**ML Functions**

(\*PROBLEM 1 \*)

(\* FUNCTION NAME: rev \*)

(\* DESCRIPTION: reverses a List L\*)

fun rev(L, R) =

if null(L) then R

else rev(tl(L), hd(L)::R);

(\* FUNCTION NAME: StringTokenizerH \*)

(\* DESCRIPTION: Helper function for StringTokenizer\*)

fun stringTokenH(L, X, R, d) =

if null(L) then nil

else if null(tl(L)) then implode(rev(hd(L)::X, nil))::R

else if (hd(L) = d) then stringTokenH(tl(L),nil,implode(rev(X,nil))::R,d)

else stringTokenH(tl(L),hd(L)::X,R,d);

(\* FUNCTION NAME: StringTokenizer \*)

(\* DESCRIPTION: Takes a string and breaks it into multiple strings

specified by delimiter d\*)

fun stringTokenizer(word, d) =

rev(stringTokenH(explode(word),nil,nil,d),nil);

(\*PROBLEM 2\*)

(\* FUNCTION NAME: CheckString\*)

(\* DESCRIPTION: Checks subString against the remaining string for a match \*)

fun checkString(subs, word) =

if null(subs) then true

else if hd(subs) = hd(word) then

checkString(tl(subs), tl(word))

else false;

(\* FUNCTION NAME: subStringH \*)

(\* DESCRIPTION: helper for subString\*)

fun subStringH(subs, word, i) =

if null(word) then ~1

else if checkString(subs, word) then i

else subStringH(subs, tl(word), i+1);

(\* FUNCTION NAME: subString \*)

(\* DESCRIPTION: checks if subStrings subs is in string word and returns the

index of the begging of the found subString or else ~1\*)

fun subString(subs, word) =

subStringH(explode(subs), explode(word), 0);

(\*PROBLEM 3\*)

(\* FUNCTION NAME: pTriangleH \*)

(\* DESCRIPTION: builds a level pascal's triangle\*)

fun pTriangleH(n,numer,denom) =

if (denom = 1) then 1 :: pTriangleH(n,numer,denom+1)

else if (n = numer) then n :: pTriangleH(n,numer -1, denom)

else if (numer = 1) then [1]

else ((n\*numer)div denom)::pTriangleH(((n\*numer)div denom), numer-1, denom +1);

(\* FUNCTION NAME: pTadder \*)

(\* DESCRIPTION: helper for pTrianlge, stacks levels of pascal's triangle\*)

fun pTadder(n,x) =

if x = 1 then [1, 1] :: pTadder(n, 2)

else if x = 0 then [1] :: pTadder(n, 1)

else if x > n then nil

else if x > 1 then pTriangleH(x,x,1) :: pTadder(n, x+1)

else nil;

(\* FUNCTION NAME: pTriangle \*)

(\* DESCRIPTION: prints out a list of lists of pascal triangle up to level n\*)

fun pTriangle(n) =

pTadder(n,0);

(\*PROBLEM 4\*)

(\*FUNCTION NAME:occrH\*)

(\*DESCRIPTION: Helper for occr, returns a list tuples of occurances of an element

with first int tuple the element and the second the numer of occurances\*)

fun occrH(a, L) =

if null(L) then [(a,1)] else

if a = (#1(hd(L))) then (#1(hd(L)),(#2(hd(L)))+1) :: tl(L)

else hd(L) :: occrH(a,tl(L));

(\*FUNCTION NAME:occr\*)

(\*DESCRIPTION: occr, takes a list and returns a list of tuples of occurances of an element\*)

fun occr(L) =

if null(L) then nil else

occrH(hd(L), occr(tl(L)));

(\*FUNCTION NAME:modeDupCheck\*)

(\*DESCRIPTION: checks for duplicate frequencies and adds them to the list\*)

fun modeDupCheck(L : (''a \* int) list, R : (''a \* int) list) =

if null(R) then L

else if (#2(hd(L)) = #2(hd(R)) andalso #1(hd(L)) <> #1(hd(R))) then hd(R) :: modeDupCheck(L, tl(R))

else modeDupCheck(L, tl(R));

(\*FUNCTION NAME:modeLH\*)

