Chapter 3 변경 데이터 표현

p.298, 299, 300

| | |
|---|---|
| 포인터 | - 포인터 변수는 다른 변수의 주소를 가진다.<br>- 다른 변수를 가리킨다.<br>ex) int *pi; => 주소가 가리키는 곳에 int형 값이 있다.<br>Lvalue 위치 *p=15;<br>d=*p;<br>Rvalue 값 |
| 동적 메모리할당 | - 힙 영역<br>- 할당 받고 반납(실행 중) ) 삽입, 삭제로 인한 변경이 있는 연결리스트 |
| 정적 메모리할당 | - 미리 선언되며 실행 중간에 변경 불가<br>- 경우에 따라 비효율적 |

P.301, 302

## 연결 리스트

- 단순 연결리스트
- 원형 연결리스트
- 이중 연결리스트
- 이중 원형 연결리스트

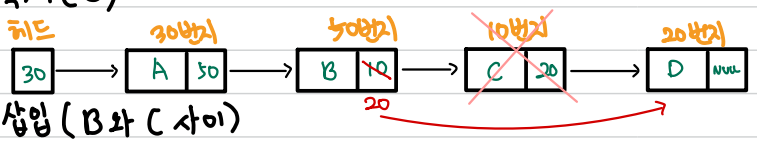
물리적인 순서 X, 기억공간에 독립적
각 노드는 링크 부분을 가진다.
접근 시간이 느리다.
삽입, 삭제는 효율적 (배열의 단점보완)

### 단순 연결리스트


삭제 (C)

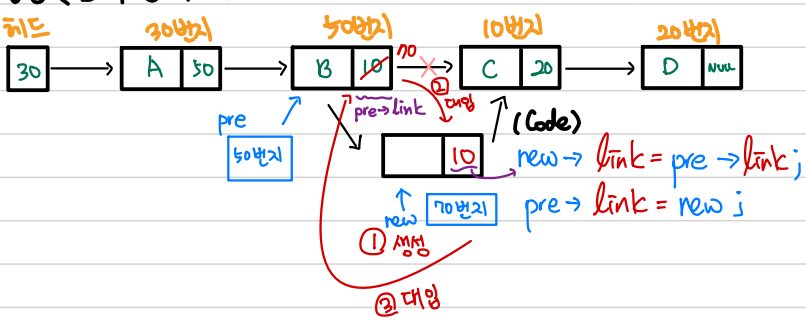
```
remove_node (L, before, removed)
if L <> NULL
  then before.link ← removed.link
  destroy (removed)
```

free (removed)




삽입 (B와 C 사이)

```
insert_node (L, before, new)
if L = NULL (연결리스트 L이 공백)
  then L ← new
  else new.link ← before.link
       before.link ← new
```

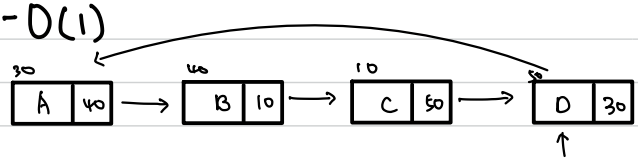



- 포인터 만큼의 기억공간 낭비
- 외부 단편화 발생 X
- 임의의 원소 수정 시 많은 탐색 요구 (헤드부터)
- 선행노드를 알아야 한다.
- O(n)

p.303, 304, 305

## 원형 연결리스트

- 연결리스트의 마지막 원소의 연결부분이 null이 아닌 첫 노드의 주소 저장
- 한 노드로부터 어디로든 연결 가능
- 헤드가 없으면 무한반복 가능성
- O(1)



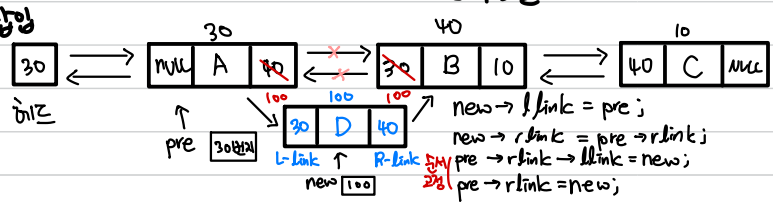

원형큐를 구현하기 위해 헤더가 마지막 노드를 가리키는 것이 효율적
∵ 마지막 삽입 ) 시 효율적
첫노드 삭제
나머지는 똑같음

## 이중 연결리스트

- 2개의 포인터, 메모리 낭비, 탐색 효율

```
new_node -> llink = before;
new_node -> rlink = before_rlink;
before -> rlink -> llink = new_node;
before -> rlink = new_node;
```

삽입




```
if(removed == phead_node) return;
removed -> llink -> rlink = removed -> rlink;
removed -> rlink -> llink = removed -> llink;
free(removed);
```

삭제