جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

قســـــم الانـــظـــمـــة الــطبية الـــذكـــيـــة

المرحلة الثانية

Lecture: (3)

Subject: Object oriented programming II
Class: Second
Lecturers: Dr. Dunia H. Hameed , Dr. Maytham N. Meqdad

Object Oriented Programming (II) – Third Lecture

3.4 Linked List

A linked list consists of nodes, each node has a value and a pointer to the next node. The header is the first node in the linked list, and it serves as the starting point for any iteration in the list. There are three types of linked lists in python they are singly linked list, doubly linked list, and circular linked list. The following section contains various Python programs on linked lists and their operations, single and doubly linked lists, and data structures using linked lists. It also includes various programs on a linked list using recursion.

3.4.1 Singly Linked List Operations

Problem Description

The program creates a singly linked list and presents the user with a menu to perform various operations on the list.

Problem Solution

1. Create a class Node with instance variables data and next.
2. Create a class LinkedList with instance variable head.
3. The variable head points to the first element in the singly linked list.
4. Define methods get_node, get_prev_node, insert_after, insert_before, insert_at_beg, insert_at_end, remove and display.
5. get_node takes an index as argument and traverses the list from the first node that many times to return the node at that index.
6. get_prev_node takes a reference node as argument and returns the previous node. It returns None when the reference node is the first node.

7. The methods insert_after and insert_before insert a node after or before some reference node in the list.
8. The methods insert_at_beg and insert_at_end insert a node at the first or last position of the list.
9. The method remove takes a node as argument and removes it from the list.
10. The method display traverses the list from the first node and prints the data of each node.
11. Create an instance of LinkedList and present the user with a menu to perform operations on the list.

Program/Source Code

```
class Node:
    def __init__(self, data):
       self.data = data
       self.next = None


class LinkedList:
    def __init__(self):
        self.head = None

    def get_node(self, index):
        current = self.head
        for i in range(index):
            if current is None:
                return None
            current = current.next
        return current

    def get_prev_node(self, ref_node):
        current = self.head
        while (current and current.next != ref_node):
            current = current.next
        return current
```

```
def insert_after(self, ref_node, new_node):
    new_node.next = ref_node.next
    ref_node.next = new_node

def insert_before(self, ref_node, new_node):
    prev_node = self.get_prev_node(ref_node)
    self.insert_after(prev_node, new_node)

def insert_at_beg(self, new_node):
    if self.head is None:
        self.head = new_node
    else:
        new_node.next = self.head
        self.head = new_node

def insert_at_end(self, new_node):
    if self.head is None:
        self.head = new_node
    else:
        current = self.head
        while current.next is not None:
            current = current.next
        current.next = new_node
    def remove(self, node):
        prev_node = self.get_prev_node(node)
        if prev_node is None:
            self.head = self.head.next
        else:
            prev_node.next = node.next

    def display(self):
        current = self.head
        while current:
            print(current.data, end = ' ')
            current = current.next


a_llist = LinkedList()

print('Menu')
print('insert <data> after <index>')
print('insert <data> before <index>')
print('insert <data> at beg')
print('insert <data> at end')
print('remove <index>')
print('quit')
```

```
while True:
    print('The list: ', end = '')
    a_llist.display()
    print()
    do = input('What would you like to do? ').split()

    operation = do[0].strip().lower()

    if operation == 'insert':
        data = int(do[1])
        position = do[3].strip().lower()
        new_node = Node(data)
        suboperation = do[2].strip().lower()
        if suboperation == 'at':
            if position == 'beg':
                a_llist.insert_at_beg(new_node)
            elif position == 'end':
                a_llist.insert_at_end(new_node)
        else:
            index = int(position)
            ref_node = a_llist.get_node(index)
            if ref_node is None:
                print('No such index.')
                continue
            if suboperation == 'after':
                a_llist.insert_after(ref_node, new_node)
            elif suboperation == 'before':
                a_llist.insert_before(ref_node, new_node)

    elif operation == 'remove':
        index = int(do[1])
        node = a_llist.get_node(index)
        if node is None:
            print('No such index.')
            continue
        a_llist.remove(node)

    elif operation == 'quit':
        break
```

Program Explanation

1. An instance of LinkedList is created.

2. The user is presented with a menu to perform various operations on the list.

3. The corresponding methods are called to perform each operation.

Runtime Test Cases

```
Case 1:
Menu
insert <data> after <index>
insert <data> before <index>
insert <data> at beg
insert <data> at end
remove <index>
quit
The list:
What would you like to do? insert 7 at beg
The list: 7
What would you like to do? insert 3 at end
The list: 7 3
What would you like to do? insert 1 after 0
The list: 7 1 3
What would you like to do? insert 9 before 2
The list: 7 1 9 3
What would you like to do? remove 2
The list: 7 1 3
What would you like to do? insert 12 at end
The list: 7 1 3 12
What would you like to do? remove 0
The list: 1 3 12
What would you like to do? quit
```

```
Case 2:
Menu
insert <data> after <index>
insert <data> before <index>
insert <data> at beg
insert <data> at end
remove <index>
quit
The list:
What would you like to do? insert 5 after 0
No such index.
The list:
What would you like to do? insert 3 at end
The list: 3
What would you like to do? insert 1 after 0
The list: 3 1
What would you like to do? insert 2 before 1
The list: 3 2 1
What would you like to do? insert 0 at end
The list: 3 2 1 0
What would you like to do? remove 3
The list: 3 2 1
What would you like to do? remove 2
The list: 3 2
What would you like to do? quit
```

3.4.2 Circular Singly Linked List Operations

Problem Description

The program creates a circular single linked list and presents the user with a menu to perform various operations on the list.

