Ch4 List

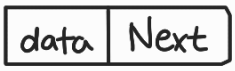
: data의 명단으로서 배열에 저장하거나 또는 linked-list에 저장
↳ Singly-Linked List (SLL)
Doubly-Linked List (DLL)

* 모든 ADT의 기본 연산: ① 검색 (search)
② 수정 (modify)
③ 삽입 (insert, add)
④ 삭제 (delete)

1. SLL (Singly-Linked List)

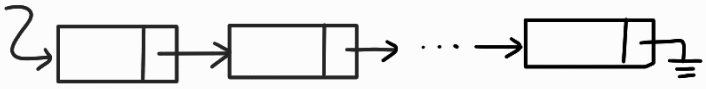
SLL의 구조체

```
typedef struct
{ Data data;
  Node *Next;
} Node;
```



Node *Head, *p

상태 { Head = NULL (⏚) (empty list)
Head



① 검색 (search) (key를 찾음)

```
p = Head
while (p ≠ NULL)
{   if (p → data == key) Found!!
    p = p → Next
}
→ Not found !!
```

} 선형검색 O(n)
linear search

〈참고〉 i) 선형 검색 (linear search) → O(n)
: 정렬 (sorting)이 안되어 있거나 sequential memory (tape, Linked List)

ii) 이진 검색 (binary search) → O(lg n)
: 배열 등의 RAM에 sorting 되어 저장되었을 때

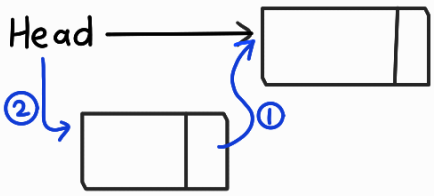
② 삽입 (insert) (p를 삽입)


```
p = (Node *) malloc (sizeof (Node))
p → data = data
```

i) 맨 앞에 삽입

① p → Next = Head

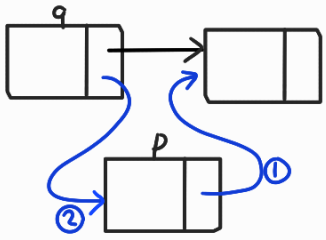
② Head = p



ii) 중간/끝에 삽입 (q 다음에)

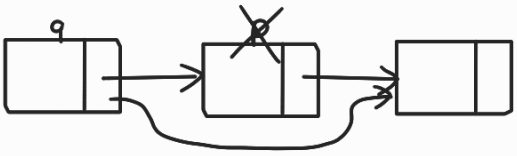
① p → Next = q → Next

② q → Next = p



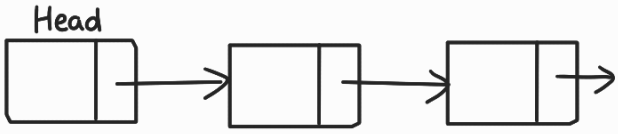
③ 삭제 (delete) (q 다음 원소 p 삭제)

```
if (Head == p) Head = p → Next
else  q → Next = p → Next
      ⋮
free (p)
```



참고) Head = (Node *) malloc (sizeof (Node))

⇒ 삽입/삭제 시 조건 test 불필요



① 삽입 (p를 q 다음에)

p → Next = q → Next

q → Next = p

② 삭제 (p 다음원소 q 삭제)

q → Next = p → Next
or
q → Next → Next

⋮

free (p)

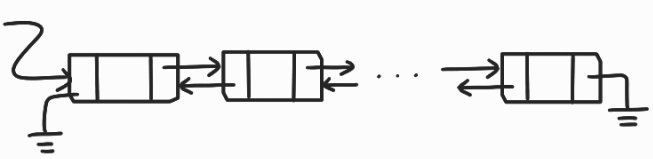
2. DLL (Doubly-Linked List)

DLL의 구조체

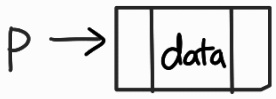
```
typedef struct{
    { Data data;
      Node *Prev, *Next;
    } Node;
```

Node *Head, *p, *q;

상태 { Head = NULL (empty list)
Head



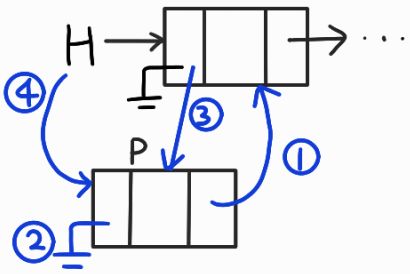
① 삽입 (insert)



