

PL Assignment #02 : Make Linked List 보고서

컴퓨터공학과 3학년

201504280 신윤호

1. 문제 해결 방법

1) `_link_last(self, element, node)`

- 함수에 전달된 노드가 적절한 노드(맨 마지막 노드)일 경우에는 새로운 노드를 생성해 element값을 저장하고, 해당 노드의 next에 연결한다. 한편, 이는 순환 호출의 종료 조건이 된다.
- 함수에 전달된 노드가 맨 마지막 노드가 아닐 경우에는 `node.next`를 인자로 함수를 순환 호출한다.

2) `_get_node(self, index, x)`

- 함수에 전달된 index값이 0인 경우 노드 탐색이 완료된 것이다. 이 경우 해당 노드를 반환한다. 이는 순환 호출의 종료 조건이 된다.
- 함수에 전달된 index값이 0 이상인 경우 노드 탐색이 진행중인 것이다. 인덱스 값을 1 줄이고(`index-1`) 다음 노드로(`x.next`) 함수를 순환 호출한다. 가령 3번 노드(4번째 노드)를 찾고자 한다면 index값이 $3 \rightarrow 2 \rightarrow 1 \rightarrow 0$ 이 되어 해당 노드를 찾아내는 방식이다.

3) `__str__(self)`

- 다음 노드가 없는 경우에는 자신이 가진 element를 반환한다. 이는 순환 호출의 종료 조건이 된다.
- 다음 노드가 있는 경우에는 '자신의 element + self.next.__str__()'을 반환한다. 즉, 자신이 가진 element와 __str__함수를 다음 노드에 순환 호출하여 얻어진 값을 더해서 반환한다.

4) `__len__(self)`

- 다음 노드가 없는 경우에는 1을 반환한다. 이는 순환 호출의 종료 조건이 된다.
- 다음 노드가 있는 경우에는 '1+self.next.__len__()'을 반환한다. 즉, 자기 자신을 카운팅한 값인 1과 __len__()함수를 다음 노드에 순환 호출하여 얻어진 값을 더해서 반환한다.

5) `_reverse(self, x, pred)`

- 링크를 바꾸게 되면 `x.next`로 가는 링크를 잃게 되기 때문에, 링크를 바꾸기에 앞서 `self.head`를 `x.next`로 한다.
- `pred`는 `x`의 선행노드를 가리키고 있다. 따라서 `x.next = pred`로 하여 링크를 뒤집어준다.
- `pred`를 `x`로 진행시킨다.
- 이 과정을 `self._reverse(self.head, pred)`를 통해 순환 호출한다. 즉, `x`의 후행노드를 가리키고 있는 `self.head`를 파라미터 `x`에 전달하고 현재 `x`위치를 가리키고 있는 `pred`를 파라미터 `pred`에 전달해 다음 노드를 기준으로 위 과정을 수행한다.
- `x.next == None`이 되는 경우는 맨 마지막 노드이다. 이 상태에서는 더 이상 순환호출을 수행하지 않아도 되므로 `x.next = pred`로 하여 링크만 조절해준다. head는 이전 단계에서 이미 마지막 노드(reverse된 리스트 기준으로는 맨 처음 노드)를 가리키고 있기 때문에 별도의

처리를 하지 않는다. 이것이 끝나면 순환호출을 빠져나가며 함수가 종료된다.

6) `_selection(self, current_node)`

- `current_node`가 `None`이 아닌 경우 수행한다.
- 최솟값을 저장할 `min_node`를 생성한다. `min_node`는 `current_node`로 초기화되고, `compare` 메소드에 의해 최솟값을 가지게 된다. `compare` 메소드를 수행하면 `min_node`와 `current_node`의 값을 교환해 준다. 그 후, 다음 노드로 함수를 순환 호출한다. 즉, `_selection` 함수를 `current_node.next`를 인자로 순환 호출한다.
- `current_node`가 `None`이 되면 정렬이 완료된 것이므로 순환호출을 빠져나가며 함수를 종료한다.

7) `compare(self, current_node, compare_node, min_node)`

- `current_node`가 `None`이 아닌 경우 수행한다.
- `min_node`의 값과 `compare_node`의 값을 비교한다. 만약 현재 `min_node`가 가진 값보다 더 작은 값을 `compare_node`가 가지고 있다면, `min_node`가 해당 노드를 가리키도록 한다. 그리고 `compare_node`가 다음 노드를 가리키게 한 후, 함수를 순환 호출한다.
- 만약 `min_node`가 가진 값이 `compare_node`의 값보다 작다면 `min_node`의 변화 없이 다음 노드로(`compare_node.next`) 함수를 순환 호출한다.
- `compare_node`가 `None`이 되면 모든 비교를 완료한 것이고, `min_node`는 정렬 대상 중 최솟값을 가진 노드를 가리킨다. 이를 반환한다.

