

# REPORT

## 보고서 작성 서약서

1. 나는 타학생의 보고서를 복사(Copy)하지 않았습니다.
2. 나는 타학생의 보고서를 인터넷에서 다운로드 하여 대체하지 않았습니다.
3. 나는 타인에게 보고서 제출 전에 보고서를 보여주지 않았습니다.
4. 보고서 제출 기한을 준수하였습니다.

나는 보고서 작성시 위법 행위를 하지 않고,  
성.균.인으로서 나의 명예를 지킬 것을 약속합니다.

과 목 : 전자전기 프로그래밍실습

과 제 명 : HW7

담당교수 : 민 형 복

학 과 : 전자전기공학부

학 년 : 2

학 번 : 2017311583

이 름 : 정해진

제 출 일 : 2018.5.28

## 1. Introduction

[2]에 의하면, Linked List의 정의는 이렇다.

이것은 각 노드를 유용한 저장 공간에 그 위치에 상관없이 저장시키고, 각 노드의 관련성을 노드에 보관하여 1차원 배열 관계를 유지하도록 함으로써 중간 노드의 삽입, 제거를 손쉽게 할 수 있는 리스트로 각 노드는 링크(또는 포인터) 부분을 가지며, 그 노드와 관련 있는 다음 노드의 주소를 그 값으로 가진다. 다시 말하면 선형 리스트의 노드 배열이 어드레스와 일치하지 않고 기억 공간에 독립적으로 이루어진 리스트를 말한다.

새로운 타입의 리스트 'Linked List'를 이용하여 주어진 조건 속에서 문제를 해결한다.

## 2. Problem Statement

### ① Describe what is the problem.

1. "linked\_list.c"가 주어진다.
2. void 타입의 "printList(NODE \*)" 함수가 주어진다.
3. 다른 5개의 함수들의 코드를 완성한다.

[각 함수들의 수행 목표]

headInsert : List의 맨 앞에 데이터 삽입

headDelete : List의 맨 앞의 데이터 삭제

tailInsert : List의 맨 뒤에 데이터 삽입

secondDelete : List의 헤드 다음의 데이터 삭제

deleteList : List의 모든 데이터 삭제

## ② Describe how do you solve the problem.

- Main 함수

다음의 함수들을 모두 실행하여 조건에 맞는 결과를 도출한다.

```
int headInsert(NODE **head, NODE **tail, int data);  
int headDelete(NODE **head, NODE **tail, int *data);  
int tailInsert(NODE **head, NODE **tail, int data);  
int secondDelete(NODE **head, NODE **tail, int *data);  
void deleteList(NODE **head, NODE **tail);  
static void printList(NODE *head);
```

- headInsert 함수

주어진 head pointer와 tail pointer를 가지는 linked list에 data를 list의 맨 앞에 삽입한다.

- headDelete 함수

주어진 head pointer와 tail pointer를 가지는 linked list에서 list의 맨 앞의 data를 삭제한다.

- tailInsert 함수

주어진 head pointer와 tail pointer를 가지는 linked list에 data를 list의 맨 뒤에 삽입한다

- secondDelete 함수

주어진 head pointer와 tail pointer를 가지는 linked list에서 list의 맨 앞에서 두번째 data를 삭제한다.

- deleteList 함수

주어진 head pointer와 tail pointer를 가지는 linked list의 모든 data를 삭제한다.