(p = (Node *) malloc (sizeof (Node))
p → data = data

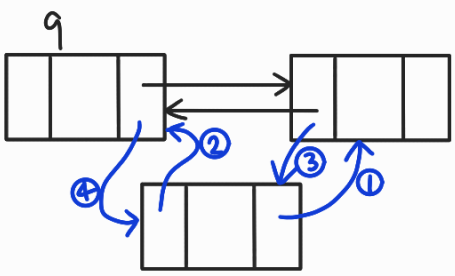
i) 맨 앞에

① p → Next = Head
② p → Prev = NULL
③ if (Head ≠ NULL)
    Head → Prev = p
④ Head = p

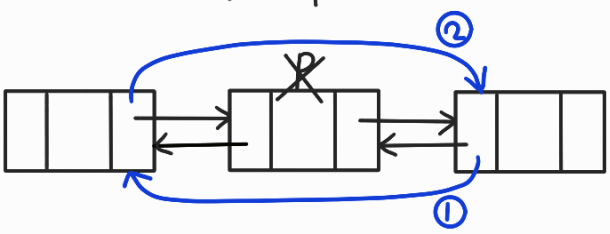


ii) 중간 / 끝에 (q 다음에)

① p → Next = q → Next
② p → Prev = q
③ if (q → Next ≠ NULL)
    q → Next → Prev = p
④ q → Next = p



② 삭제 (delete) (p를 삭제)



p → Next → Prev = P → Prev

① if (p → Next ≠ NULL) p → Next → Prev = p → Prev

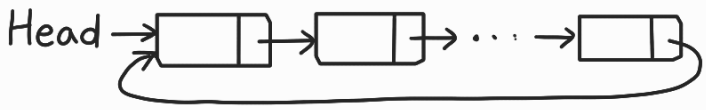
② if (p → Prev ≠ NULL) p → Prev → Next = p → Next

else Head = p → Next

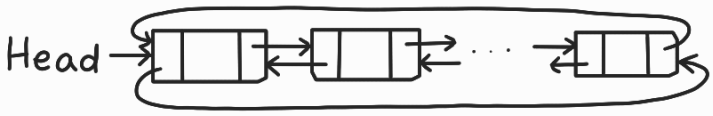
⋮

free(p)

3. 원형 리스트 (Circular List)



CSLL



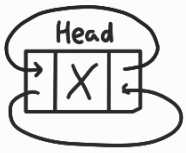
CDLL

4. Circular DLL with Structural Head (CDLLS)

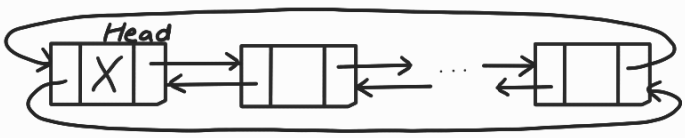
Node *Head, *p, *q

Head = (Node *) malloc (sizeof(Node))

상태



: empty list (H → Next = H → Prev = H)



특징 : NULL pointer가 전혀 없어서 삽입/삭제가 단순해진다.

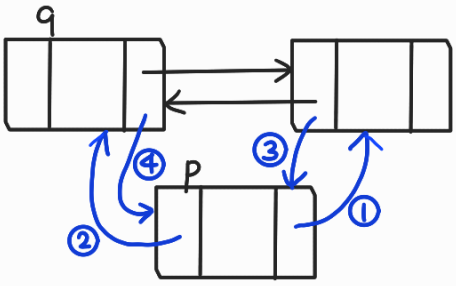
① 삽입 (insert) : q 다음에 p 삽입 (q == Head 가능)

① p → Next = q → Next

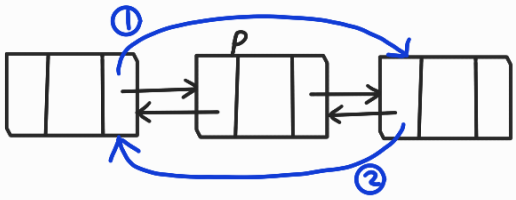
② p → Prev = q

③ q → Next → Prev = p

④ q → Next = p



② 삭제 (delete) : p를 삭제



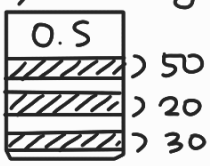
① p → Prev → Next = p → Next

② p → Next → Prev = p → Prev

⋮

free(p)

* 참고 : UNIX OS (운영체제)의 Memory Management의 실제 모델로 CDLLS를 사용
  - 사용중인 공간 list (위치, 크기)
  - 사용가능 공간 list (")



* linear linked list는 소형 데이터에만 사용 가능