Lists

1. IntL

下面我 现一个最简单的版本：

```
1  pu
2
3
4
5
6
7
8
9  }
```

这样- 或一个拥有5，10，15三个整数的链表，不得不这么做：

```
1  In
2  L.
3  L.
```

或者

```
1  In
2  L = new IntList(10, L);
3  L
```

尽管看 们还是要为它添加几个方法，首先是size方法：

```
1  /*
2  pu
3
4
5
6
7  }
8
9  /*
10 pu
11
12
13
14
15
16
17
18 }
```

... recursion! */

no recursion! */

随后的 中呈现，先是讲解方法分为Destructive和Non-Destructive，也就是是否改变参数的值。然后讲义一个链表 uctive版本、Non-Destructive迭代版本和Non-Destructive递归版本(实现代码)，然后让学生自己实的Destruc 的实现：

```
1
```

```
5
6       public static IntList dcatenate(IntList A, IntList B) {
7           //TODO:  fill in method
8           IntList res = A;
9           while (A.rest != null) {
10              A = A.rest;
11          }
12          A.rest = B;
13          return res;
14      }
15
16      /**
17       * Returns a list consisting of the elements of A followed by the
18       * * elements of B.  May NOT modify items of A.  Use 'new'.
19       */
20      public static IntList catenate(IntList A, IntList B) {
21          //TODO:  fill in method
22          IntList res = new IntList(A.first, null);
23          IntList ptr = res;
24          A = A.rest;
25          while (A != null) {
26              ptr.rest = new IntList(A.first, null);
27              ptr = ptr.rest;
28              A = A.rest;
29          }
30          ptr.rest = B;
31          return res;
32      }
```

2. SLLists

理论上讲，IntList可以实现一个链表能够做的所有事，不过它难以使用，代码难以理解。最主要的是，如果一个程序员想使用它，就必须了解它复杂想，要手动写出next的指向，这会大大降低这种数据结构的使用率。

所以，在原有IntList的基础上，我们设计一种新的类SLLists(Single Linked Lists)，并为它提供一系列的方法。

首先我们创建一个新的类IntNode：

```
1  public class IntNode {
2      public int item;
3      public IntNode next;
4
5      public IntNode(int i, IntNode n) {
6          item = i;
7          next = n;
8      }
9  }
```

除了名字不同以外，它好像和IntList没有任何区别，没错，我们还需要创建另外一个类SLList：

```
1  public class SLList {
2      public IntNode first;
3
4      public SLList(int x) {
5      first = new IntNode(x, null);
6      }
7  }
```

它将IntNode进行了封装，会使用者提供了更简洁的使用方法，当使用IntList和SLList时，你可以清楚的看出SLList优秀的地方：

```
1  IntList L1 = new IntList(5, null);
2  SLList L2  = new SLList(5);
```

SLList隐藏了IntNode中靠next取得的联系，使用者不必关心SLList内部如何实现，只知道如何用它来存储数据就足够了。我们再为SLList添加几个方

```
1  public class SLList {
2      public IntNode first;
3
4      public SLList(int x) {
```

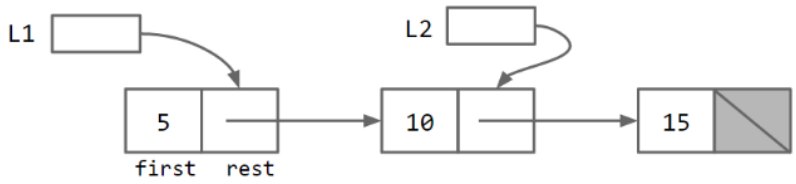
```
        first = new IntNode(x, null);
    }

    /** Adds an item to the front of the list. */
    public void addFirst(int x) {
        first = new IntNode(x, first);
    }

    /** Retrieves the front item from the list. */
    public int getFirst() {
        return first.item;
    }
}
```

