CSL101

Solutions for Programming Assignment 2

1. *Write your own function **myadd** that detects overflow (using exception) when you add two integers.*

Solution:

```
exception Overflow ;;

let myadd : int*int -> int = function (x,y) ->
        if (x>0 && y>0) then
                if ((x+y)<=0) then
                        raise Overflow
                else
                        (x+y)
        else
                if (x<0 && y<0) then
                        if ((x+y)>=0) then
                                raise Overflow
                        else
                                (x+y)
                else
                        (x+y) ;;
```

The program should work for all combinations of positive and negative inputs.

2. *Consider a game of NIM that is played between two players P1 and P2. The game begins with n tokens and players P1 and P2. A player can draw one or two tokens from the pile. Their turns alternate and the number of tokens in the pile is known at all times. The player who draws the last token **loses**. Starting from n tokens, we say that there is a winning strategy for player P1 if P1 starts and irrespective of whatever tokens player P2 draws, P1 can win (i.e. force P2 to pick the last token).*
   For example, if n =3, P1 can pick 2 so that P2 must pick the last token.
   Write an ocaml program that given n returns 1 if there is a winning strategy for P1 and 0 otherwise.

Solution :

Player P1 has a winning strategy available, then we have at least one of the two conditions :
a) Player P1 takes 1 coin and Player P2 has no winning strategy for n-1 coins.
b) Player P1 takes 2 coin and Player P2 has no winning strategy for n-2 coins.

For base case, n = 1, Player P1 has no winning strategy.
For n = 2, Player P1 has a winning startegy; P1 can draw 1 coin and P2 will necessarily take the last coin.
So, n = 1 results in false.
n = 2 results in true.

n = 3 results in true.

Manually solving for some values helps in identifying the pattern that P1 does not have winning strategy if and only if (n – 1) is divisible by 3.

```
exception NoOrNegativeTokens;;

let nim : int -> int = function (n) ->
        if n<=0 then
                raise NoOrNegativeTokens
        else
                if ((n-1) mod 3) = 0 then
                        0
                else
                        1;;
```

The problem can be solved recursively also, but that will take linear time with respect to input value.

It will be perferable if the function also prints the number of coins P1 should pick at first for winning strategy.

3. *Towers of Hanoi: Move a stack of n disks arranged in increasing sizes from top to bottom from peg A to peg B one by one such that at no intermediate step a larger disk should reside on a smaller disk. You can use an intermediate peg C for this purpose. Your program should list all the disk moves.*

Solution :

```
exception NoOrNegativeDisks;;

let rec hanoi: int * string * string * string  -> (int * string) list = function (n,a,b,c) ->
match n with
          1 -> [(1,a ^" to "^ b)]

        | k ->   if k<=0 then
                                raise NoOrNegativeDisks
                        else
                             hanoi((k-1),a,c,b) @ [(k,a ^" to "^ b)] @
                        hanoi((k-1),c,b,a);;
```

4. *Write an program to convert an integer (in decimal) to base k (> 1) representation.*

Hint: You may want to use the observation that the least significant digit in base k representation is the number modulo k.

Solution :

```
let rec convert_helper : int * int * int list  -> int list = function (n,k,l) ->
        if ((n/k)=0) then
```

```
                ((n mod k) :: l)
    else
                convert_helper((n/k), k, ((n mod k) :: l)) ;;


let conversion : int * int -> int list = function (n , k) ->
        if (k<=1) then
                raise (Input_Type "base cannot be less than or equal to 1")
    else convert_helper(n,k,[]) ;;
```

5. *Write a function unique : int list-> bool which takes an integer list M as input (with possibly repeated elements) and it returns true if all elements are unique and false otherwise.*
How many operations does your program take for a list of n elements ?

Solution :

```
let rec compare: int * int list -> bool = function(x,xs) ->
match xs with
          []              ->      false
        | y::ys  ->       if x=y then
                                        true
                                else
                                        compare (x,ys);;

let rec unique: int list -> bool =  function(m) ->
match m with
          []              ->      true
        | x::xs  ->       if compare(x,xs) then
                                        false
                                else
                                        unique (xs);;
```

The program takes at most n * (n – 1) / 2 operations.