**ASSIGNMENT - 3**

**Introduction:**

Welcome to the third programming assignment of Fundamentals of Programming Languages.

This homework is divided into 4 parts, each of which has an associated point total. The total of all parts combines to 100 points. In this homework you will be exploring threading and locking in Java.

The homework is due on **Friday November 8th at 11:59 PM.**

Part-1: Infinite Lists (50 Points)

Part-2: Generalized Printing (30 Points)

Part-3: ZIP and Evaluation (20 Points)

Note: Please adhere to the **type signatures** that we request you to follow as you will be graded exactly based on them. If you do not follow the **type signatures** you will be assigned 0 points.

**Requirements and Submission:**

All files should have your name and UB email address in comments at the top of the file prior to any code you have written. Create one tarball which should be named with your UBIT id, underscore and then HW3 (example: UBITID\_HW3.tar), this tarball should contain your source file.

Submit using **submit\_cse505 <filename>**

For details on how to submit, refer to:

<https://wiki.cse.buffalo.edu/services/content/submit-script>

<https://wiki.cse.buffalo.edu/services/content/student-servers>

**Part 1 – Infinite Lists: 50 Points**

Start with the following datatype and function bindings given in the Lesson 12 slides on piazza.

datatype 'a inflist = NIL

| CONS of 'a \* (unit -> 'a inflist);

fun HD (CONS(a,b)) = a

| HD NIL = raise Subscript;

fun TL (CONS(a,b)) = b()

| TL NIL = raise Subscript;

fun NULL NIL = true

| NULL \_ = false;

fun FILTER f l = if NULL l

then NIL

else if f (HD l)

then CONS(HD l, fn () =>

(FILTER f (TL l)))

else FILTER f (TL l);

fun TAKE(xs, 0) = []

| TAKE(NIL, n) = raise Subscript

| TAKE(CONS(x,xf), n) = x::TAKE(xf(), n-1);

You are tasked with creating three infinite lists as well as appropriate helper functions with the following type signatures:

val even = fn : int -> bool

val odd = fn : int -> bool

val fib = fn : int -> int -> int inflist

val fibs = CONS (0,fn) : int inflist

val evenFibs = CONS (0,fn) : int inflist

val oddFibs = CONS (1,fn) : int inflist

Write a function called “even” which will test the argument if it is even. Write a function called “odd” which will test the argument if it is odd. Write a function call “fib” that will generate an infinite list containing the Fibonacci sequence (hint: take a look at the FROMN function in the class slides as an example of a generating function). “fib” should be a curried function that takes two arguments, the first two Fibonacci numbers (ex: fib 0 1) Create an infinite list called “fibs” that has the entire Fibonacci sequence. Create an infinite list called “evenFibs” that has only the even numbers in the entire Fibonacci sequence. Create an infinite list called “oddFibs” that has only the odd numbers in the entire Fibonacci sequence. You should use the FILTER function and the infinite sequence “fibs” to create “evenFibs” and “oddFibs” by filtering the odd and even Fibonacci numbers respectively.

**Part 2 – Generalized Printing: 30 Points**

You are tasked with creating generalized list printing functions. Write a function called “printGenList” which takes a function f and a list l and applies the function f to each element of the list l recursively. The function should have the following type signature:

val printGenList = fn : ('a -> 'b) -> 'a list -> unit

Create a function called “printList” that will pretty print an integer list. You will find the string concatenation operator in SML useful (^). Example: print("hello" ^ " world");. printList should leverage printGenList and provide an anonymous function (the fn … => … construct) that will do the appropriate pretty printing. This anonymous function should print the element of the list and then a space character. printList should have the following type signature:

val printList = fn : int list -> unit

Create a function called “printPairList” that will pretty print a list consisting of integer pairs. printPairList should leverage printGenList and provide an anonymous function (the fn … => … construct) that will do the appropriate pretty printing. This anonymous function should print an open parenthesis the first element of the pair, a comma, a space, the second element of the pair, and then a close parenthesis. printPairList should have the following type signature:

val printPairList = fn : (int \* int) list -> unit

**Part 3 – ZIP and Evaluation: 20 Points**

Write a function called “ZIP” which will zip together two infinite lists (hint: take a look at how the zip function works that we went over in class in lesson 11 on piazza). ZIP should have the following type signature.

val ZIP = fn : 'a inflist \* 'b inflist -> ('a \* 'b) inflist

Use the functions from part 2 and 3 to print the first 20 Fibonacci numbers form the infinite list “fibs”. Print the first 10 even Fibonacci numbers from the infinite list “evenFibs”. Print the first 10 odd Fibonacci numbers from the infinite list “oddFibs”. Lastly, print the first 10 pairs from an infinite list created by zipping “evenFibs” and “oddFibs”. You should use the “TAKE” function to get the required amount of elements from each infinite list.

**Example output for the entire homework:**

use "hw3.sml";

[opening hw3.sml]

datatype 'a inflist = CONS of 'a \* (unit -> 'a inflist) | NIL

val HD = fn : 'a inflist -> 'a

val TL = fn : 'a inflist -> 'a inflist

val NULL = fn : 'a inflist -> bool

val FILTER = fn : ('a -> bool) -> 'a inflist -> 'a inflist

val TAKE = fn : 'a inflist \* int -> 'a list

val even = fn : int -> bool

val odd = fn : int -> bool

val fib = fn : int -> int -> int inflist

val fibs = CONS (0,fn) : int inflist

val evenFibs = CONS (0,fn) : int inflist

val oddFibs = CONS (1,fn) : int inflist

val printGenList = fn : ('a -> 'b) -> 'a list -> unit

val printList = fn : int list -> unit

val printPairList = fn : (int \* int) list -> unit

val ZIP = fn : 'a inflist \* 'b inflist -> ('a \* 'b) inflist

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 val it = () : unit

0 2 8 34 144 610 2584 10946 46368 196418 val it = () : unit

1 1 3 5 13 21 55 89 233 377 val it = () : unit

(0, 1) (2, 1) (8, 3) (34, 5) (144, 13) (610, 21) (2584, 55) (10946, 89) (46368, 233) (196418, 377) val it = () : unit

val it = () : unit