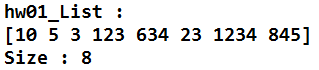
|  |  |  |
| --- | --- | --- |
| 과제 번호 | : | hw01 |
| 분 반 | : | 00분반 |
| 소 속 | : | 컴퓨터공학과 |
| 학 번 | : | 201000869 |
| 이 름 | : | 최준우 |

프로그래밍 언어 개론

1. **Algorithm 및 Method 설명**
   1. 주요 알고리즘 및 자료구조
      1. Linked List : 각각의 생성된 Node들을 포인터로 연결하여 List 형태로 저장하여 자료에 접근하는 구조이다.
      2. Recursion : 주어진 문제를 해결하기 위해 하나의 함수에서 자신을 다시 호출하여 작업을 수행하는 방식이다.
   2. 주요 함수 설명서
      1. private void linkLast(int element, Node x)  
          Node x의 next를 검사하여 null인지 확인한다. List의 특성에 의해 next가 null이면 List의 End point이므로 next가 null일때 까지 linkLast를 재귀 호출하여 다음 노드를 방문하며 list의 last를 찾은 뒤, 새로운 노드를 끝에 추가한다.
      2. private Node node(int index, Node x)  
          해당 index를 찾아갈 때, node를 call할 때마다 index를 1 줄여주며 index가 0이 되면 해당 node를 반환한다.
      3. private int length(Node x)  
          재귀 호출을 하면 call stack이 쌓이는 것을 이용하여 call이 return 되는 list 끝에서 1을 반환하고 쌓인 스택마다 +1을 하여 list의 length를 알 수 있다.
      4. private int length(Node x)  
          재귀 호출을 하면 call stack이 쌓이는 것을 이용하여 call이 return 되는 list 끝에서 1을 반환하고 쌓인 스택마다 +1을 하여 list의 length를 알 수 있다.
      5. private String toString(Node x)  
          Node가 last가 아닐 때, 자신의 item과 다음 toString을 재귀 호출하여 스택의 최상단에서 last에 있는 item을 반환하며 반환 될 때 마다 String에 해당 node에 item이 쌓여 모든 노드에 있는 item을 반환한다.
      6. private void reverse(Node x, Node pred)  
          list의 끝까지 스택을 쌓고, list의 끝을 만나면 last node를 head에 대입한 뒤, 스택을 내려오면서 node x의 next를 pred node로 연결하여 준다
2. **실행 결과 및 분석**
   1. 입력 파일(hw01.txt)



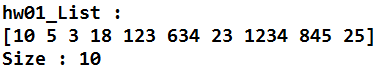
* 1. Test
     + 1. hw01.txt 안에 있는 data를 linked list로 만든다.
          1. 코드  
             System.out.println("hw01\_List : ");  
             System.out.println(list);  
             System.out.println("Size : " + list.size());
          2. 결과 화면



* + - 1. list의 끝에 element가 25인 node를 추가한다.
         1. 코드  
            System.out.println("\nhw01\_List : ");  
            list.add(25);  
            System.out.println(list);  
            System.out.println("Size : " + list.size());
         2. 결과 화면



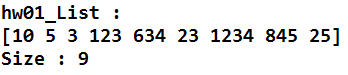
* + - 1. index가 3인 자리에 element가 18인 node를 추가한다.
         1. 코드  
            System.out.println("\nhw01\_List : ");  
            list.add(3, 18);  
            System.out.println(list);  
            System.out.println("Size : " + list.size());
         2. 결과 화면



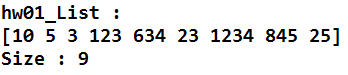
* + - 1. index가 3인 자리에 element를 출력한다.
         1. 코드  
            System.out.println("\nindex 3 : " + list.get(3));
         2. 결과 화면



* + - 1. index가 3인 자리의 node를 삭제한다.
         1. 코드  
            System.out.println("\nhw01\_List : ");  
            list.remove(3);  
            System.out.println(list);  
            System.out.println("Size : " + list.size());
         2. 결과 화면



* + - 1. list를 반전한 뒤 출력하여 준다.
         1. 코드  
            System.out.println("\nReverse\_List : ");  
            list.reverse();  
            System.out.println(list);  
            System.out.println("Size : " + list.size());
         2. 결과 화면



1. **구현 후 느낀 점**

평소에 Iteration으로만 구현하였던 Linked List의 Method를 Recursion을 통해 구현하여 Recursion이라는 개념이 좀 더 친숙해졌다. Call Stack이 쌓이는 과정과 Stack이 반환 되면서 node에 작용하는 Action을 이해하고 Node간에 Link들의 연결을 통해 List의 구조에 대해 자세히 알 수 있었다.  
 또한 각각의 Method를 구현하며 List에서의 삽입, 삭제, 접근이 어떻게 일어 나는지 알 수 있었고, 반전 함수를 통하여 Node간 상호 작용을 하는 방법에도 친숙해 질 수 있었다.