Root

最简单的递归链表

- size用的递归方法

```java
public class IntList {
    public int first; // 当前节点存的整数
    public IntList rest; // 指向下一个节点（null 表示链表结束）


    public IntList(int f, IntList r) {
        first = f;
        rest = r;
    }
}


public int size() {
    if (rest == null) { // 递归终止条件：如果 rest 是null，说明是最后一个节点
        return 1;
    }
    return 1 + this.rest.size(); // 递归调用 size() 计算剩下的长度
}
```

头插法和头节点

```java
public class SLList {
   public IntNode first;

   public SLList(int x) {
      first = new IntNode(x, null);
   }

   public void addFirst(int x) {
       first = new IntNode(x, first);
   }

   public int getFirst() {
     return first.item;
```

优化:头插法判空

在 while 循环之前，先检查 first 是否为空：

```java
private IntNode first;
   private int size;

   public SLList() {
      first = null;
      size = 0;
   }


public void addLast(int x) {
    size += 1;


    if (first == null) { // 如果链表是空的，直接创建一个新节点
        first = new IntNode(x, null);
        return;
    }
```

dummy 不用判空

```java
public class SLLIst {
    private IntNode dummy; // 哨兵节点
    private int size;


    public SLLIst() {
        dummy = new IntNode(-1, null); 点
        size = 0;
    }


    public void addLast(int x) {
        size += 1;


        IntNode p = dummy; // 遍历从哨兵
```