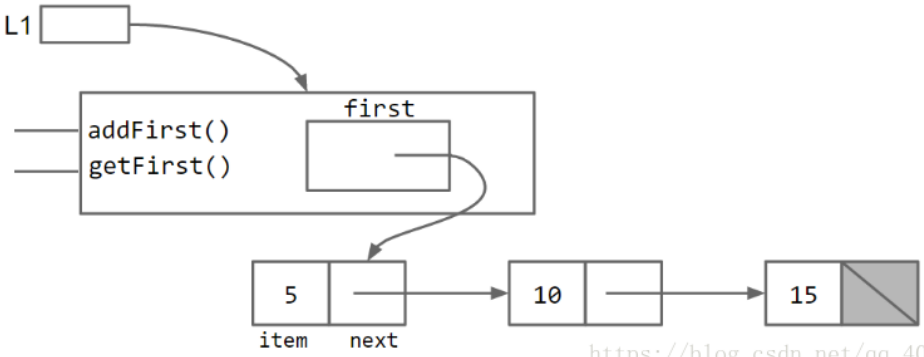
通过比较，你会发现SLList使用起来更简单了：

```
SLList L = new SLList(15);
L.addFirst(10);
L.addFirst(5);
int x = L.getFirst();
```

```
IntList L = new IntList(15, null);
L = new IntList(10, L);
L = new IntList(5, L);
int x = L.first;
```

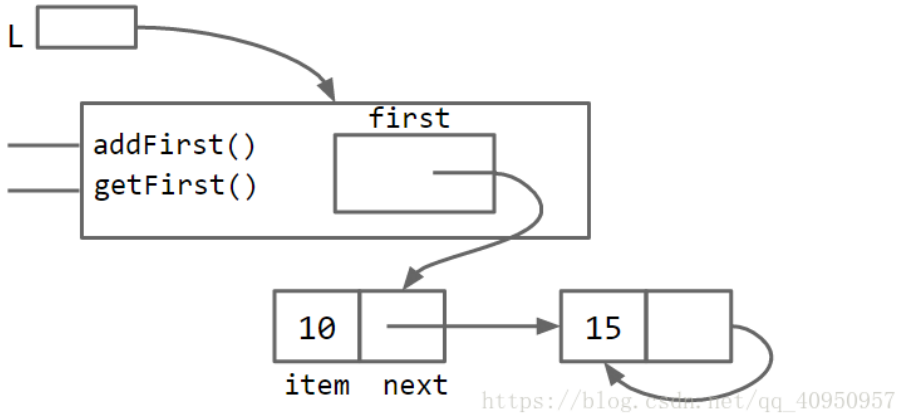
再来比较一下它们的内部实现：





不过，如果使用者这样使用呢?

```
SLList L = new SLList(15);
L.addFirst(10);
L.first.next.next = L.first.next;
```



这将会导致死循环，所以我们要把first声明为私有变量，禁止使用者调用它，这样，上面的操作将会报错。不过这时，我们有两个 `.java` 文件，使用都要导入这两个文件，所以，我们可以 将IntNode写入SLList类 中，这也称为Nested Classes。并且IntNode不会调用它以外的SLList中的值，所以我们 static类型 ，实现如下：

```
public class SLList {
       public static class IntNode {
            public int item;
            public IntNode next;
            public IntNode(int i, IntNode n) {
                item = i;
                next = n;
            }
       }

       private IntNode first;
       ...
```

我们再为它添加 `size` 方法：

```
/** Returns the size of the list starting at IntNode p. */
private static int size(IntNode p) {
    if (p.next == null) {
        return 1;
    }

    return 1 + size(p.next);
}

public int size() {
    return size(first);
}
```

然而，size方法内部使用了递归或者迭代，如果长度为1000的链表的size需要耗时2秒，那么计算长度为1,000,000的链表的size就可能会耗时2000秒 要将size方法的时间复杂度设计为常数，这里用到了 储存 (caching)的思想，我们 添加一个size变量 ，并时刻跟踪改变它的值：

```
public class SLList {
    ... /* IntNode declaration omitted. */
    private IntNode first;
    private int size;

    public SLList(int x) {
        first = new IntNode(x, null);
        size = 1;
    }

    public void addFirst(int x) {
        first = new IntNode(x, first);
        size += 1;
    }
```

```
15
16      public int size() {
17          return size;
18      }
19      ...
20  }
```

