## 前期准备

(\*树类型定义\*)

datatype tree = Empty | Node of tree \* int \* tree;

(\*插入x到有序树中，小于插入左子树，等于或大于插入右子树\*)

**fun** Ins(x, Empty) = Node(Empty, x, Empty)

    | Ins(x, Node(t1, y, t2)) =

        case Int.compare(x,y) of

        LESS => Node(Ins(x, t1), y, t2)

        | \_ => Node(t1, y, Ins(x, t2));

(\*树的中序遍历\*)

**fun** trav Empty = []

    | trav(Node(t1,x,t2)) = trav t1 @ (x :: trav t2);

## Mission 1

**1）**

(\* int list -> int list \*)

(\* 表参数的逆序输出 \*)

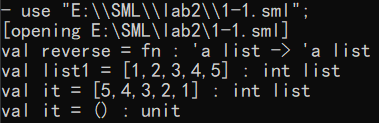
**fun** reverse [] = []

    | reverse(x::L) = reverse L @ [x];

(\* 测试 \*)

**val** list1 = [1,2,3,4,5];

reverse list1;

****

**2）**

(\* int list \* int list -> int list \*)

(\* 辅助函数，revhelp(L,M)的结果是将L的逆序存入M并在L为空时返回M \*)

**fun** revhelp ([],M) = M

    | revhelp (x::L,M) = revhelp (L,x::M);

(\* int list -> int list \*)

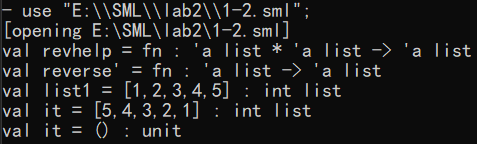
(\* 表参数的逆序输出 \*)

**fun** reverse' L = revhelp (L,[]);

(\* 测试 \*)

**val** list1 = [1,2,3,4,5];

reverse' list1;

****

## Mission 2

(\* int list \* int list -> int list \*)

(\* 两个list合并，且交替出现 \*)

**fun** interleave(l1,[]) = l1

    | interleave([],l2) = l2

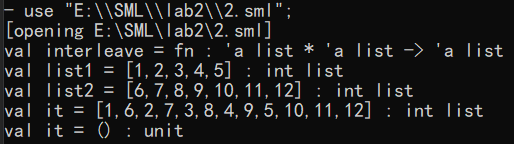
    | interleave(x::l1,y::l2) = x::y::interleave(l1,l2);

(\* 测试 \*)

**val** list1 = [1,2,3,4,5];

**val** list2 = [6,7,8,9,10,11,12];

interleave(list1,list2);

****

## Mission 3

(\* int list -> int list \* int \* int list \*)

(\* 将list拆分成长度相差小于1的两个list \*)

**fun** split [] = ([],0,[])

    | split [x] = ([],x,[])

    | split (x::L) =

        let **val** (A,y,B) = split L

        in

        if length(A) > length(B) then (A, x, y::B)

        else (y::A, x, B)

        end;

(\* int list -> tree \*)

(\* 将一个表转为平衡树 \*)

**fun** listToTree [] = Empty

    | listToTree [x] = Node(Empty,x,Empty)

    | listToTree L =

        let **val** (A,y,B) = split L

        in Node(listToTree A, y, listToTree B)

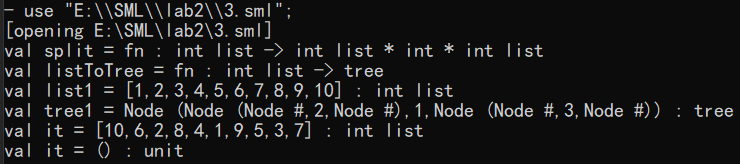
        end;

(\* 测试 \*)

**val** list1 = [1,2,3,4,5,6,7,8,9,10];

**val** tree1 = listToTree list1;

trav tree1;

****

## Mission 4

(\* tree -> tree \*)

(\* 对树进行反转 \*)

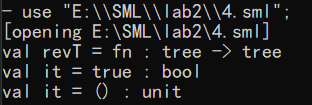
**fun** revT Empty = Empty

| revT(Node(t1,x,t2)) =

Node(revT t2, x, revT t1);

(\* 测试，若相等则返回true \*)

trav(revT tree1) = reverse(trav tree1);

****

**性能分析：假设树的总节点数为n，则work(n) = O(n)。而span(n)和树的高度有关，最坏情况下树的n个节点呈线性排列，则span(n) = O(n)，最好情况下树为完全二叉树，则span(n) = O(log2n)。**

## Mission 5

(\* tree \* int -> bool \*)

(\* 要求输入树为有序树，若树中包含值为输入数的节点，则返回true，否则返回false \*)

**fun** binarySearch (Empty,\_) = false

    | binarySearch (Node(t1,x,t2),input) =

        case Int.compare(input,x) of

           LESS => binarySearch(t1,input)

         | EQUAL => true

         | GREATER => binarySearch(t2,input);

(\* 测试 \*)

(\* 构造一个有序树 \*)

**fun** createSortedTree [] = Empty

    | createSortedTree (x::L) =

      Ins(x,createSortedTree(L));

**val** sortedTree =

createSortedTree [9,8,7,6,5,4,3,2,1];

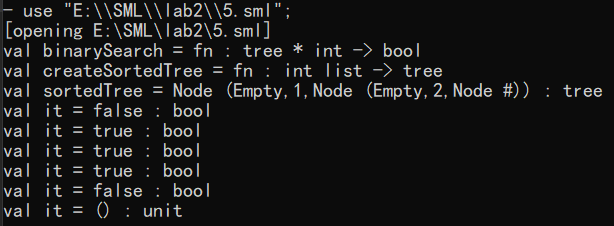
binarySearch(sortedTree,0);   (\*false\*)

binarySearch(sortedTree,2);   (\*true\*)

binarySearch(sortedTree,5);   (\*true\*)

binarySearch(sortedTree,7);   (\*true\*)

binarySearch(sortedTree,10);  (\*false\*)

****

**CS1701-熊逸钦-U201714501**