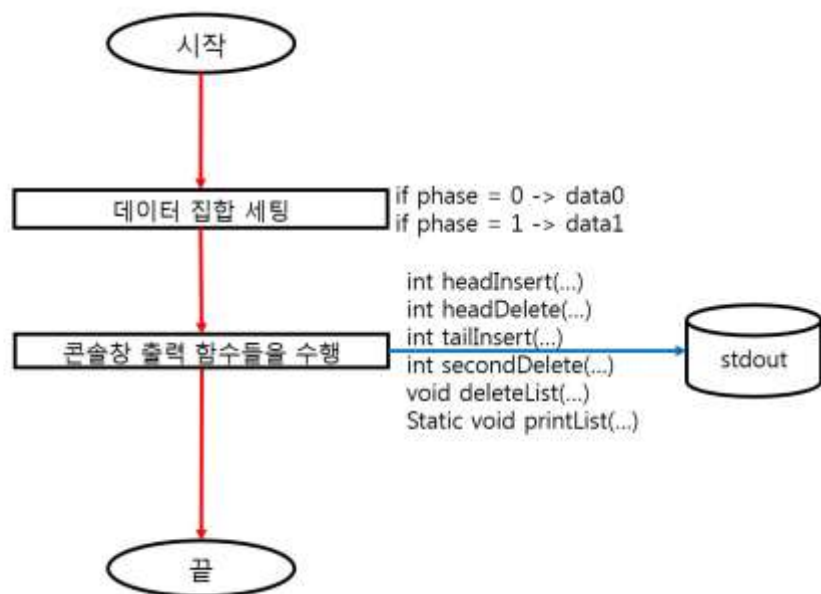
- printList 함수

주어진 head pointer와 tail pointer를 가지는 linked list의 모든 data를 출력한다.

### ③ Draw a flowchart of your algorithm

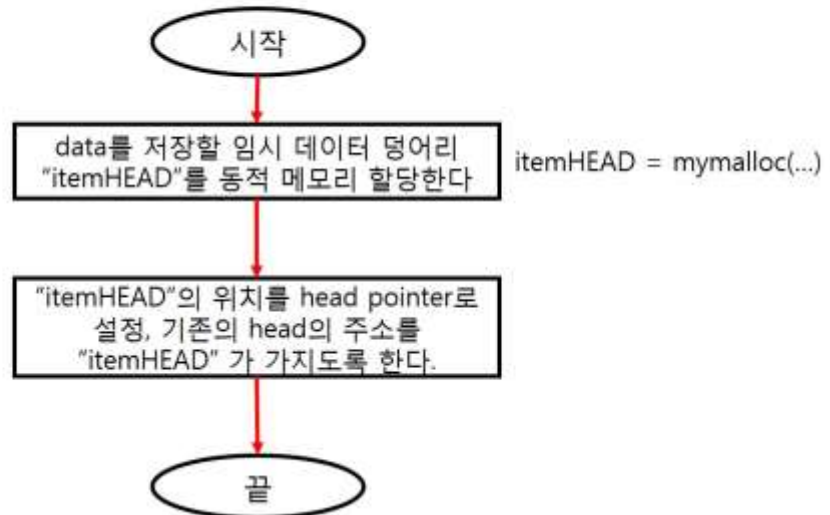
1. Main 함수

**int main(void)**



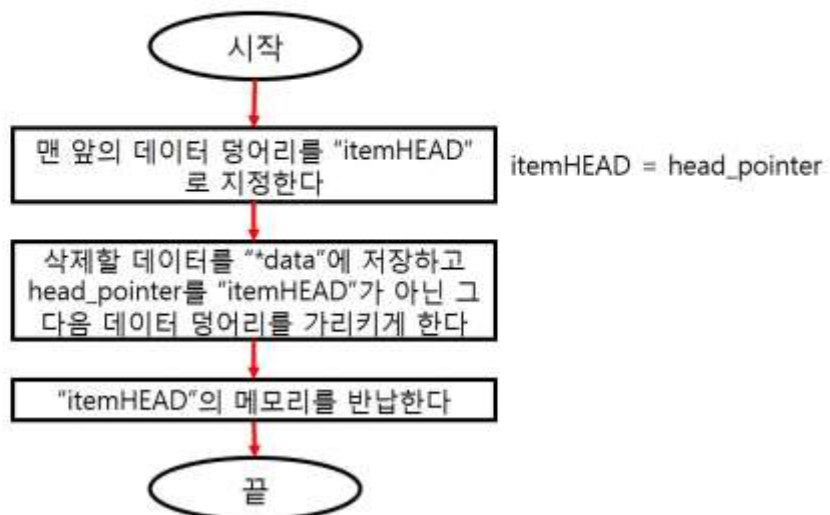
## 2. headInsert 함수

```
int headInsert (NODE **head, NODE **tail, int data)
```