这样，无论链表多么长，size方法都会很快的返回它的值。下面我们再仔细思考，如果一个链表为空，我们如何表示它？一种方法是为它设计空参

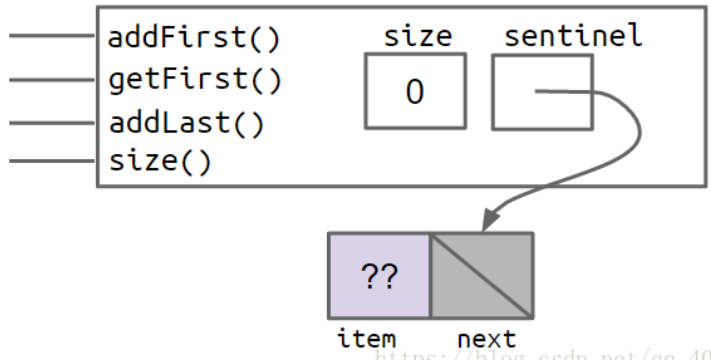
```
1  public SLList() {
2      first = null;
3      size = 0;
4  }
```

不过它的addLst方法将变得很庞大，而且之后每添加一个类似的方法，我们都需要考虑 链表为空 的情况。

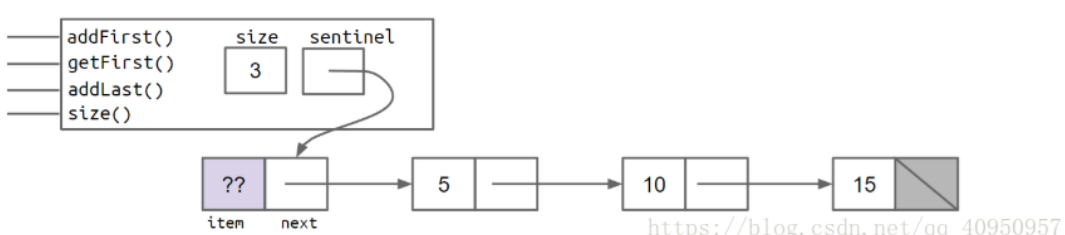
```
1  public void addLast(int x) {
2      size += 1;
3
4      if (first == null) {
5      first = new IntNode(x, null);
6          return;
7      }
8
9      IntNode p = first;
10     while (p.next != null) {
11         p = p.next;
12     }
13
14     p.next = new IntNode(x, null);
15 }
```

所以，在设计伊始，我们设计的模型要在根本上解决这个问题。这里的设计思想一开始可能会难以接受，不过渐渐你会发现它的优秀之处。我们采添加一个空的node节点，称之为sentinel node。

于是，当我们通过 `SLList L = new SLList()` 实例化一个SLList对象时，它的内部将会变成这样：



当为链表添加5，10，15三个元素时，它将变成这样：



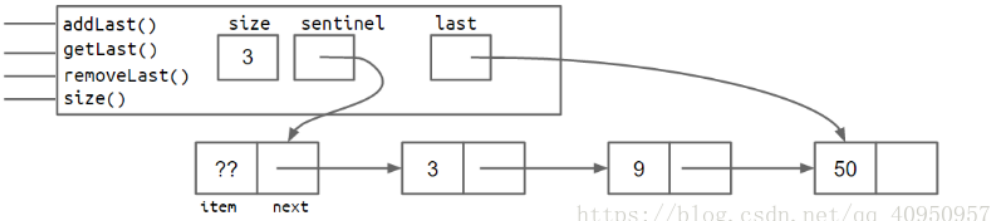
这样，addLast方法就变得简洁易读多了。

```
public void addLast(int x) {
    size += 1;
    IntNode p = sentinel;
    while (p.next != null) {
        p = p.next;
    }

    p.next = new IntNode(x, null);
}
```

但是，addLst方法同样存在一个问题，就是链表的长度越长，它的执行时间也就越长。为了解决这一问题，我们同样可以使用储存的思想，为它添量，时刻记录尾节点的位置。

```
public class SLList {
    private IntNode sentinel;
    private IntNode last;
    private int size;

    public void addLast(int x) {
        last.next = new IntNode(x, null);
        last = last.next;
        size += 1;
    }
    ...
}
```

它的内部就变成了下面这样：



再考虑，如果我们要添加一个 移除最后一个元素 的removeLast方法，该如何实现？大致的思路就是 获取到倒数第二个节点 ，然后将倒数第二个节点的null，可是， 如何获取到倒数第二个节点呢?

