UNCA CSCI 431
Exam 1 Spring 2019
Open textbook section
10 April 2019

This version contains *some* answers.

This is the open textbook part of the exam. Communication with anyone other than the instructor is not allowed during the exam. **Furthermore, calculators, cell phones, and any other electronic or communication devices may not be used during this exam.** Anyone needing a break during the exam must leave their exam with the instructor. Cell phones or computers may not be used during breaks.

This exam must be turned in before 1:45 PM.

Name:________________________________

Give short definitions for the following terms:

Problem 1 (20 points) Possible Program Problems
Suppose a file contains the following alleged C code. Indicate the lexical (scanner), syntax (parser), static semantic, and *possibly* dynamic semantic errors in the following long example. It might be a good idea to explain your reasoning.

It seems like everything is legal C. Of course, your program might crash.

```
char g(int) ;
```

Legal C prototype
If illegal, would syntax error

```
int f(X int) {
```

Static semantic error – int is a type

```
     int temp ;
```

Because X wasn't legally declared, all statements using X would result in a **static semantic** error.
However, if X were an int, all would be OK in C and Java.

```
     float Y = X++ + 431 ;
     int z = (int)Y + X ;
     if (z<0) {
```

Undefined reference error could occur in link if g missing
Compiler would not "know" this. Some kind of semantic

```
          temp = X + g(X) ;
     return temp + 7;
}
```

Syntax error – There is no closing }

Problem 2 (4 points)

What exactly is *enclosed* by an *enclosure*?

Relevant parts of the execution environment needed to run the enclosure. This would be the “external” variables used within the enclosure.

For example, in the Python lambda expression:

```
    lambda i, j -> i * x * y + j * y
```

The values of x and y would need to be enclosed.

Does a Java method reference, such as c in the example below:

```
    Consumer<String> c = System.out::println ;
```

really require an enclosure?

c must be implemented by a class implementing a functional interface. This would certainly need a reference to System.out.println, but is that an addition to the “execution environment” since it’s out is a static field of a static class?

Problem 3 (6 points)

Write, in both Java and Python, lambda expressions implementing a function (in Python) and functional interface (in Java) that receives an argument *X* and returns *X*+431. (Yes, they are very similar.)

Java: `x -> x + 431`
Python: **lambda x : x + 431**

Problem 4 (12 points)

Translate the following C expressions into **both** prefix and postfix notation:

```
    sqrt(x) + y * (a + c) % z
```

In C, Java and Python: % and / have the same precedence and are left to right associative. This means the expression is evaluated as

```
    sqrt(x) + (y * (a + c)) % z
```

Try out 2 * 3 % 5 in gdb, jshell, and python. The result is 1.

```
Prefix:     +   sqrt   x   *   y   %   +   a   c   z
Postfix:    x   sqrt   y   a   c   +   *   z   %   +
```

Prefix and postfix should *never* have parentheses. That’s what makes them both cool and useful.

Problem 5 (10 points)
Consider the following psuedocode, adopted from page 171 of the textbook.

```
procedure P(A, B: real)
   X: real
   procedure Q(B, C: real)
      Y: real
      … body of Q
   procedure R(A, C: real)
      Z: real
      … body of R
   … body of P
```

What procedures can be called and what variables (including procedure arguments) can be accessed from the *body of Q*?

All procedures – P, Q and R
All variables in Q – B, C and Y
No variables in R
A and X in P

What procedures can be called and what variables (including procedure arguments) can be accessed from the *body of R*?

All procedures – P, Q and R
All variables in R – A, C and Z
No variables in Q
B and X in P

Problem 6 (10 points)
Continue with the Problem 5 psuedocode. Suppose that P calls R which calls Q which calls P which calls R as shown in the preceding problem. Draw an abstract picture of the stack containing all five active stack frames **which also illustrates the static and dynamic links.**

Drawing a diagram in an editor is too much for me, so I'm doing this horizontally.

The stack with dynamic links goes like this. That is each call points to the callee
P «««« R «««« Q «««« P   «««« R
We having no idea what called the first P, so me admit it's dynamic link.

The static link is a bit messier. With only one level of "enclosure" at most one static link is needed for each procedure. Notice that both Q and the first R, point to the first Q.

```
P <+= R   +=Q       P  «««« R
     '====='
```

Problem 7 (12 points)

Start with the following C structure:

```
     struct CS {
          int I;
          char C[5] ;
          float D ;
          short S ;
     }
```

Given the usual x86_64 alignment what would be the offsets of the four fields from the beginning of the structure? (If you are not sure what the “usual” alignment, state your assumptions.)

I at offset 0
C at offset 4
D at offset 12
S at offset 16
Total length is 18

Problem 8 (12 points)

Continuing with the structure of Problem 7, suppose X is a two-dimensional array of `struct CS` declared as follows:

```
     struct CS X[431][235] ;
```

If α is the address of the base of the array, what is the address of the start of element `A[i][j]` of the array? Show your fancy math!

Size of struct must be at least 20 to allow int I to have appropriate alignment
Many C compilers will align at 8-byte boundaries, which would be 24
Address of `A[i][j]` would be α + I * 235 * 20 + j * 20

Also what is the address of `A[i][j].C[3]` ?

Offset of C[3} with the structure would be 7
Address of `A[i][j].C[3]` would be α + I * 235 * 20 + j * 20 + 7