct reCALC programs and REJECT with error messages for incorrectly structured reCALC programs, calling the lexical analyzer for the tokens. After you have guaranteed that your YACC specification is syntactically sound, you will extend your YACC grammar by adding attributes and semantic rules with actions to develop an interpreter for reCALC that does various static and

dynamic semantics checks, and performs the calculations specified in the reCALC program. Besides reporting grammatical errors, the interpreter also performs the following semantics checks:
Duplicate declaration: A variable is declared multiple times.
Undeclared variable: A variable is used before it is declared.
Uninitialized variable: A variable is referenced before initialized.
Division by 0 error: The denominator in a division expression is 0.
Type error: Use of a constant or variable in an incorrect context due to type.
The program can exit after reporting one semantic/syntax error. If there is no error, the program should execute each statement and output the result of the expression in each print statement (interpreter function). Test out the sample executable to see what is expected.

To facilitate semantic checks and program interpretation and translation, a symbol table must be created to store variables and the related information. The symbol table will need to have at least four fields: the name of the variable, the type of the variable (string and integer types in reCALC), the value of the variable, and a tag indicating whether the variable has been initialized. You may assume that strings can be no more than 2000 characters in length. A good idea would be to create an array of structs, where each struct corresponds to an entry in the symbol table. You may limit the size of this array to2000entries. When the parser encounters an identifier in the declaration, the identifier must be inserted into the symbol table. When the parser encounters an identifier in an expression, it must look up the symbol table to check if the variable has been initialized and obtain its value (if initialized). After the parser processes an assignment statement, the value of the variable in the left-hand side of the assignment statement must be updated in the symbol table.

Interpreting reCALC with YACC is relatively simple. Basically, for every assignment statement, the program first evaluates the value of the expression in the right-hand side of the statement. After that, the interpreter updates the value of the variable in the left-hand side of the statement. When processing a print statement, the interpreter evaluates the value of the expression and prints the result to the standard output.

Your parser should print appropriate error messages. You do not have to implement any error recovery. **Your program should match the output of the sample executable exactly.**

Assignment Submission Instructions

Your recalc.y file should have a header comment that has your name and FSUID. **Ensure your recalc.y works when compiled in the manner described in the Summary section of the first page.**

Recognize correct programs and detect incorrect programs (30).

Semantic checks and error reporting (10).

Correct Interpretation (10).

Your program will be tested with a series of reCALC programs. Some of the testing programs are provided in the project package. You are encouraged to modify the programs to further test your scanner. **If your recalc.y file cannot be used to create a y.tab.c file, 0 points will be given.**