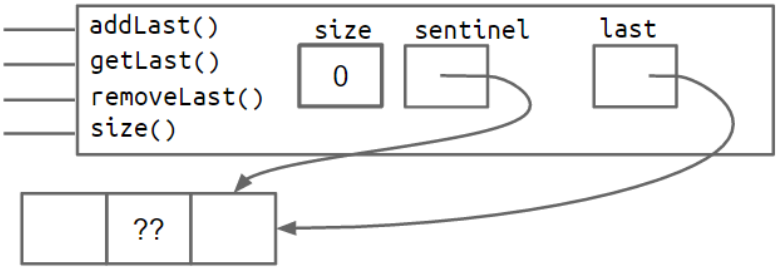
从sentinel向后遍历是不可能的了，那将使方法具有线性复杂度，其实，在现有的模型结构下很难实现，所以，我们不得在现有模型的基础上再次设数据结构，也就是下面的 DLList 。

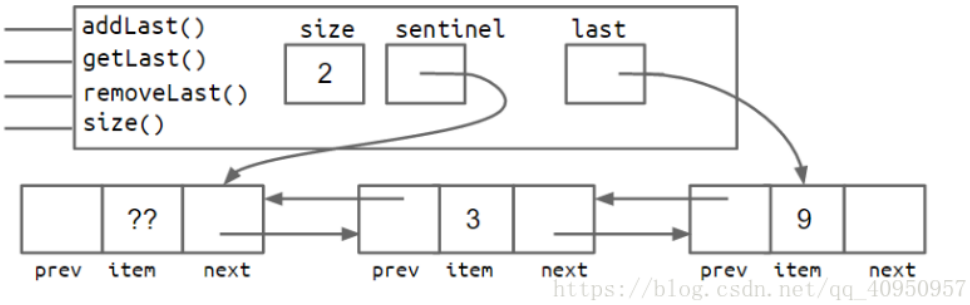
3. DLLists

这一次的改造更加彻底，我们在每一个IntNode中 添加一个prev变量 ，也就是为整个链表添加了 Back pointers ：

```
public class IntNode {
    public IntNode prev;
    public int item;
    public IntNode next;
}
```

之所以称之为DLList，其实是Doubly linked list，也就是 双向链表 ，空链表和非空链表的内部结构如下所示：

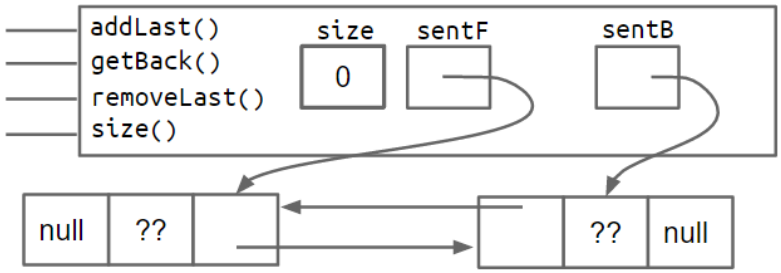


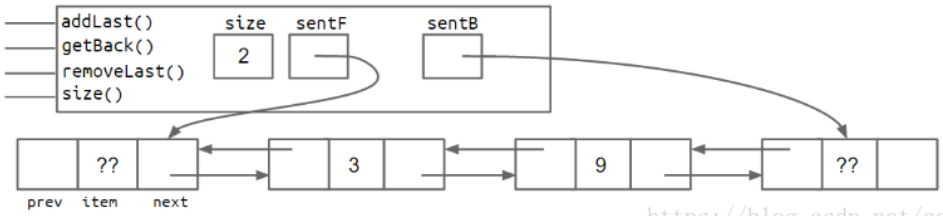


目前，我们设计的DLList已经可以在头部和尾部进行快速的添加、获取和删除操作，唯一不足的地方就是 `last` 指针有时指向带有数据的节点，有时这种特殊的空接点，在以后的实现中可能会造成混乱。