Problem Solution

1. Create a class Node with instance variables data and next.
2. Create a class CircularLinkedList with instance variable head.
3. The variable head points to the first element in the circular single linked list.
4. Define methods get_node, get_prev_node, insert_after, insert_before, insert_at_beg, insert_at_end, remove and display.
5. get_node takes an index as argument and traverses the list from the first node that many times to return the node at that index. It stops if it reaches the first node again.
6. get_prev_node takes a reference node as argument and returns the previous node.
7. The methods insert_after and insert_before insert a node after or before some reference node in the list.
8. The methods insert_at_beg and insert_at_end insert a node at the first or last position of the list.
9. The method remove takes a node as argument and removes it from the list.
10. The method display traverses the list from the first node and prints the data of each node. It stops when it reaches the first node again.
11. Create an instance of Circular LinkedList and present the user with a menu to perform operations on the list.

Program/Source Code

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None


class CircularLinkedList:
    def __init__(self):
        self.head = None

    def get_node(self, index):
        if self.head is None:
            return None
        current = self.head
        for i in range(index):
            current = current.next
            if current == self.head:
                return None
        return current
```

```python
def get_prev_node(self, ref_node):
    if self.head is None:
        return None
    current = self.head
    while current.next != ref_node:
        current = current.next
    return current

def insert_after(self, ref_node, new_node):
    new_node.next = ref_node.next
    ref_node.next = new_node

def insert_before(self, ref_node, new_node):
    prev_node = self.get_prev_node(ref_node)
    self.insert_after(prev_node, new_node)
```

```
def insert_at_end(self, new_node):
    if self.head is None:
        self.head = new_node
        new_node.next = new_node
    else:
        self.insert_before(self.head, new_node)

def insert_at_beg(self, new_node):
    self.insert_at_end(new_node)
    self.head = new_node

def remove(self, node):
    if self.head.next == self.head:
        self.head = None
    else:
        prev_node = self.get_prev_node(node)
        prev_node.next = node.next
        if self.head == node:
            self.head = node.next
    def display(self):
        if self.head is None:
            return
        current = self.head
        while True:
            print(current.data, end = ' ')
            current = current.next
            if current == self.head:
                break


a_cllist = CircularLinkedList()

print('Menu')
print('insert <data> after <index>')
print('insert <data> before <index>')
print('insert <data> at beg')
print('insert <data> at end')
print('remove <index>')
print('quit')
while True:
    print('The list: ', end = '')
    a_cllist.display()
    print()
    do = input('What would you like to do? ').split()

    operation = do[0].strip().lower()

    if operation == 'insert':
        data = int(do[1])
        position = do[3].strip().lower()
        new_node = Node(data)
        suboperation = do[2].strip().lower()
        if suboperation == 'at':
            if position == 'beg':
                a_cllist.insert_at_beg(new_node)
            elif position == 'end':
                a_cllist.insert_at_end(new_node)
    else:
        index = int(position)
        ref_node = a_cllist.get_node(index)
        if ref_node is None:
            print('No such index.')
            continue
        if suboperation == 'after':
            a_cllist.insert_after(ref_node, new_node)
        elif suboperation == 'before':
            a_cllist.insert_before(ref_node, new_node)

elif operation == 'remove':
    index = int(do[1])
    node = a_cllist.get_node(index)
    if node is None:
        print('No such index.')
        continue
    a_cllist.remove(node)

elif operation == 'quit':
    break
```

Program Explanation

1. An instance of CircularLinkedList is created.
2. The user is presented with a menu to perform various operations on the list.
3. The corresponding methods are called to perform each operation.

Runtime Test Cases

```
Case 1:
Menu
insert <data> after <index>
insert <data> before <index>
insert <data> at beg
insert <data> at end
remove <index>
quit
The list:
What would you like to do? insert 7 at beg
The list: 7
What would you like to do? insert 1 before 0
The list: 7 1
What would you like to do? insert 10 after 0
The list: 7 10 1
What would you like to do? insert 3 at end
The list: 7 10 1 3
What would you like to do? remove 2
The list: 7 10 3
What would you like to do? remove 0
The list: 10 3
What would you like to do? quit
```

```
Case 2:
Menu
insert <data> after <index>
insert <data> before <index>
insert <data> at beg
insert <data> at end
remove <index>
quit
The list:
What would you like to do? insert 1 at beg
The list: 1
What would you like to do? insert 3 after 2
No such index.
The list: 1
What would you like to do? remove 1
No such index.
The list: 1
What would you like to do? insert 6 after 0
The list: 1 6
What would you like to do? remove 0
The list: 6
What would you like to do? quit
```