云南大学数学与统计学院
上机实践报告

课程名称： 数据结构与算法实验	**年级：** 2015 级	**上机实践成绩：**
指导教师： 陆正福	**姓名：** 刘鹏	
上机实践名称： 链表与表的抽象化实验	**学号：** 20151910042	**上机实践日期：** 2017-04-10
上机实践编号： No.7	**组号：**	**上机实践时间：** 上午三、四节

一、实验目的

1. 熟悉链表以及表的抽象化等有关的概念，理解从数组到各类链表的选择；
2. 熟悉主讲教材 Chapter 7 的代码片段。

二、实验内容

1. 调试主讲教材 Chapter 7 的 Python 程序。

三、实验平台

Windows 10 Enterprise 中文版；
Python 3.6.0；
Wing IDE Professional 6.0.2-1 集成开发环境。

四、实验记录与实验结果分析

```
1   # 7.1.1 Implementing a Stack with a Singly Linked List
2
3   class LinkedStack:
4       """LIFO Stack implementation using a singly linked list for storage."""
5
6       #---------------------------- nested _Node class ----------------------------
7       class _Node:
8           """Lightweight, nonpublic class for storing a singly linked node."""
9           __slots__ = '_element','_next'         # streamline memory usage
10
11          def __init__(self,element,next):       # initialize node's fields
12              self._element = element            # reference to user's element
13              self._next = next                  # reference to next node
14
15      #------------------------------- stack methods -------------------------------
16      def __init__(self):
17          """Create an empty stack."""
18          self._head = None                      # reference to the head node
19          self._size = 0                         # number of stack elements
20
21      def __len__(self):
22          """Return the number of elements in the stack."""
23          return self._size
24
25      def is_empty(slef):
26          """Return True if the stack is empty."""
27          return self._size == 0
28
29      def push(self,e):
30          """Add element e to the top of the stack."""
31          self._head = self._Node(e,self._head)   # create and link a new node
32          self._size += 1
33
34      def top(self):
35          """Return (but not remove) the element at the top of the stack.
```

```
36
37          Raise Empty exception if the stack is empty.
38          """
39          if self.is_empty():
40              raise Empty('Stack is empty')
41          return self._head._element            # top of stack is at head of list
42
43      def pop(self):
44          """Remove and return the element from the top of the stack (i.e., LIFO).
45
46          Raise Empty exception if the stack is empty.
47          """
48          if self.is_empty():
49              raise Empty('Stack is empty')
50          answer = self._head._next
51          self._head = self._head._next          # bypass the former top node
52          self._size -= 1
53          return answer
54
55  #------------------------------ my main function ------------------------------
56  a = LinkedStack
57  for i in range(10):
58      a.push(1)
59
60  for i in range(10):
61      a.pop()
```

五、实验体会

Translation：

Chapter 7 Linked Lists

*第七章 链表

In Chapter 5 we carefully examined Python's array-based list class, and in Chapter 6 we demonstrated use of that class in implementing the classic stack, queue, and dequeue ADTs. Python's list class is highly optimized, and often a great choice for storage. With that said, there are some notable disadvantages:

*我们在第五章详细讨论了 Python 下面的基于数组的列表类，后来又在第六章里面，通过经典的栈、队列、双端队列等抽象数据类型的程序实现，展示了列表类的用途。Python 的列表类非常高效，通常是用来存储的良好选择。但是，列表类也有很多劣势，如下：

1. The length of a dynamic array might be longer than the actual number of elements that it stores.
   *动态数组所占用的实际内存可能比数组本身的理论内存要大。

2. Amortized bounds for operations may be unacceptable in real-time systems.
   *在实时系统中，均摊操作可能是不可接受的。

3. Insertions and deletions at interior positions of an array are expensive.
   *在内部位置插入或者删除一个元素的代价可能很大。

In this chapter, we introduce a data structure known as a ***linked list***, which provides an alternative to an array-based sequence (such as a Python list)。 Both array-based sequences and linked lists keep elements in a certain order, but using a very different style. An array provides the more centralized representation, with one large chunk of memory capable of accommodating references to many elements. A linked list, in contrast, relies on a more distributed representation in which a lightweight object, known as a ***node***, is allocated for each element. Each node maintains a reference to its element and one or more references to neighboring nodes in order to collectively represent the linear order of the sequence.

*我们将在这一章中，介绍一种叫做链表的数据结构，这种结构可以在某些时候替代基于数组的序列。无论是基于数组的序列还是链表，他们的元素都是按照一定的顺序进行保存的，但是两者的保存方式却大有不同。数组采用的是一种集中性很高的实现方法，那就是开辟一大块连续的内存区域用以存放元素。而链表则不然，链表采用一种分布式的实现方式，而这个方式是依靠为每一个元素分配一个叫做节点的轻量级对象而实现的。每一个节点都包含着与它的对应元素所匹配的指向，同时也包含着其相邻的节点的指向，这样一来就表现出了序列的线性结构。

We will demonstrate a trade-off of advantages and disadvantages when contrasting array-based sequences and linked lists. Elements of a linked list cannot be efficiently accessed by a numeric index *k*, and we cannot tell just by examining a node if it is the second, fifth, or twentieth node in the list. However, linked lists avoid the three disadvantages noted above for array-based sequence.

* 当对比基于数组的数列与链表的时候，我们会均衡地展示两者的优缺点。链表中的元素不能通过下标索引进行高效的访问，并且我们不能通过检查节点来判断一个节点在表中的位置。然而，链表可以很好地避免上面提到的那三个数组序列的弊端。

END

六、参考文献

[1] Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser, *Data Structures and Algorithms in Python*, Chapter 4

[2] 实验教材：汪萍，陆正福等编著 数据结构与算法的问题与实验 第 1 章