# 더미 기반 DBLinkedList 활용
# ID 추천 프로그램

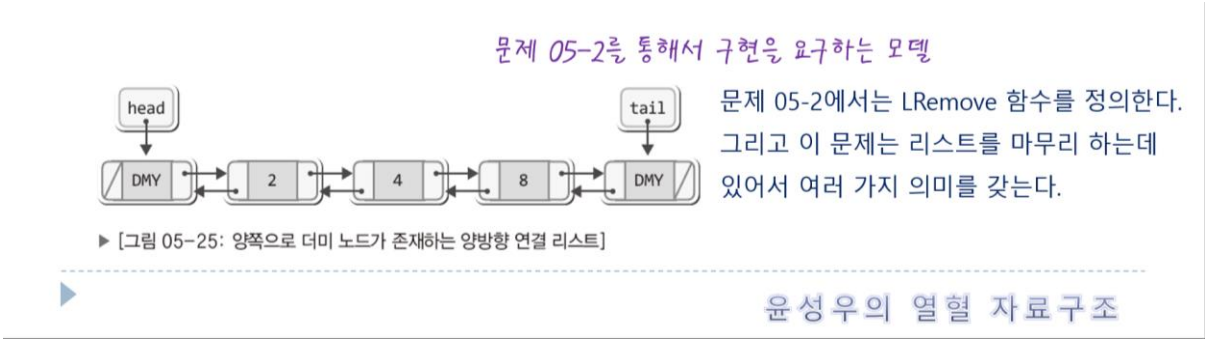
201811567_ 주용한

## 사용된 연결리스트

더미 기반으로 만들어진 양방향 연결리스트 사용.

제시된 문제의 각 단계별 알고리즘을 살펴보면 양 끝 방향으로의 추가 또는 삭제가 많이 이루어진다.
따라서 양쪽 방향으로 접근할 수 있는 더미 기반 양방향 리스트의 ADT 를 먼저 만들고 ADT 를 기반으로 메인 코드 작성 후 구현했다.

아래 그림은 강의자료 Chapter 05 연결리스트 - 22p 에 나와있는 더미기반 양방향 연결리스트의 모양이다. 밑의 그림을 참고해서 DBLinkedList.h, DBLinkedList.c 를 만들었다.




장점: head 또는 tail 양쪽으로 접근이 가능하며 더미 기반으로 노드를 추가할 시 head 또는 tail 양쪽으로 모두 추가할 수 있다. (첫번째 노드와 그후 노드 모두 같은 코드 적용가능)

ex) 6 단계(크기가 15 보다 크면 뒤 부분 삭제)의 알고리즘 구현 부분 일부를 살펴보면

```
while (LCount(&charList)>15){       // 크기가 15 보다 클 때
        LLast(&charList, pdata);        // 마지막으로 이동, 삭제
        LRemove(&charList);
}
```

위 코드를 보면

1. 크기가 15 보다 더 크다면 반복해서 LLast()을 통해 마지막 노드로 이동
2. 삭제 및 반복

**tail 으로 접근 가능하다는 장점**을 이용해 아주 간단하게 6 단계를 구현할 수 있다.

## ADT 구성요소

```
typedef char LData;                    // char 데이터
typedef struct _node                   // 이전노드의 위치 포함
{          Node *next, prev;
             LData  data;
} Node;


typedef struct _dblinkedList           //리스트
{                                      // tail 을 사용해 꼬리 쪽으로 접근, 노드 추가 등 가능
          Node *head, tail, cur;
          int numOfData ;
}DBLinkedList;
```

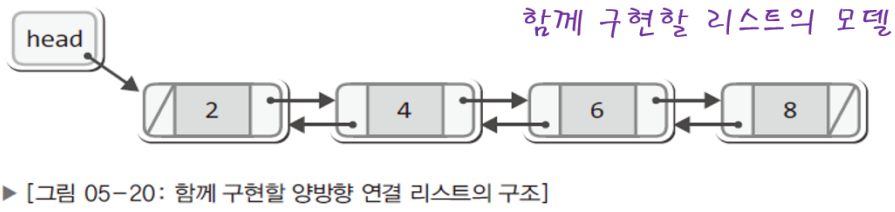
**구성 함수**

```
void ListInit(List *plist);                  // List 초기화
void LInsert(List *plist, LData data);       // Head 에 노드 추가
void TInsert(List *plist, LData data);       // Tail 에 노드 추가
int LFirst(List *plist, LData *pdata);       // 처음으로 이동 후 TRUE or FALSE 반환
int LLast(List *plist, LData *pdata);        // 마지막으로 이동 후 TRUE or FALSE 반환
int LNext(List *plist, LData *pdata);        // 다음으로 이동 후 TRUE or FALSE 반환
int LPrevious(List *plist, LData *pdata);    // 이전으로 이동 후 TRUE or FALSE 반환
int LCount(List *plist);                     // List 크기 반환
int LReplace(List *plist, LData data);       // 노드 안 데이터 교체후 TRUE or FALSE 반환
LData LRemove(List *plistm);                 // 노드삭제 후 노드안의 데이터 반환
```

