CSE 240
Homework 4 – Programming with LISP

A. What This Assignment Is About:

- Lists
- Control structures (conditions and loops)
- Define Functions
- Recursive Calls
- Data structures

B. Use the following Guidelines

- **In headers add your Name, Date, and Program Description.**
- Give identifiers semantic meaning and make them easy to read (examples num-students, gross-pay, etc.).
- Use tabs or spaces to indent code within blocks (code surrounded by braces). This includes classes, methods, and code associated with ifs, switches and loops. Be consistent with the number of spaces or tabs that you use to indent.
- Use white space and newlines (enters) to make your program more readable.

C. Instructions

Code the following functions in a single lisp file named **homework.lisp**.

1. multiply (number number)
Write a function named **multiply** that takes two numbers as parameters and returns the multiplication of the two numbers.

2. maximum (list)
Write a function named **maximum** that takes a list of numbers as a parameter. Search the entire list and return the largest value from the list.

3. average (list)
Write a function named **average** that takes a list of numbers as a parameter. Return the average of all numbers in the list.

4. count-of (symbol list)
Write a function named **count-of** that takes two parameters, a symbol and a list. Count the number of instances of x in the list. Do not count the number of x in any sub-lists within the list.

5. iterative-factorial (number)
Write an **iterative** function that calculates the factorial of a positive integer and return the result. The function should be equivalent to this Java code.

```
public static double iterative-factorial (double n) {
     double sum = 1;
     for (double i=0; i < n; i++)
            sum = sum * (1 + i);
     return sum;
}
```

6. recursive-factorial (number)
Write a **recursive** function that calculates the factorial of a positive integer and return the result. The function should be equivalent to this Java code.

```
public static double recursive-factorial (double n) {
     if (n<=1)
       return 1;
     else
       return n * recursive-factorial (n-1);
}
```

7. fibonacci (number)
Write a function named **fibonacci** that takes a number, n, as a parameter. Return the nth value. Use the following definition of the fibonnaci sequence:

$$fib(n) = \begin{cases} 0 & if\ n = 0 \\ 1 & if\ n = 1 \\ fib(n-1) + fib(n-2) & otherwise \end{cases}$$

8. trim-at (symbol list)
Write a function named **trim-at** that takes a symbol and list as parameters. Return a new list ending at the first occurrence of the input symbol in the input list (include the symbol). If there is no occurrence of the symbol, return nil.

For example:
(trim-at ‘c ‘(a b c d e))
This should return the following list:
‘(a b c)

9. ackermann (number number)
Write a function named **ackermann** that takes two integers. Use the following function definition: (Note: due to this function’s complexity it may not finish for inputs larger than 3)

$$A(x,y) = \begin{cases} y+1 & if\ x = 0 \\ A(x-1,1) & if\ y = 0 \\ A(x-1, A(x, y-1)) & otherwise \end{cases}$$

10. Copy and paste the following code at the end of your file.

```
(defun test()
(print (muliply 3 1))
(print (maximum '(5 78 9 8 3)))
(print (average '(1 2 3 4 5 6 7 8 9)))
(print (count-of 'a '(a '(a c) d c a)))
(print (iterative-factorial 5))
(print (recursive-factorial 4))
(print (fibonacci 8))
(print (trim-at 'c '(a b c d e)))
(print (ackermann 1 1)))

(test)
```

Sample Output:

```
3
78
5
2
120
24
13
(a b c)
3
```

D. Grading Criteria

- 5 pts: Program executes with no errors
- 5 pts: File is named correctly with adequate comments for the functions and file header
- 10 pts: muliply is correct
- 10 pts: maximum is correct
- 10 pts: average is correct
- 10 pts: count-of is correct
- 10 pts: iterative-factorial is correct
- 10 pts: recursive-factorial is correct
- 10 pts: fibonacci is correct
- 10 pts: trim-at is correct
- 10 pts: ackermann is correct