CSE305, Spring 2023 Assignment 6 Due Thu. Apr. 20, 11:59pm

The **Second Prelim Exam** will be on **Thursday, April 27** in class period. It will cover material cumulatively through chapter 12 of the text and assignments 1–6 (esp. 4–6). **Reading:** For this coming week, read Sebesta chapters 11 and 12. Chapter 11 should be largely review for you from CSE250 or equivalent course, but it sets up a contrast to classes. **There is no TopHat part.**

(1) (9+18+12+6 = 45 pts.) Consider the following code written in C:

```
#include "stdio.h"
int main(int argc, char** argv) {
   int i = 3;
   i += 7*(i = 4);
   int j = 3;
   j = j + 7*(j = 4);
   printf("After doing i,j = 3; and i += 7*(i = 4); but j = j + 7*(j=4); we have \n");
   printf("i = %d and j = %d\n" , i, j);
   j = (i = 5) + i + 0*(i = 10) + i;
   printf("After j = (i = 5) + i + 0*(i = 10) + i; we now have \n");
   printf("i = %d and j = %d\n" , i, j);
   return 0;
}
```

(a) Compile and run this with these three compilers available on `timberlake`: `gcc`, `CC`, and `clang`. (You may use `g++` instead of `gcc`, pretending the code is in C++. No optimizations or other switches are needed.) You may get all different answers. Say what they are. (9 pts.)

(b) Translate the code into our "rudimentary stack language." Because we have made the convention that `x += EXP` is supposed to be translated the same as `x = x + EXP`, and because `j` is only an *lvalue* in the line `j = (i = 5) + i + 0*(i = 10) + i;` you can skip doing the lines `int j = 3;` and `j = j + 7*(j = 4);` here. (But was this identity between `i` and `j` true for each compiler? *Hmmmm...* 18 pts.)

(c) Trace the results of your stack code in (c) by hand, showing changes to the storage objects for `i` and `j` as they happen. Which, if any, compiler did your results agree with? (12 pts.)

(d) Now choose **either** the equivalent code in Java **or** the equivalent code in `Javascript` below. (They give identical results to each other in my tests—should they on any system?) Compile and run it. Does it agree with any C compiler, and/or with your stack code trace? (6 pts.)

```
public class WeirdCode {
   public static void main(String[] args) {
      int i = 3;
      i += 7*(i = 4);
      int j = 3;
      j = j + 7*(j = 4);
      System.out.println("After i,j = 3; and i += 7*(i = 4); but j = j + 7*(j=4); we have");
      System.out.println("i = "+ i +", and j = " + j);
      j = (i = 5) + i + 0*(i = 10) + i;
      System.out.println("After j = (i = 5) + i + 0*(i = 10) + i; we now have ");
      System.out.println("i = "+ i +", and j = " + j);
      System.exit(0);
   }
}
```

```
//Javascript
let i = 3;
i += 7*(i = 4);
let j = 3;
j = j + 7*(j = 4);
console.log("After doing i,j = 3; and i += 7*(i = 4); but j = j + 7*(j = 4); we have ");
console.log("i = " + i + ", and j = " + j);
j = (i = 5) + i + 0*(i = 10) + i;
console.log("After int j = (i = 5) + i + 0*(i = 10) + i; we now have ");
console.log("i = " + i + ", and j = " + j);
```

(2) (12 pts.) Compile and run the following C++ code. Except for the use of call-by-reference in "`int& y`," this would be C code.

```
#include <stdio.h>
int foo(int x, int& y) {        //also change to int foo(int x, int y)
   y = x + 1;
   return x + y;
}
int main(int argc, char** argv) {
   int x = 4;
   int y = 3;
   int z = foo(x,y);
   printf("z + y = %d\n",z+y);
}
```

With diagrams like those in lectures for call-by-reference versus call-by-value, explain what happens and what is printed, also when "`int& y`" in the header is changed to "`int y`" to use call-by-value.

(3) (18 pts., for 75 total) Translate the following two C/C++ functions into OCaml. Use recursion to simulate a `for`-loop or `while`-loop, and an OCaml `list` to simulate an array. Have OCaml return the whole list (as implemented, it is a pointer anyway). You must use tail recursion and accumulator-passing style. In the second one you must make `f` a parameter, i.e. write a higher-order function.

```
int* prefixSums(int* a, int n) {  /* n is the length of the array a */
   for (int i = 1; i < n; i++) {
      a[i] += a[i-1];             /* LOOP INV: a[i] holds sum of original */
   }                              /* array entries up to i.               */
   return a;                      /* return pointer to modified array     */
}

int f (int x) {... unspecified ...}   //make f a *parameter* in the OCaml code.

int numIterations(int arg, int stop) {
   int count = 0;
   while (arg != stop) {
      arg = f(arg);
      count++;
   }
   return count;
}
```