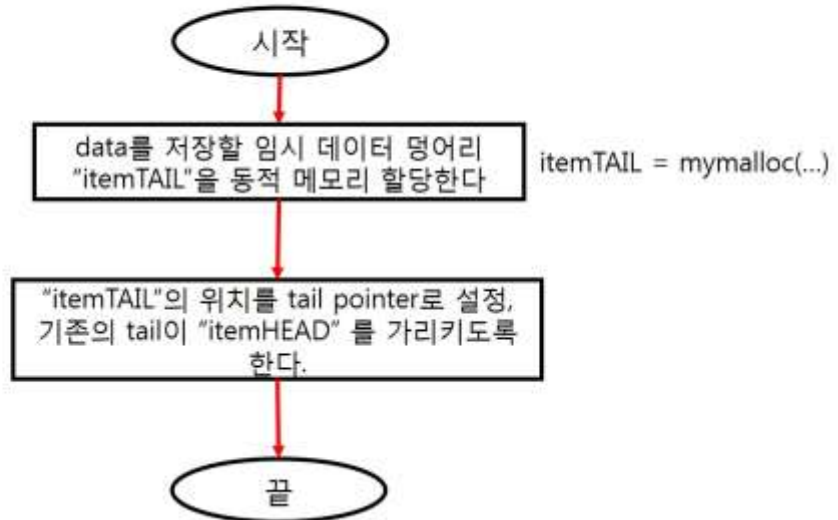
## 3. headDelete 함수

```
int headDelete (NODE **head, NODE **tail, int *data)
```



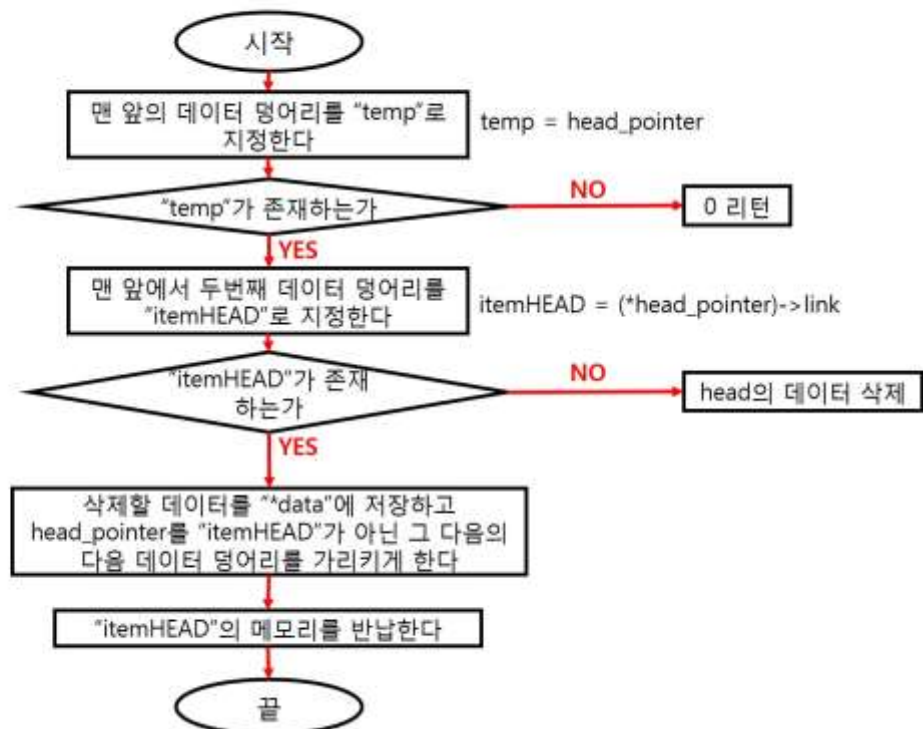
#### 4. tailInsert 함수

```
int tailInsert (NODE **head, NODE **tail, int data)
```



