**Homework on SML and PROLOG**

To do alone or in a team of 2 students.

**SML**

1. What are the types of the following expressions?

[(1,5), (2,3), (5,6)]; (int \* int) list

fun f(x:real) = true; real -> bool

map f; real list -> bool list

1. Provide expressions of the following types:

int \* bool (7,false)

int list \* bool ([6,7,8], false);

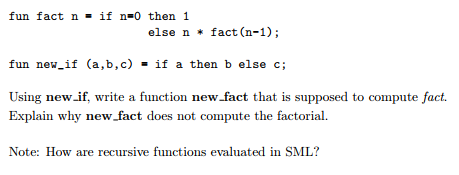
int \* real -> bool list fun f(x:int, y:real) = [false, true];

1. Write the following SML functions:



fun Fac(n) = if n = 0 then 1

else 2\*Fac(n-1);

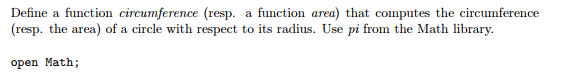


fun new\_if(a,b,c) = if a then b else c;

fun new\_fact(n) = new\_if(n = 0, 1, n\*new\_fact(n-1));

new\_fact(4);

The arguments are not evaluated in the right order so the function does not compute factorial.



open Math;

fun circumference(r:real) = 2.0\*pi\*r;

circumference(6.0);

fun area(r) = pi\*r\*r;

area(6.0);

val circumference = fn : real -> real

val it = 37.6991118431 : real

val area = fn : real -> real

val it = 113.097335529 : real

How to use map to add 3 to each elements of a list?

map (fn n => n + 3) [4,5,6];

val it = [7,8,9] : int list



fun move(n) = tl n @ [hd n];

move[5,6,7,8,9];

val move = fn : 'a list -> 'a list

val it = [6,7,8,9,5] : int list

1. Implement the datatype BinaryTree and all the functions that are provided in the lecture notes: lookup, inorder, preorder, postorde, left\_subtree, right\_subtree and label. Provide screenshots to show that your code is correct. Provide 2 tests for each function.

datatype 'a BinaryTree = btempty | bt of 'a \* 'a BinaryTree \* 'a BinaryTree;

fun inorder (btempty) = [] | inorder(bt(root:'a, left, right)) =

inorder(left) @ (root :: inorder(right));

fun preorder (btempty) = [] | preorder(bt(root:'a, left, right)) =

root :: (preorder(left) @ preorder(right));

fun postorder (btempty) = [] | postorder(bt(root:'a, left, right)) =

(postorder(left) @ postorder(right)) @ (root :: []);

fun lookup (btempty, \_) = false | lookup(bt(root:int,left,right), x:int) =

if (x = root) then true

else (if (x <= root) then lookup(left,x)

else lookup(right,x));

fun left\_subtree btempty = btempty | left\_subtree(bt(\_,left,\_)) = left;

fun right\_subtree btempty = btempty | right\_subtree(bt(\_,\_,right)) = right;

exception label\_has\_nil\_argument;

fun label btempty = raise label\_has\_nil\_argument | label(bt(value,\_,\_)) = value;

val Tree = bt(2,btempty, bt(3,btempty, bt(7,bt(6,bt(5,btempty,btempty),btempty),

bt(8,btempty,btempty))));

val Tree1 = bt(3,btempty,btempty);

val Tree2 = bt(5,bt(1,btempty,btempty),btempty);

val Tree3 = bt(7,bt(4,btempty,btempty),

bt(12,btempty,btempty));

val Tree4 = bt("\*",

bt("/",

bt("-",bt("7",btempty,btempty),

bt("a",btempty,btempty) ),

bt("5",btempty,btempty) ),

bt("exp",

bt("+",bt("a",btempty,btempty),

bt("b",btempty,btempty) ),

bt("3",btempty,btempty) ));

val Expression = bt("+",

bt("\*",

bt("+",

bt("2",btempty,btempty),

bt("5",btempty,btempty) ),

bt("\*",

bt("3",btempty,btempty),

bt("4",btempty,btempty) ) ),

bt("\*",

bt("1",btempty,btempty),

bt("6",btempty,btempty) ) );

lookup(Tree,6);

lookup(Tree,1);

lookup(Tree,8);

lookup(Tree,9);

lookup(btempty,6);

inorder(Tree);

inorder(Tree4);

preorder(Tree);

preorder(Tree4);