2. 느낀점

연결리스트 자료구조를 순환을 이용해 구현해본 것은 처음이었다. 지난번 과제에 이어 순환 호출에 대한 이해를 심화시킬 수 있는 좋은 기회였다. 특히 선택정렬을 반복을 이용해 구현해본 적은 있지만, 순환 호출을 이용해 구현해본 것은 이번이 처음이라 많은 공부가 되었다. 또한 파이썬의 클래스와 메소드를 이번 기회에 다루어보게 되었는데 `self`에 관한 개념, 스페셜 메소드 등을 알게 되어 파이썬을 학습하는데 도움이 되었다.

3. 테스트 코드 실행 결과

```
def test_recursion_linked_list():
    INPUT = ['a', 'b', 'c', 'd', 'e']
    test_RLL = RecursionLinkedList()
    for i in INPUT:
        test_RLL.add(i)
    print str(test_RLL)
    print "List size is " + str(len(test_RLL))
    test_RLL.add('z', 0)
    print "List size is " + str(len(test_RLL))
    print str(test_RLL)
    print test_RLL.get_node(4).element
    print test_RLL.remove(0)
```

```

print str(test_RLL)
test_RLL.reverse()
print str(test_RLL)

def test_selection_sort():
    random_numbers=[]
    for i in range(10):
        random_numbers.append(random.randrange(0, 100))

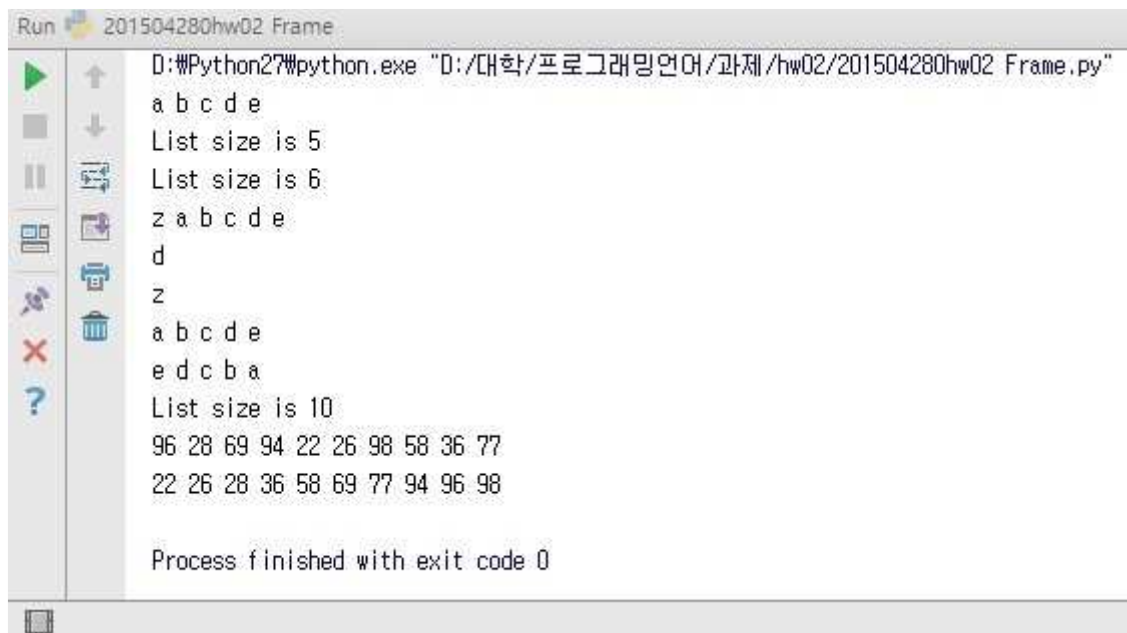
    test_RSS = RecursionLinkedList()
    for i in random_numbers:
        test_RSS.add(i)
    print "List size is " + str(len(test_RSS))
    print str(test_RSS)
    #test_RLL.iter_selection_sort()
    test_RSS.selection_sort()
    print str(test_RSS)

```

```

test_recursion_linked_list()
test_selection_sort()

```



Run 201504280hw02 Frame

```

D:\Python27\python.exe "D:/대학/프로그래밍언어/과제/hw02/201504280hw02 Frame.py"
a b c d e
List size is 5
List size is 6
z a b c d e
d
z
a b c d e
e d c b a
List size is 10
96 28 69 94 22 26 98 58 36 77
22 26 28 36 58 69 77 94 96 98

Process finished with exit code 0

```