CPE111: Programming with Data Structures

Week4: Linked List

Practice I) Implement a Stack with a Singly linked list
Mission: Complete the Code in List1

Stack ADT

A *stack* is a data structure that stores a linear collection of items with access limited to a last-in first-out (LIFO) order. Adding and removing items is restricted to one end known as the *top* of the stack. An empty stack is one containing no items.

- **Stack():** Creates a new empty stack.
- **isEmpty():** Returns a boolean value indicating if the stack is empty.
- **length ():** Returns the number of items in the stack.
- **pop():** Removes and returns the top item of the stack, if the stack is not empty. Items cannot be popped from an empty stack. The next item on the stack becomes the new top item.
- **peek():** Returns a reference to the item on top of a non-empty stack without removing it. Peeking, which cannot be done on an empty stack, does not modify the stack contents.
- **push(item):** Adds the given item to the top of the stack.

List1: Stacklinkedlist.py

```
1 # Implements the Stack ADT using a singly linked list.
2
3  #Defines a private storage class for creating list nodes.
4 class ListNode( object ):
5         def __init__( self, item ) :
6                 self._item = item
7                 self._next = None
8
9 class LStack :
10        # Constructs an empty stack.
11        def __init__( self ):
12                self._head = None
13                self._size = 0
14
15        # Return True if the stack is empty or False otherwise.
16        def isEmpty(self):
17                return len( self ) == 0
18
19        # Returns the number of items in the stack.
20        def __len__( self ):
21                return self._size
22
23        # Return the top item on the stack without removing it.
24        def peek( self ):
25                assert not self.isEmpty( ),"Cannot peek at an empty stack"
```

```
26              return self._head._item
27
28      # Removes and return the top (head of the linked list) item  on the stack
29      def pop( self ):
30              assert not self.isEmpty( ),"Cannot pop from an empty stack"
31              …
32              …
33              …
34              …
35              …
36
37      # push a new item onto the top of the stack (head of the linked list).
38      def push( self, item ):
39              …
40              …
41              …
42              …
43
44      # Return the items in the Stack
45      def __repr__( self ):
46              curNode = self._head
47              s = "--\n"
48              while curNode is not None:
49                      s = s + str( curNode._item )+ "\n"
50                      curNode = curNode._next
51              s = s + "--"
52              return s
53
54      # Return the items in the Stack when using print( )
55      def __str__( self ):
56              curNode = self._head
57              s = "--\n"
58              while curNode is not None:
59                      s = s + str( curNode._item )+ "\n"
60                      curNode = curNode._next
61              s = s + "--"
63              return s
64
65      # Determines if an item is contained in the stack.
66      def isContain( self, target ):
67              curNode = self._head
68              while curNode is not None and curNode._item != target :
69                      curNode = curNode._next
70              return curNode is not None
```

CPE111: Programming with Data Structures

Practice II) Implementing a Queue with a Singly linked list
Mission: from the queue ADT, Write the code to implement it with the given singly linked list
Hint: enqueue from the tail and dequeue from the head

Queue ADT

A *queue* is a data structure that a linear collection of items in which access is restricted to a first-in first-out (FIFO) basis. New items are inserted at the back and existing items are removed from the front. The items are maintained in the order in which they are added to the structure.

- **Queue():** Creates a new empty queue, which is a queue containing no items.
- **isEmpty():** Returns a boolean value indicating whether the queue is empty.
- **length ():** Returns the number of items currently in the queue.
- **enqueue(item):** Adds the given item to the back of the queue.
- **dequeue():** Removes and returns the front item from the queue. An item cannot be dequeued from an empty queue.

List2: Fragment of code **Queuelinkedlist.py**

```
#Defines a private storage class for creating list nodes.
class ListNode( object ):
        def __init__( self, item ) :
                self._item = item
                self._next = None
class LQueue :
        # Constructs an empty queue.
        def __init__( self ):
                self._head = None
                self._tail = None
                self._size = 0
```

Practice III) Implementing a Deque with a Doubly linked list
Mission: from Double-ended queue ADT, write the code to complete the implementation of a Deque with a Doubly linked list.

Double-ended queue ADT

A **doubled-ended queue** is a queue-like data structure that supports insertion and deletion at both the front and the back of the queue.

- **Deque():** Create a new empty deque, which no item with in.
- **AddFirst(item)**: Add an item to the front of deque.
- **AddRear(item)**: Add an item to the back of deque.
- **DeleteFirst( )**: Remove and return the first item from deque; an error occurs if the deque is empty.
- **DeleteRear ( )**: Remove and return the last item from deque; an error occurs if the deque is empty.

- **First()**: Return (but do not remove) the first item of deque; an error occurs if the deque is empty.
- **Rear()**: Return (but do not remove) the last item of deque; an error occurs if the deque is empty.
- **isEmpty( )**: Return True if deque does not contain any items.
- **length( )**: Return the number of items in deque.

List3: DequeDlinkedlist.py

```
1 # Implements the Deque ADT using a doubly linked list.
2
3 # Defines a Simple DlinkNode.
4 class DlinkNode(object):
5       def __init__(self,item,prev,next):
6               self._item = item
7               self._prev = prev
8               self._next = next
9
10 class DDeque:
11      # Construct an empty Deque.
12      def __init__(self):
13              self._header = DlinkNode(None,None,None)
14              self._trailer = DlinkNode(None,None,None)
15              self._header._next = self._trailer
16              self._trailer._prev = self._header
17              self._size = 0
18
19      # Return the number of items in the deque
20      def __len__(self):
21              return self._size
22
23      # Return True if the Deque is empty or False otherwise
24      def isEmpty(self):
25              return self._size == 0
26
27      # Insert node between predecessor and successor
28      def insert_between(self,item,predecessor,successor):
29              newNode = DlinkNode(item,predecessor,successor)
30              predecessor._next = newNode
31              successor._prev = newNode
32              self._size += 1
33
34      # Delete a node
35      def delete_node(self,node):
36              predecessor = node._prev
37              successor = node._next
38              predecessor._next = successor
39              successor._prev = predecessor
40              self._size -= 1
```

```
41              item = node._item
42              node._prev = node._next = node._item = None
43              return item
44
45      def __str__(self):
46              curNode = self._header
47              s = "["
48              while curNode is not None:
49                      s = s + str(curNode._item)+ " "
50                      curNode = curNode._next
51              s = s[:-1] + "]"
52              return s
53
54      def __repr__(self):
55              curNode = self._header
56              s = "["
57              while curNode is not None:
58                      s = s + str(curNode._item)+ " "
59                      curNode = curNode._next
60              s = s[:-1] + "]"
61              return s
62
63      def First(self):
64              assert not self.isEmpty(), "Deque is empty"
65              …
66
67      def Rear(self):
68              assert not self.isEmpty(), "Deque is empty"
69              …
70
71      def AddFirst(self,item):
72              …
73
74      def AddRear(self,item):
75              …
76
77      def DeleteFirst(self):
78              assert not self.isEmpty(),"Deque is empty"
79              …
80
81      def DeleteRear(self):
82              assert not self.isEmpty(),"Deque is empty"
83              …
```

CPE111: Programming with Data Structures

Reference

Rance D. Necaise. *Data Structures and algorithms using python*. *Chapter7*. John Wiley&Sons,Inc., 2011

Michael T.Goodrich, Roberto Tamassia, Michael H. Goodwasser. *Data Structures and Algorithms in python*. *Chapter5*. John Wiley&Sons,Inc. 2013