Sarah Huang
COSC 461: Intro to Compilers
October 16, 2022

Cexpr Project Report

1. The problem I attempted to solve is to make a scanner and parser using lex and yacc respectively for a calculator based on C integer expressions. It also supports 26 integer variables named after each letter of the alphabet. Expressions include basic arithmetic, parentheses, bitwise operations, shifts, negation, and assignment.

2. First, I took a look at the provided example files ex.l and ex.y. I was still not familiar with yacc grammar rules and the lex token defining process, so I referred to these examples and did outside research on the syntax. For scan.l, I thought it would be helpful if I labeled every supported operation so I knew to only use these characters in cexpr.y

```
In scan.l
"="          { return  EQUAL; }
"+="         { return  ADD_EQUAL; }

And so on...
```

```
In cexpr.y
equal: VAR EQUAL equal {}
     | VAR ADD_EQUAL equal {}

And so on...
```

   In cexpr.y, I would set the $$ token to equal INT_MIN if there is an overflow and dividebyzero error. When you encounter these errors, the program should not print anything, but it has to get to the top of the parser. To remember to not print since there is an error, I store do `$$ = INT_MIN;` and check for that before printing the output at the end.

3. To check the program, I tested the example input text file and any expressions I could think of including divide by zero and integer overflow errors. I made sure to use ref_cexpr to verify all my outputs.

4. I was not familiar with lex and yacc, so I did prior research and looked at simpler lex and yacc calculators that only did basic arithmetic. I wish the errors messages were more helpful instead of “invalid expression” because it made it harder to identify where the problem was. Additionally, being unable to use yaac’s built-in precedence %left and %right tags were a little annoying.