Name: *ANUJA B*

Roll no: *05*

Subject: *Functional Programming*

ASSIGNMENT- II

QUESTION-1

Prove # (zip Xs Ys) = (#Xs) min (#Ys)

Let us prove using induction on both xs and ys.

Case [], ys

#zip ([], ys) = #[ ]   (zip. 1)

= 0   (# .1)
= 0 min (#ys)

Since, 0 min n = 0 for every natural number n.

Case (x : xs), [ ]

#zip (x : xs, [ ]) = #[ ]   (zip.2)

= 0   (# .1)

= #(x : xs) min 0,

Since n min 0 = 0 for every natural number n.

Case (x : xs), (Y : ys)
#zip (x : xS, Y : ys) = #((x , y) : zip (xs , ys) (zip .3)

= l + #zip (xs, ys)   (zip.2)

= 1 + (# xs min # ys) (hypothesis)

= (l + #xs) min (l + # ys) (+ distributes through min)

= # (x : xs) min #(y : ys) (#.2)

In this cases ([], ys), (x : xs, []), and (x : xs, y : ys) cover every possible combination of two lists. Hence, it is proved #(zip Xs Ys) = (#Xs) min (#Ys)

QUESTION-2

Prove take n xs ++ drop n xs = xs.

Let us prove using induction on both n and xs.

Case 0 , xs

0 xs ++ drop 0 xs = [ ] ++ xs (take .1, drop.1)

= xs (++ . 1)

Case (n + 1) , [ ]

take (n + 1) [ ] ++ drop (n + 1) [ ]

= [ ] ++ [ ] (take .2, drop.2)

= [ ] (++ . 1)

Case (n + 1) , (x : xs)

take (n + 1) (x : xs) ++ drop (n + 1) (x : xs)
= (x : take n xs) ++ drop n xs (take .3, drop.3)

= x : (take n xs ++ drop n xs) (++ .2)

= x : xs (hypothesis)

Hence, it is proved.

QUESTION-3

a) Define head and tail using recursion.

b) Prove [hd xs] ++ tl xs = xs

a) Head Recursion: If a recursive function calling itself and that recursive call is the first statement in the function then it's known as Head recursion**.** There's no statement, no operation before the call. The function doesn't have to process or perform any operation at the time of calling and all operations are done at returning time.

Tail Recursion: If a recursive function calling itself and that recursive call is the last statement in the function then it's known as Tail recursion. After that call the recursive function performs nothing. The function has to process or perform any operation at the time of calling and it does nothing at returning time.

The function 'hd' selects the first element of a list, and 'tl' selects the remaining portion. These can be defined by a simple case analysis, without any need to resort to recursion,

hd (x : xs) = x
tl (x : xs)   = xs

b) Proof: Let us prove the result using induction on xs.

Case (x : xs),

[hd (x : xs)] ++ tl (x : xs)
= [x] ++ xs (hd.1 , tl.1)
= x : xs (++.2, ++ . 1)

The case for [] was not included because it is given that xs is non-empty. This simple proof requires only a trivial case analysis, but no induction. Hence, it is proved.

QUESTION-4

a) Define Init and last using recursion.

b) Prove Init xs = take (#xs - 1) xs.

a) These functions are similar to head and tail, but they select the initial segment of the list and its last element. Init and last can be defined using recursion as following:

```
Init [x]            = [ ]
Init (x : x' : xs)  = x : Init (x' : xs)

last [x]            = x
last (x : x' : xs)  = last (x' : xs)
```

b) Proof: Let us prove the result using induction on xs.

Case [x],

Init [x] = [ ] (Init.1)

= take 0 [x] (take.1)
= take (#[x] - 1) [x] (#.1, #.2)

Case (x : x' : xs),

For the induction hypothesis, assume that :

Init (x' : xs) = take (#(x' : xs) - 1) (x' : xs) (hypothesis)

```
Init (x : (x' : xs )) = x : Init (x' : xs)        ( init.2)
= x : take (#(x': xs) - 1) (x': xs)                (hypothesis)
= x : take (#xs) (x' : xs)                         (#.2)
= take (#xs + 1) (x : x' : xs)                     (take.3)
= take (#( x : x' : xs) - 1) (x : x' : xs)         (#.2)
```
Hence, it is proved.

QUESTION-5

Define map and filter using recursion.

The function map applies a function to each element of a list, and the function filter removes elements of a list that do not satisfy a predicate. map and filter can be defined using recursion as following:

```
map f [ ]       = [ ]
map f (x : xs)  = f x : map f xs
```

```
filter p [ ]        = [ ]
filter p (x : xs ) = x : filter p xs , if p x
                    = filter p xs, otherwise
```

The new elements we have here are higher-order functions and conditionals in a recursive function definition.

-----------------------------------End--