CSci 4651 Fall 2008
Problem Set 7: Storage management.
Due Friday, November 7th in class

IMPORTANT, please read before you start working on the problems: This problem set uses Java-like syntax for convenience. However, this is NOT Java code. Read each problem carefully to understand which model you are working in (pass-by-value vs. pass-by-reference, static vs. dynamic scope rules, etc.). Assume that all code given below is syntactically correct and does not cause any compilation errors. Assume that `print("x = " + x)` is a printing command which works like `System.out.println("x = " + x)` in Java for all types of variables used in this problem set.

Problem 1 (5 points). Consider the following code (the lines are numbered for easy reference):

```
1.   int x = 3;
2.   {
3.       int y = 5 + x;
4.       {
5.           int x = 2 + y;
6.       }
7.       x = y;
8.   }
```

Question 1. Draw the program stack right after line 5 gets executed.
Question 2. What is the final value of the global `x` (the one defined on line 1) in this program fragment?

Problem 2 (5 points). Consider the call `f(2, -1)` to the following function:

```
int f  (int x, int y) {
    int z = 0;
    int w = x * x;
    if (y < 0) {
        int z = -w;
        print("z = " + z);
    }
    return z;
}
```

Question 1. Draw the program stack at the point right after the line
`int z = -w;`
is executed.

Question 2. What will be printed by the `print` statement? What will be the value returned by the function? Explain your answers using the stack diagram from Question 1.

Problem 3 (10 points). Which of the following Java methods are tail-recursive? If a method is not tail-recursive, please write its tail-recursive version (make sure to show the first call to the method that initializes the extra parameters, if any). Feel free to test your code; you may submit electronic solutions. The source code is also linked from the assignments page for copy/pasting.

```
public class TailRecursion {
    public static void main(String [] args) {
        int [] A = {3, 6, 7, 5, 4};
        int [] B = {3, 6, 5, 7, 9};
        System.out.println("sum_array(A,0) = " + sum_array(A,0));
        System.out.println("find(A,0,5) = " + find(A,0,5));
        System.out.println("find(A,0,2) = " + find(A,0,2));
        System.out.println("equal(A,B,0) = " + equal(A,B,0));
        System.out.println("equal(A,A,0) = " + equal(A,A,0));
        System.out.println("equal(A, new int[2],0) = " + equal(A, new int[2],0));
        System.out.println("to_string_reverse(A,0)" +  to_string_reverse(A,0));
   }

    public static int sum_array(int [] A, int i) {
        if (i < A.length) return (A[i] + sum_array(A, i+1));
        return 0;
    }

    public static boolean find(int [] A, int i, int x) {
        if (i >= A.length) return false;
        if (A[i] == x) return true;
        return find(A, i+1, x);
    }

    public static boolean equal(int[] A, int[] B, int i) {
        if (i == A.length)
            if (i == B.length) return true;
            else return false;
        if (A[i] == B[i]) return equal(A, B, i+1);
        else return false;
    }

    public static String to_string_reverse(int [] A, int i) {
        if (i < A.length) {
            return to_string_reverse(A, i + 1) + " " + A[i];
        }
```

```
        return "";
    }
}
```

Question 2. Describe how a tail-recursive function may be optimized. Explain why the optimized program is more efficient (what overhead has been eliminated by the optimization?).

Problem 4 (8 points). Consider the following function. Notice that the first parameter is passed by reference, and the second one is passed by value.

```
int f(int &n, int m) {
    n = n + 1;
    m = m + 1;
    return n + m;
}
```

Question 1. Which of the following are valid calls to this functions? Assume that all variables used below are of type `int` and `A` is an array of ints, and all elements of the array referenced below are within the array boundary. Explain your answers briefly.

- `f(x + 1, 5)`
- `f(A[i+1], A[i])`
- `f(2, 3)`
- `f(x, f(x, y))`

Question 2. For the following function call please draw the function stack right after f was called and right before f returns. What will be printed by the program? Justify your answer using the stack diagram.

```
... main (...) {
    int x = 2;
    int y = 0;
    y = f(x, x);
    print x;
    print y;
}
```

Problem 5 (10 points). Consider the following program, where `main` is the first function in program execution. Parameters are passed by value.

```
int x = 2;
int y = 3;

void f(int n) {
    x = x + n;
    y = y - n;
```

```
}

void main () {
    int x = 1;
    f(1);
    {
        int y = 5;
        f(x);
    }
}
```

Question 1. Assuming dynamic scope rules, draw the program stack at two points in execution: right before the call `f(1)` returns and right before the call `f(x)` returns. Show values of all variables in the stack pictures.

Note that for the second function call the stack will have both kinds of blocks: in-line blocks and those associated with a function.

Question 2. What are the final values of **global** `x` and `y` in the case of dynamic scope rules? Use the stack pictures to explain your answer.

Questions 3 and 4. The same as 1 and 2, but for static (lexical) scope rules.

Problem 6 (8 points). Consider the following program where `main` is the first function in program execution. Assume the static scope rules. `int -> int` is the type of functions from an integer to another integer.

```
int x = 0;

int f(int a) {
    if (a == 1) return x;
    else {
        x = x  + a;
        return f(a - 1);
    }
}

void g (int -> int h) {
    int x = 5;
    print(h(2));
}

void main () {
    g(f);
}
```

Question 1. Note that `f` is recursive. Draw the program stack and function closures when all activation records for `f` are pushed on the stack.

Question 2. What is going to be printed in the program? Use the stack picture

to explain your answer.

Problem 7 (8 points). Assume the static scope rules and notations as in the previous problem. Consider the following code fragment, where `f` returns a function `g`. The function type `void -> void` means that the function takes no parameters and does not return a value.

```
1.   void -> void f () {
2.       int x = 0;
3.       return  ( void g() { x = x + 1; } );
4.   }
5.   void main () {
6.       void -> void h = f();
7.       h();
8.       void -> void j = f();
9.       j();
10. }
```

Draw the program stack and all the function closures at the following points of the program execution:

1. Right before line 7 is executed.
2. Right after line 7 is executed.
3. Right after line 8 is executed.
4. Right after line 9 is executed.

Show the values of all variables.