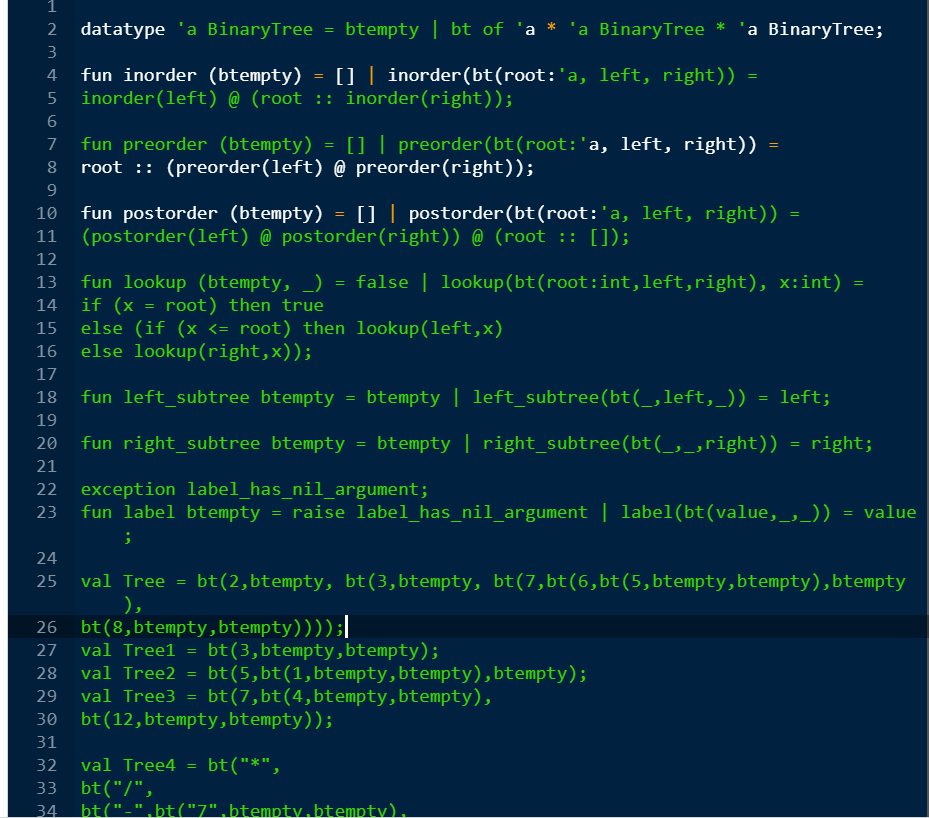
postorder(Tree);

