DeepL
订阅DeepL Pro以翻译大型文件。
欲了解更多信息，请访问www.DeepL.com/pro。

CSE 101
数据结构和算法介绍 编程作业5

在这个项目中，你将创建一个新的、有点不同的整数List ADT，这次是用C++语言。你将使用这个List来执行洗牌操作，并确定需要洗多少次才能使List回到原来的顺序。首先要仔细回顾网页上 Examples/C++ 中的 Queue 和 Stack 例子。这些例子建立了我们在C++语言中构建ADT的规范和惯例。还可以阅读*C++中的ADTs*讲义。头文件List.h也已张贴在Examples/pa5，还有一个测试客户端、一些输出文件和本项目的Makefile。

完美的摇摆舞
完美洗牌是指将一副牌平均分割，然后将每一半的牌交替插入新的牌中，合并成一副新的牌。例如，如果我们的牌包含7张牌，标记为0-6，我们将执行以下步骤。

甲板。	0 1 2 3 4 5 6
分开。	0 1 2 | 3 4 5 6
准备合并。	3 4 5 6 0 1 2
合并。	3 0 4 1 5 2 6

在新的列表上进行同样的完美洗牌操作，我们得到。1 3 5 0 2 4 6。
再一次洗牌，就得到了原来的顺序。我们说这种重新排列（或称排列）的*顺序*是3，因为在任何一副牌上应用3次就能使这副牌恢复到原来的顺序。

我们将用一个长度为n的列表来表示一副由整数（$0, 1, 2, \ldots, n\text{-}1$）组成的$n$牌。如果$n$是偶数，那么我们就可以把列表分成两半，每半长度为$n/2$。

$$(0, 1, \ldots, \frac{n}{2} - 1) \qquad (\frac{n}{2}, \frac{n}{2} + 1, \ldots, n - 1)$$

如果n是奇数，我们把多余的牌放在右半边。左半边包含$\lfloor n/2 \rfloor$，右半边则是$\lceil n/2 \rceil$.

$$(0, 1, \ldots, \lfloor \frac{n}{2} \rfloor - 1) \qquad (\lfloor \frac{n}{2} \rfloor, \lfloor \frac{n}{2} \rfloor + 1, \ldots, n - 1)$$

请注意，后一个公式在偶数和奇数情况下都是正确的。你在这个项目中的顶级客户端将用C++编写，称为Shuffle.cpp。它将包含一个具有以下原型的函数。

```
void shuffle(List& D)。
```

函数`shuffle()`将通过执行一次洗牌操作改变其`List&`(List reference)参数`D`，如上所述。函数`main()`将读取一个命令行参数，它将是一个正整数，指定一副牌中的最大牌数。

对于1到这个最大范围内的每一个n，你的程序将执行洗牌操作，直到列表（$0, 1, 2, \ldots, n - 1$）恢复到原来的顺序。

的过程中计算洗牌的次数。它将在标准输出中打印一个表格，给出每个n值的计数。下面是一个示例会话。

```
$ Shuffle 16
扑克牌              的大小和数量
-----------------------------------------------------
 1                   1
 2                   2
 3                   2
 4                   4
 5                   4
 6                   3
 7                   3
 8                   6
 9                   6
 10                  10
 11                  10
 12                  12
 13                  12
 14                  4
 15                  4
 16                  8
$
```

像往常一样，$符号代表Unix提示。如果你把你的程序输出重新导向一个文件，那么你可以通过与文件out10、out35和out75分别进行比较来验证你的格式和结果，这些文件都张贴在网页上。例如，Shuffle 35 > myout35，然后diff myout35 out35将验证你的结果，直至牌面大小为35。

列表 ADT

在这个项目中，你的大部分工作将是在C++中建立List ADT。正如你所期望的，这个实现将被分成两个文件，List.h和List.cpp。List.h是在网站上提供的，并将在提交时不作任何修改。注意，与C语言中的ADT实现不同，定义导出类型的主要类在.h文件中，而不是在.cpp文件中。因此，List.h文件包括一个名为Node（不是NodeObj）的私有内部结构，List类的字段声明，以及ADT操作的原型。

List ADT的这个化身的底层数据结构将是一个Node对象的双链列表，在前面和后面有两个假节点。List的空状态将由这两个哨兵节点来表示，它们分别作为next和prev指向对方。储存在虚拟节点的数据字段中的值可以是你喜欢的任何东西，并且不会被读出或写入。正如你可能知道的，假节点对于简化插入和删除操作中出现的特殊情况很有用。

这个 List ADT 和你在以前的作业中创建的 ADT 的一个关键区别是光标。光标不是躺在列表元素下面的水平条，而是想象成一个站在两个元素之间的垂直条，或者站在所有元素的左边或右边（在客户视图中）。事实上，元素本身是没有索引的。相反，元素之间的空间是有索引的。与C语言中的List不同，*光标将始终站在这些中间位置中的一个，并且不能成为未定义的。*因此，一个包含n元素的 List 将有精确的$n+1$ 个可能的光标位置，即 0，表示 List 的前面，到n的后面。例如，如果$n=7$，列表中有8个可用的光标位置。

```
0     1     2     3     4     5     6     7
|a | b | c | d | e | f | g |
```

为了在ADT中表示光标，我们将使用*两个*Node指针，它们在.h文件中被称为`beforeCursor`和`afterCusor`。这些指针将始终横跨垂直光标，指向光标位置之前和之后的节点对象。如果光标在列表的前面（位置0），那么 `beforeCursor` 将指向 `frontDummy`。同样的，如果光标在列表的后面（位置为 *n*），那么 `afterCursor` 将指向 `backDummy`。

所有的List操作在List.h的注释块中都有详细描述。其中一些函数的实现可能具有挑战性。研究一下网页上发布的C++中的队列和堆栈ADT的实现，将是非常值得的。例如，队列ADT中的函数`join()`与List中的`concat()`非常相似。在C++程序中，重载内置的操作符是非常常见的。在这个项目中，你将重载`operator<<()`（流插入）、`operator==()`（平等比较）和`operator=()`（分配）。如果没有一些指导，这些运算符可能很难正确使用。幸运的是，所有这些操作符都在队列和堆栈的例子中被重载。参见参考页

https://en.cppreference.com/w/cpp/language/operators

以及以下文章

https://www.cplusplus.com/articles/y8hv0pDG/

来了解一些关于C++中复制构造函数和赋值运算符之间关系的有用信息。

要交的东西

一旦清单构建完成并经过测试，构建客户端Shuffle.cpp对大多数学生来说应该没有大的困难。提交以下6个文件到你在git.ucsc.edu的pa5目录。

阅读手册	由你来写，是提交的文件的目录和对评分者的任何说明。
制作文件	提供的，按你认为合适的方式修改
`List.h`	所提供的，**不要改变**
`List.cpp`	由你来写，本作业中的大部分工作
`ListTest.cpp`	由你来写，为你的List
`Shuffle.cpp`提供一个测试线束。	由你来写

像往常一样，不要交出可执行文件、二进制文件或上面没有列出的任何东西。尽早开始，多问问题，并在到期日前提交你的项目。这里还有一些关于C++重要主题的链接。

指针与反向操作符重载标准
异常类

此外，推荐的教科书《*Data Abstraction & Problem Solving with C++*》（6th版），作者是Carrano和Henry（Pearson 2013 ISBN 10: 0-13-292372-6），在第31-37页有一个很好的说明，说明如何在C++中实现ADT，还有很多其他的例子。