# Lex and Yacc Extra Credit

**John Dempsey**
COMP-232 Programming Languages
California State University, Channel Islands
October 18, 2023

LAB 10 and ciLisp project will require you to use two new Unix utilities: lex and yacc. You are encouraged to read papers and view YouTube videos on how to use lex and yacc. Under LAB 10 in Canvas, you can find two YouTube links introducing lex/yacc. (Let me know if you find better videos.)

There is an excellent paper titled **A Compact Guide to Lex & Yacc** also found under LAB 10**.** This introductory paper provides sample code for two calculators called calc and calc3.

**Implement the calc example as described in the paper and you'll receive 5 extra credit points.**

**Implement the calc3 example for the compiler, interpreter, and syntax tree graph and you'll receive an additional 5 points.**

In total, you can receive up to 10 points of extra credit. The 10 points go towards the overall 600 class points but **will not** be added to the 105 out of 150 points needed to pass the lab section of the course.

**YOU CAN COPY THE CODE IN THE PAPER. YOU JUST NEED TO GET IT TO WORK.**

**To receive extra credit, upload your code and run scripts to comp232.com in CALC and CALC3 directories and then email me.**

The run script that I used for calc is:

```
john@oho:~/CALC$ more run
yacc -d calc.y
lex calc.l
gcc lex.yy.c y.tab.c -ll -o calc
```

**Note:** The calc.y code on page 15 is missing a "#include <stdio.h>" after the first %{ group needed to support the printf calls.

Below is sample output of the calc program found in Practice, Part II (usually found on pages 16-18).

```
john@oho:~/CALC$ calc
12+5
17
6-9
-3
22 * 10
220
22/10
2
q
Quit
```

Below is sample output of the calc3 program found in Calculator section (usually found on pages 19-33).

To compile the graphing calc3 program, you can use:

```
john@oho:~/CALC3$ more runi
yacc -d calc3.y
lex calc3.l
cc interpreter.c lex.yy.c y.tab.c -o calc3i

john@oho:~ /CALC3$ more runc
yacc -d calc3.y
lex calc3.l
cc compiler.c lex.yy.c y.tab.c -o calc3c

john@oho:~ /CALC3$ more rung
yacc -d calc3.y
lex calc3.l
cc graph.c lex.yy.c y.tab.c -o calc3g
```

The above run scripts are executable:

```
john@oho:~ /CALC3$ ls -l run*
-rwxr-xr-x 1 john john 68 Mar 29 22:32 runc
-rwxr-xr-x 1 john john 65 Mar 29 22:30 rung
-rwxr-xr-x 1 john john 71 Mar 29 22:31 runi
```

The program used in the paper is:

```
john@oho:~$ more program.txt
x = 0;
while (x < 3) {
  print x;
  x = x + 1;
}
```

The example run for the compiler calc3c follows:

```
john@oho:~ /CALC3$ calc3c < program.txt
      push   0
      pop    x
L000:
      push   x
      push   3
      compLT
```

```
    jz    L001
    push  x
    print
    push  x
    push  1
    add
    pop   x
    jmp   L000
L001:
```

The example run for the interpreter calc3 follows:

```
john@oho:~ /CALC3$ calc3i < program.txt
0
1
2
```

The example run for the graphing syntax tree calc3 follows:

```
john@oho:~ /CALC3$ calc3g < program.txt
```

```
Graph 0:

    [=]
     |
   |----|
   |    |
 id(X) c(0)

Graph 1:

                while
                  |
     |----------------|
     |                |
    [<]              [;]
     |                |
   |----|     |----------|
   |    |     |          |
 id(X) c(3) print       [=]
              |          |
              |     |-------|
              |     |       |
            id(X) id(X)    [+]
                            |
                          |----|
                          |    |
                        id(X) c(1)
```