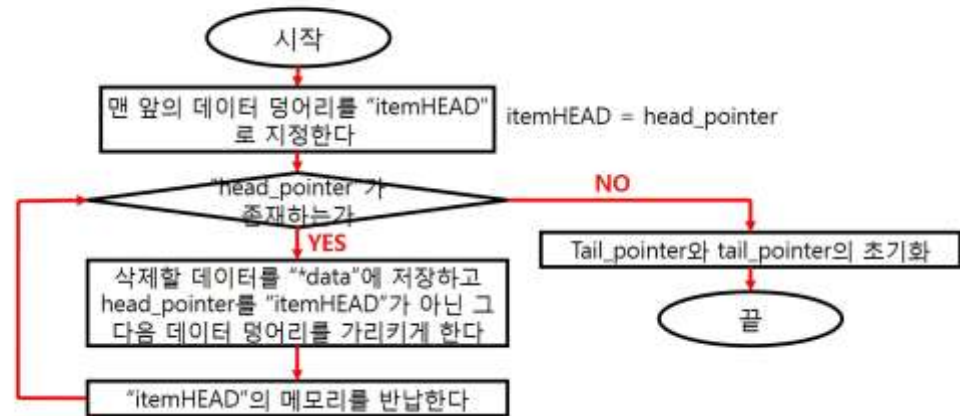
#### 5. secondDelete 함수

```
int secondDelete (NODE **head, NODE **tail, int *data)
```



## 6. deleteList 함수

```
void deleteList (NODE **head, NODE **tail)
```



## 7. printList 함수

```
static void printList (NODE **head)
```



### 3. Implementation

#### - Main 함수

다음의 함수들을 모두 실행하여 조건에 맞는 결과를 도출한다. 각 함수들은 데이터 집합 2개에 대하여, linked list를 생성하고 데이터를 삽입하거나 삭제하는 작업을 수행한다(printList 함수는 해당 linked list 출력).

```
int headInsert(NODE **head, NODE **tail, int data);
int headDelete(NODE **head, NODE **tail, int *data);
int tailInsert(NODE **head, NODE **tail, int data);
int secondDelete(NODE **head, NODE **tail, int *data);
void deleteList(NODE **head, NODE **tail);
static void printList(NODE *head);
```

#### - headInsert 함수

주어진 head pointer와 tail pointer를 가지는 linked list에 data를 list의 맨 앞에 삽입한다. 삽입할 때, 임시 데이터 덩어리 "itemHEAD"를 dynamic memory allocation을 통해 생성하고, 그 데이터 덩어리를 기존의 linked list에 연결하여 새로운 linked list를 만들도록 한다.

#### - headDelete 함수

주어진 head pointer와 tail pointer를 가지는 linked list에서 list의 맨 앞의 data를 삭제한다.

#### - tailInsert 함수

주어진 head pointer와 tail pointer를 가지는 linked list에 data를 list의 맨 뒤에 삽입한다. 삽입할 때, 임시 데이터 덩어리 "itemTAIL"을 dynamic memory allocation을 통해 생성하고, 그 데이터 덩어리를 기존의 linked list에 연결하여 새로운 linked list를 만들도록 한다.

#### - secondDelete 함수



주어진 head pointer와 tail pointer를 가지는 linked list에서 list의 맨 앞에서 두번째 data를 삭제한다. 맨 앞의 데이터와 그 다음 데이터를 다뤄서 함수 기능을 원활히 이끌어낸다. 데이터가 하나인 경우와 존재하지 않는 경우 같은 예외도 생각한다. 데이터가 하나인 경우에는 list의 맨 앞부분에 해당되는 head 부분이므로, head data를 삭제한다. 데이터가 존재하지 않는 경우에는 0을 return한다.

