Precept 5

This precept will help familiarize you with material on closure conversion, tail recursion and continuation-passing style. You will also continue work on skills for proving things about programs.

Refer to these notes to help you through this week's precept materials.

http://www.cs.princeton.edu/~dpw/courses/cos326-12/notes/reasoning.php
http://www.cs.princeton.edu/~dpw/courses/cos326-12/notes/reasoning-data.php
http://www.cs.princeton.edu/~dpw/courses/cos326-12/lec/10-space-model.pdf

Part I: Continuation-passing style

Consider the following functions.

```
type nat = int ;;

let rec fact (n:nat) : nat =
  if n <= 0 then 1
  else fact (n-1) * n
;;

let rec fib (n:nat) : nat =
  match n with
    0 -> 0
  | 1 -> 1
  | n -> fib (n-1) + fib (n-2)
;;
```

Write the following tail-recursive functions:

```
type cont = nat -> nat
fact_cont : nat -> cont -> nat
fib_cont : nat -> cont -> nat
```

Prove that your continuation-passing function fact_cont is equivalent to the version of fact presented above. In other words, prove the following lemma and then the theorem:

```
Lemma 1: For all natural numbers n,
         for all k:cont, fact_cont n k == k (fact n)

Theorem 2: For all natural numbers n,
           fact_cont n (fun m -> m) == fact n
```

Part II: Closure Conversion

Closure-convert the following code. In other words, rewrite the functions apply and sum (call them apply_code and sum_code) so that they contain no free variables. Ensure the resulting code type checks.

```
let c = 5;;
let d = 6;;

let apply (f : int -> int) : int = f (c*d) ;;

let sum (y : int) : int = y + c + d;;

let result = apply sum;;
```

Part III: Induction: Trees

```
type tree = Leaf | Node of int * tree * tree;;

let rec inc (t:tree) (a:int) : tree =
  match t with
    Leaf -> Leaf
  | Node(i,left,right) -> Node(i+a, inc left a, inc right a)
;;
```

Prove the following theorem:

```
Theorem 3: for all t:tree, inc (inc t a) b == inc t (a+b).
```