Recursive Algorithms

Edited by Neng-Fa Zhou

Recursive Functions on Integers (given pred and succ)

```
add(0,Y) = Y.
add(X,Y) = succ(add(pred(X),Y)).

sub(X,0) = X.
sub(X,Y) = sub(pred(X),pred(Y)).

mul(0,_) = 0.
mul(X,Y) = add(mul(pred(X),Y),Y).

% name it my_div, because div/2 is a built-in in Picat
my_div(X,Y) = 0, lt(X,Y) => true.
my_div(X,Y) = succ(my_div(sub(X,Y),Y)).

% rem/2 is a built-in function
my_rem(X,Y) = X, lt(X,Y) => true.   % if X < Y
my_rem(X,Y) = my_rem(sub(X,Y),Y).

exp(_,0) = succ(0).
exp(X,Y) = mul(X,exp(X,pred(Y))).

% factorial/1 is a built-in function
fact(0) = 1.
fact(X) = mul(X,fact(pred(X))).

% gcd/2 is a built-in function
my_gcd(0,Y) = Y.
my_gcd(X,0) = X.
my_gcd(X,Y) = my_gcd(Y,rem(X,Y)).

fib(1) = succ(0).
fib(2) = succ(0).
fib(X) = add(fib(pred(X)),fib(pred(pred(X)))).
```

Recursive Functions on Lists

```
my_last([X]) = X.
my_last([_|T]) = my_last(T).

contains([E|_],E) => true.
contains([_|T],E) => contains(T,E).

find_first_of(L,E) = find_first_of(L,E,1).

find_first_of([E|_],E,I) = I.
find_first_of([_|T],E,I) = find_first_of(T,E,I+1).
find_first_of([],_E,_I) = -1.

find_last_of(L,E) = find_last_of(L,E,1,-1).

find_last_of([],_E,_I,PreI) = PreI.
find_last_of([E|T],E,I,_PreI) = find_last_of(T,E,I+1,I).
find_last_of([_|T],E,I,PreI) = find_last_of(T,E,I+1,PreI).
```

```
% the kth_elm(L,K) is the same as L[K]
kth_elm([E|_],1) = E.
kth_elm([_|T],K) = kth_elm(T,K-1).

my_len([]) = 0.
my_len([_|T]) = my_len(T)+1.

my_reverse([]) = [].
my_reverse([H|T]) = my_reverse(T) ++ [H].

power_set([]) = [[]].
power_set([H|T]) = P1++P2 =>
   P1 = power_set(T),
   P2 = [[H|S] : S in P1].

perm([]) = [[]].
perm(Lst) = [[E|P] : E in Lst, P in perm(Lst.delete(E))].
```

Merge Sort and Quick Sort

```
merge_sort([]) = [].
merge_sort([X]) = [X].
merge_sort(L) = SL => split(L,L1,L2), SL = merge(merge_sort(L1),merge_sort(L2)).

split([X,Y|Zs],L1,L2) => L1=[X|LL1], L2=[Y|LL2], split(Zs,LL1,LL2).
split(Zs,L1,L2) => L1 = Zs, L2 = [].

merge([],Ys) = Ys.
merge(Xs,[]) = Xs.
merge([X|Xs],Ys@[Y|_]) = [X|Zs], X<Y => Zs = merge(Xs,Ys).  % Ys@[Y|_] is an as-pattern
merge(Xs,[Y|Ys]) = [Y|Zs] => Zs = merge(Xs,Ys).

qsort([]) = [].
qsort([H|T]) = qsort([E : E in T, E=<H])++[H]++qsort([E : E in T, E>H]).
```

Recursive Functions on Binary Trees

```
is_btree({}) => true.
is_btree({_Val,Left,Right}) =>
   is_btree(Left),
   is_btree(Right).

inorder({}) = [].
inorder({Val,Left,Right}) =
   inorder(Left) ++ [Val] ++
   inorder(Right).

lookup_bstree({Elm,_,_},Elm) => true.
lookup_bstree({Val,Left,_},Elm), Elm < Val =>
   lookup_bstree(Left,Elm).
lookup_bstree({_,_,Right},Elm) =>
   lookup_bstree(Right,Elm).
```

```
is_bstree({}) => true.
is_bstree(Tree) =>
   Min = left_most(Tree),
   Max = right_most(Tree),
   is_bstree(Tree,Min,Max).

is_bstree({},_Min,_Max) => true.
is_bstree({Val,Left,Right},Min,Max) =>
   Min =< Val,
   Val =< Max,
   is_bstree(Left,Min,Val),
   is_bstree(Right,Val,Max).

left_most({}) = _ => throw(empty_tree).
left_most({Val,{},_Right}) = Val.
left_most({_,Left,_Right}) = left_most(Left).
```

Exercises (C++ or Java)

1. Write a function, displayInBase(n,b), that prints a decimal integer n in base b.

2. Write a function, printIntegers(n), that prints n such that groups of three (counting from the right) are separated by commas. For example, when n=12345, the output should be 12,345.

3. Write a function that takes an array (unsorted) and an integer k that returns the kth largest element in the array.

4. The following implementation of exp(X,Y) is not efficient:

   exp(_,0) = succ(0).
   exp(X,Y) = mul(X,exp(X,pred(Y))).

   Design and implement a more efficient algorithm for the function.