## 변경된 ADT

아래는 수업에 나온 양방향 연결리스트이다. (Chapter 05 연결리스트 - 22p)

## 기존 양방향 리스트



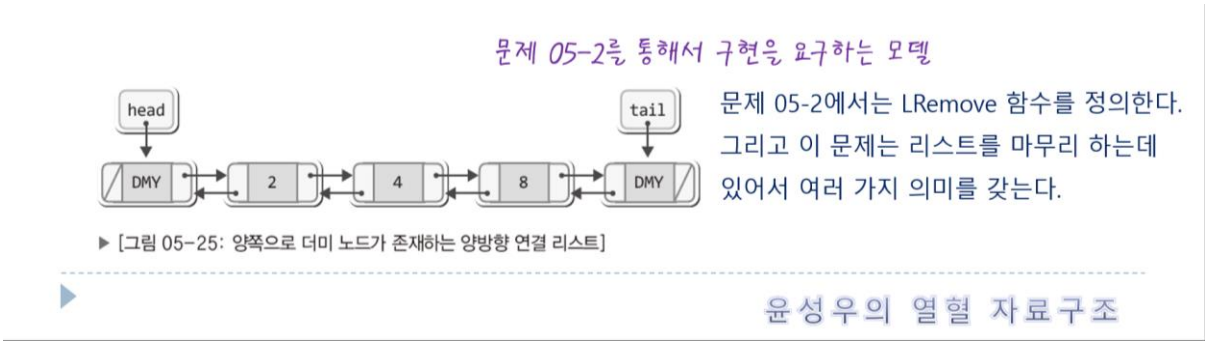

▶ [그림 05-20: 함께 구현할 양방향 연결 리스트의 구조]

√ LRemove 함수를 ADT에서 제외시킨다.

√ 대신에 왼쪽 노드의 데이터를 참조하는 LPrevious 함수를 ADT에 추가시킨다.

√ 새 노드는 머리에 추가한다.

## 더미기반 양방향 리스트




## 기존 내용에서 변경 및 추가시킨 후

1. 기존 첫번째 데이터 삽입과 그 이후 데이터 삽입이 다른 코드를 통해 이루어지지만 더미 기반을 이용해 **같은 방식으로** 첫 데이터와 그후 데이터를 삽입을 할 수 있다.
2. 새 노드를 TAIL 에도 추가시킬 수 있다.
3. LRemove 함수를 ADT 에서 제외시키지 않는다.
4. LFirst()처럼 LLast()를 통해 마지막으로 이동할 수 있다.
5. LReplace() 함수를 이용해 노드의 데이터를 변경시킬 수 있다.

## ADT 수정 이유

## 1 번의 경우

0 번째 노드와 이후 노드를 같은 방식으로 노드를 추가하여 더미 기반 리스트의 장점을 살릴 수 있다.

## 2 번의 경우

Tail 로 추가해야 하는 알고리즘이 필요해서 추가하게 되었다.

Ex - 7 단계 알고리즘(크기가 3 이하일 때 리스트 연장)을 살펴보면

```
LLast(&charList, pdata);                // 마지막 노드로 이동, 데이터 pdata 에 받아오기
while (LCount(&charList)<3){            // 크기가 3 이하일 때
        TInsert(&charList, *pdata);     // tail 쪽으로 마지막 노드의 데이터를 추가
}
```

와 같은 방식으로 구현되는데 꼬리 삽입 함수가 사용된다.

## 3 번의 경우

강의자료 ppt 에서는 LRemove 를 제외시킨다고 나와있지만 LRemove 함수는 알고리즘 구현에 있어 필수적이므로 제외시키지 않고 구현하도록 한다.

## 4 번의 경우

2 번의 경우에서 나왔던 예시코드처럼 LLast() 함수를 이용해 바로 끝부분의 노드에 접근 할 수 있다.

## 5 번의 경우

1 단계 알고리즘은 대문자를 소문자로 바꾸는 역할을 하는데 노드를 삭제하거나 추가하지 않고 간편히 대문자에서 소문자로 바꿀 수 있다.