(\*DESCRIPTION: returns the a tuple list containing the most frequent element and

it frequency \*)

fun modeLH(L : (''a \* int) list, n : (''a \* int)) =

if null(L) then [n]

else if #2(hd(L)) > #2(n) then modeLH(tl(L), hd(L))

else modeLH(tl(L), n);

(\*FUNCTION NAME: modeL\*)

(\*DESCRIPTION: returns a list of tuples containing (element, # of occurences) if that

element is the mode\*)

fun modeL(L) =

modeDupCheck(modeLH(occr(L), (0,~1)), occr(L));

(\*PROBLEM 5\*)

(\* FUNCTION NAME: frontall \*)

(\* DESCRIPTION: adds a to the front of a list\*)

fun frontall(a, nil) = nil

| frontall(a, x::xs) = (a::x)::frontall(a, xs);

(\* FUNCTION NAME: inseach \*)

(\* DESCRIPTION: adds a to each postion in a list\*)

fun inseach(a, nil) = [[a]]

| inseach(a, x::xs) = (a::x::xs)::frontall(x,inseach(a, xs));

(\* FUNCTION NAME: putInAll \*)

(\* DESCRIPTION: interleaves a with each list in R\*)

fun putInAll (a, R) =

if null(R) then nil

else inseach(a, hd(R)) @ putInAll(a, tl(R));

(\* FUNCTION NAME: perm \*)

(\* DESCRIPTION: interleaves each element in L with all the resulting lists of

previous interleaves\*)

fun perm(L, R ) =

if null(L) then R

else if null(R) then perm(tl(L), [[hd(L)]])

else perm(tl(L), putInAll(hd(L), R));

(\* FUNCTION NAME: popList \*)

(\* DESCRIPTION: returns a list of numbers from i to n \*)

fun popList(n, i) =

if i > n then nil

else i :: popList(n, i+1);

(\* FUNCTION NAME: permu \*)

(\* DESCRIPTION: returns a list of lists, each list a permutation of [1,2,3,4] \*)

fun permu(n) =

perm(popList(n,1),nil);

(\*PROBLEM 6\*)

(\* FUNCTION NAME: delete \*)

(\* DESCRIPTION: deletes and element x from the list, and returns the new list \*)

fun delete(x : int list, n : int) =

if null(tl(x)) then nil

else if null(x) then nil

else if hd(x) = n then tl(x)

else if hd(tl(x)) = n then hd x :: tl(tl(x))

else hd(x) :: delete(tl(x), n);

(\* FUNCTION NAME: ssComp \*)

(\* DESCRIPTION: compares n to each element of the list behind it,

returns smallest element\*)

fun ssComp(n, L) =

if null(L) then n

else if n <= hd(L) then ssComp(n, tl(L))

else ssComp(hd(L),tl(L));

(\* FUNCTION NAME: ssHelper \*)

(\* DESCRIPTION: helper for ssort takes smallest element from L and adds it

to S until sorted\*)

fun ssHelper(S, L) =

if null(L) then S

else if null(S) then ssComp(hd(L), L) :: ssHelper(S, delete(L, ssComp(hd(L), L) ))

else ssComp(hd(L), L) :: ssHelper(S, delete(L, ssComp(hd(L), L) ))

(\* FUNCTION NAME: ssort \*)

(\* DESCRIPTION: sorts and returns list L \*)

fun ssort(L) =

ssHelper(nil, L);

(\*PROBLEM 7\*)

(\* FUNCTION NAME: filter \*)

(\* DESCRIPTION: a generic filter that takes a two parameter predicate \*)

fun filter(P, nil) = nil

|filter(P, x::xs) =

if P(x,2) then x::filter(P, xs)

else filter(P, xs);

(\* FUNCTION NAME: popList \*)

(\* DESCRIPTION: returns a list of numbers from 2 to n \*)

fun popList(n, i) =

if i > n then nil

else i :: popList(n, i+1);

(\* FUNCTION NAME: primeTestH \*)

(\* DESCRIPTION: compares n to every number 2 to n-1 to see if its prime \*)

fun primeTestH(n,x) =

if x = n then true

else if (n mod x) = 0 then false

else primeTestH(n, x+1);

(\* FUNCTION NAME: plist \*)

(\* DESCRIPTION: returns a list of prime numbers up to n \*)

fun plist(n) =

if n < 2 then nil

else filter(primeTestH, popList(n, 2));