- deleteList 함수

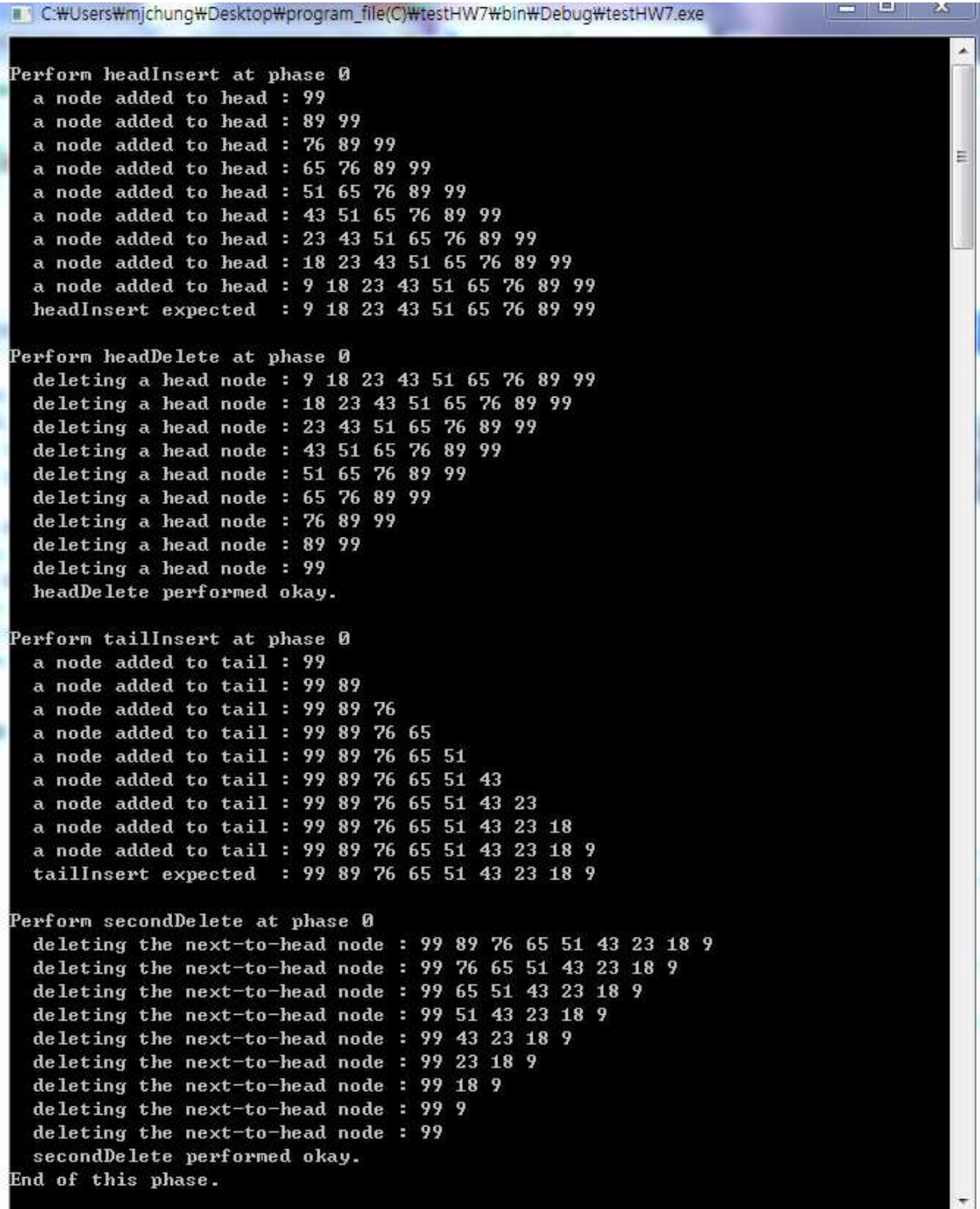
주어진 head pointer와 tail pointer를 가지는 linked list의 모든 data를 삭제한다. headDelete 함수를 반복 실행하는 것과 같다.

- printList 함수

주어진 head pointer와 tail pointer를 가지는 linked list의 모든 data를 출력한다.