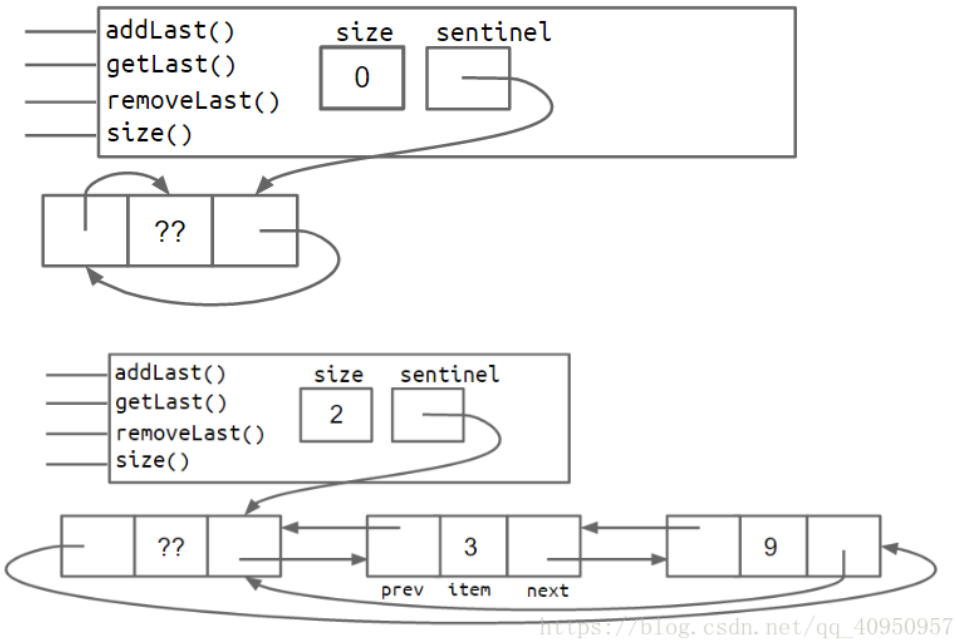
所以，我们可以用以下两种方法来避免：

- 一种是在尾部增加一个sentinel空结点。





- 一种是采用 `circle` 的的思想，让首尾指针指向同一个sentinel空结点。



这两种设计思想都很完美，在这里，cs61b让学生自己实现这种数据结构，并添加一定的方法，而且需要设计成泛型，也就是Project 1A的第一部分
我采取的第二种实现版本，这里是我的实现，上一个目录有说明文件。

至此，我们已经设计并实现了一种健壮实用易扩展的数据结构，更重要的是在设计过程中发现问题、解决问题的设计思想。 底层实现过程和设计思想
cs61b最重视的部分。

CS61B Homework6作业回顾（附代码） 92
本次作业重点是hascode和compress function，评价一个compress function好坏的标... 来自： everest115的博客

Berkeley CS61B_Data_Structures Video Lecture 1351
recorded from: http://webcast.berkeley.edu/course_details.php?seriesid=190697838... 来自： ramboisme的专栏

cs61a spring 类与继承笔记 36
继承和属性查找 来自： Siucaan

相关热词 cs61b cs61b是什么 cs61b课程 cs61b作业 cs61b字幕

cs61a spring 2018 Container笔记和补充 75
1.What is sequences? A sequence is an ordered collection of values. It is a collection ... 来自： Siucaan

cs61a 2018 spring Lab4 Lists and Data Abstraction 笔记 89
list Lists are Python data structures that can store multiple values. list comprehension... 来自： Siucaan

CS61B sp2018笔记 | Efficient Programming 12
Efficient Programming “An engineer will do for a dime what any fool will do for a dolla... 来自： 隐秀_

