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;
```

```
- use "E:\\SML\\lab2\\1-1.sml";
[opening E:\SML\lab2\1-1.sml]
val reverse = fn : 'a list -> 'a list
val list1 = [1,2,3,4,5] : int list
val it = [5,4,3,2,1] : int list
val it = () : unit
```

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;
```

```
- use "E:\\SML\\lab2\\1-2.sml";
[opening E:\SML\lab2\1-2.sml]
val revhelp = fn : 'a list * 'a list -> 'a list
val reverse' = fn : 'a list -> 'a list
val list1 = [1,2,3,4,5] : int list
val it = [5,4,3,2,1] : int list
val it = () : unit
```

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);
```

```
- use "E:\\SML\\lab2\\2.sml";
[opening E:\SML\lab2\2.sml]
val interleave = fn : 'a list * 'a list -> 'a list
val list1 = [1,2,3,4,5] : int list
val list2 = [6,7,8,9,10,11,12] : int list
val it = [1,6,2,7,3,8,4,9,5,10,11,12] : int list
val it = () : unit
```

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;
```

```
- use "E:\\SML\\lab2\\3.sml";
[opening E:\SML\lab2\3.sml]
val split = fn : int list -> int list * int * int list
val listToTree = fn : int list -> tree
val list1 = [1,2,3,4,5,6,7,8,9,10] : int list
val tree1 = Node (Node (Node #,2,Node #),1,Node (Node #,3,Node #)) : tree
val it = [10,6,2,8,4,1,9,5,3,7] : int list
val it = () : unit
```

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);
```

```
- use "E:\\SML\\lab2\\4.sml";
[opening E:\SML\lab2\4.sml]
val revT = fn : tree -> tree
val it = true : bool
val it = () : unit
```

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) = $O(\log_2 n)$。

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*)
```

```
- use "E:\\SML\\lab2\\5.sml";
[opening E:\SML\lab2\5.sml]
val binarySearch = fn : tree * int -> bool
val createSortedTree = fn : int list -> tree
val sortedTree = Node (Empty,1,Node (Empty,2,Node #)) : tree
val it = false : bool
val it = true : bool
val it = true : bool
val it = true : bool
val it = false : bool
val it = () : unit
```

CS1701-熊逸钦-U201714501