## 4. Result

"2017311583.정해진.HW7.c" 를 컴파일 후 실행한 결과이다.



```
C:\Users\Wmjchung\Desktop\program_file(C)\testHW7\bin\Debug\testHW7.exe

Perform headInsert at phase 0
a node added to head : 99
a node added to head : 89 99
a node added to head : 76 89 99
a node added to head : 65 76 89 99
a node added to head : 51 65 76 89 99
a node added to head : 43 51 65 76 89 99
a node added to head : 23 43 51 65 76 89 99
a node added to head : 18 23 43 51 65 76 89 99
a node added to head : 9 18 23 43 51 65 76 89 99
headInsert expected : 9 18 23 43 51 65 76 89 99

Perform headDelete at phase 0
deleting a head node : 9 18 23 43 51 65 76 89 99
deleting a head node : 18 23 43 51 65 76 89 99
deleting a head node : 23 43 51 65 76 89 99
deleting a head node : 43 51 65 76 89 99
deleting a head node : 51 65 76 89 99
deleting a head node : 65 76 89 99
deleting a head node : 76 89 99
deleting a head node : 89 99
deleting a head node : 99
headDelete performed okay.

Perform tailInsert at phase 0
a node added to tail : 99
a node added to tail : 99 89
a node added to tail : 99 89 76
a node added to tail : 99 89 76 65
a node added to tail : 99 89 76 65 51
a node added to tail : 99 89 76 65 51 43
a node added to tail : 99 89 76 65 51 43 23
a node added to tail : 99 89 76 65 51 43 23 18
a node added to tail : 99 89 76 65 51 43 23 18 9
tailInsert expected : 99 89 76 65 51 43 23 18 9

Perform secondDelete at phase 0
deleting the next-to-head node : 99 89 76 65 51 43 23 18 9
deleting the next-to-head node : 99 76 65 51 43 23 18 9
deleting the next-to-head node : 99 65 51 43 23 18 9
deleting the next-to-head node : 99 51 43 23 18 9
deleting the next-to-head node : 99 43 23 18 9
deleting the next-to-head node : 99 23 18 9
deleting the next-to-head node : 99 18 9
deleting the next-to-head node : 99 9
deleting the next-to-head node : 99
secondDelete performed okay.
End of this phase.
```

첫번째 데이터 집합 {99, 89, 76, 65, 51, 43, 23, 18, 9}에 대해 잘 실행되었다.

```
C:\Users\Wmjchung\Desktop\program_file(C)\testHW7\bin\Debug\testHW7.exe

Perform headInsert at phase 1
  a node added to head : 512
  a node added to head : 415 512
  a node added to head : 328 415 512
  a node added to head : 256 328 415 512
  a node added to head : 128 256 328 415 512
  headInsert expected : 128 256 328 415 512

Perform headDelete at phase 1
  deleting a head node : 128 256 328 415 512
  deleting a head node : 256 328 415 512
  deleting a head node : 328 415 512
  deleting a head node : 415 512
  deleting a head node : 512
  headDelete performed okay.

Perform tailInsert at phase 1
  a node added to tail : 512
  a node added to tail : 512 415
  a node added to tail : 512 415 328
  a node added to tail : 512 415 328 256
  a node added to tail : 512 415 328 256 128
  tailInsert expected : 512 415 328 256 128

Perform secondDelete at phase 1
  deleting the next-to-head node : 512 415 328 256 128
  deleting the next-to-head node : 512 328 256 128
  deleting the next-to-head node : 512 256 128
  deleting the next-to-head node : 512 128
  deleting the next-to-head node : 512
  secondDelete performed okay.

End of this phase.
+++++ Checking memory... +++++
+++++ Memory is okay. +++++

Process returned 0 (0x0)   execution time : 0.049 s
Press any key to continue.
```

두번째 데이터 집합 {512, 415, 328, 256, 128}에 대해 잘 실행되었다.

## 5. Conclusion & Evaluation

실습을 통해 새로운 형식의 list인 linked list를 다루는 방법에 대해 잘 알게 되었다. 그리고 이를 계기로 다른 형식의 linked list가 있는지 알아보게 되었는데, 원형 linked list(순환 구조의 linked list), double linked list(양 옆에 포인터가 달려있는 구조) 등이 있다.

## 6. 참고 문헌

[1] Min, H. B. and SKKU, “linkedlist\_slide.pdf”

[2] 컴퓨터인터넷 IT 용어대사전, “연결된 리스트”, 2011. 1. 20.