CS61A 系列课程笔记（一） 1338
嗯 今天刚看第二周的课程，大量的 reading 材料我是真的要崩溃了，全英。。。。我... 来自： 柠檬黄先生的博客

股市第三次机会来了！不看就亏大了！
唐煌投资 · 燃燚

CS61BHomework3作业回顾（附代码） 127
Part 1.smoosh()函数，目的是将数组中出现的所有数字产生重复的部分删除，例如11... 来自： everest115的博客

伯克利计算机低年级核心课程之CS61A-SICP 8775
http://bbs.eol.cn/forum.php?mod=viewthread&extra=page%3D1&tid=2391800 给想... 来自： fengjiexyb的专栏

伯克利大学数据结构课程 104
http://datastructur.es/sp17/ CS61B-Data Structures UC berkeley 来自： yaoxiaofeng_000...

C++实现控制台版2048（内附彩蛋） 124
前言 之前做过一个JavaScript版本的2048游戏，最近在学习C++，昨天晚上突然... 来自： 隐秀_

CS61B Homework9作业回顾（附代码） 57
本次作业主要学会运用Disjoin Sets，DisjointSets的class作业已经写好给出。class中... 来自： everest115的博客

CS61B Homework7作业回顾 (附代码) 36
发现编程真的是很强调逻辑。首先Track down the whole tree, split 3-key node into 2 ... 来自： everest115的博客

cs61a课程总结--lecture7 递归（和一种数据结构） 520
数据结构 tuple 是一种至类型 来自： vczhfan的专栏

下载 **Data Structures Lecture Notes (UCB CS61b)** 10-20

Data Structures Lecture Notes (UCB CS61b) Data Structures Lecture Notes (UCB CS61b)

Erlang 中lists的用法详解和例子说明，详细 全 3449

-。- 收集的。官方doc也不尽详细呢。。 一，带函数Pred 1, all(Pred, List) -> boolean()... 来自： Erlang技术交流

erlang lists模块函数使用大全 8038

一，带函数Pred 1, all(Pred, List) -> boolean() 如果List中的每个元素作为Pred函数的参... 来自： 持续前进

lists这个类无法使用maven打包 314

org.assertj assertj-core 来自： oppoppoppo的...

Ubuntu上更新出错的解决方法。 6399

http://ubuntuforums.org/showthread.php?p=10606904http://www.ithowto.ro/2008... 来自： forlong401的专...

Lists数组转换list 666

将数组转换为list的时候， 尽量不要用Arrays.asList，Arrays.asList()只是把数组伪装成... 来自： sunhuwh的专栏

在Ubuntu下解决E: 无法对目录 /var/lib/apt/lists/ 加锁的问题 2431

使用sudo apt-get update命令时出现如下错误： E: Could not get lock /var/lib/apt/list... 来自： BigsillyZhao的博客

Ispell 在emacs中常见问题 4360

1：M-x flyspell-word 后总是提示"Error: No word lists can be found for the language ... 来自： ryuali2010的专栏

update-manager 软件包的错误/var/lib/apt/lists的解决命令 1656

无法初始化软件包信息 update-manager软件包错误 sudo rm /var/lib/apt/lists/* -vf s... 来自： ACanoe的专栏

redis--lists类型及操作 1822

lists类型List是一个链表结构，主要功能是push、pop、获取 一个范围的所有值等等，... 来自： 杨森源的博客

下载 **A Java Reference: Assorted Java Reference Material** 10-08

UC Berkeley CS61B Data Structure推荐阅读 Fall 2018

下载 **CS61B java intro** 03-31

berkeley cs61b java introduction，对初学者很有帮助

Vue警告 4417

Vue警告component lists rendered with v-for should have explicit keys问题描述： 来自： Dear_Mr的博客

erlang list的使用与优化建议 5963

erlang有两种复合结构，tuple和list，两者的区别是tuple子元素的个数是固定不变的... 来自： 没有开花的树

Lists的使用 87

List（接口）顺序时List最重要的特性：它可保证元素按照规定的顺序排列。List为Coll... 来自： qq_37003223的...

Vuejs（一）入门 5746

Vue.js （一） 入门官方网址 ✈️第一次真正接触到 Vue.js 其实是源于一次头条中的视... 来自： 风起叶落，若是...