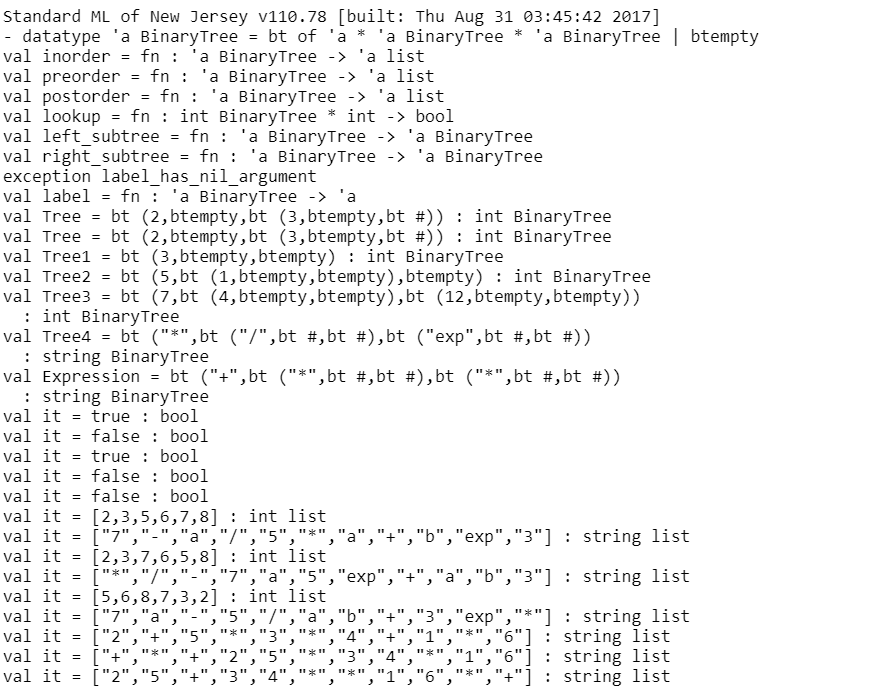
postorder(Tree4);

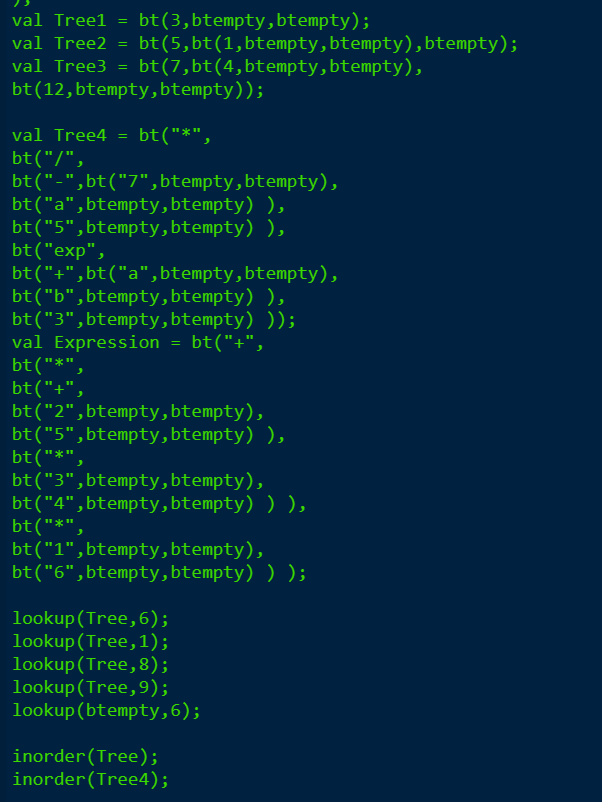
inorder(Expression);

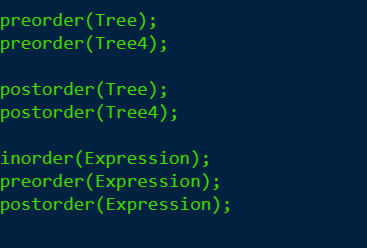
preorder(Expression);

postorder(Expression);









Prolog

1. Let us consider the following set of facts that describe the mother predicate.

mother(linda, paul).

mother(cathy, andrew).

mother(cathy, laura)

* Define a predicate female(X) which holds iff X is a female

female(linda).

female(cathy).

female(laura).

parent(cathy,andrew).

parent(cathy,laura).

parent(linda,paul).

* Define a predicate sister(X,Y) which holds iff X and Y are sisters

sibling(X, Y) :- parent(Z, X),parent(Z, Y), X \= Y.

sister(X, Y) :- sibling(X, Y),female(X), X \= Y.

* Implement female and sister in PROLOG
* Provide screenshots



1. Implement the function g such that g(x) = x+5.

g(X,Y) :- Y is X + 5.

g(2,Y).

Y=7