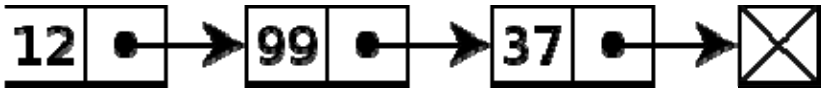
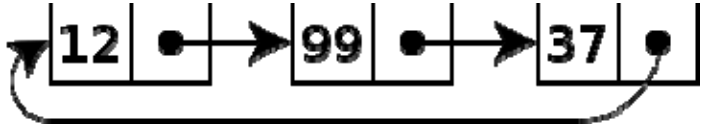
제 7 강의. 고급 연결리스트

1. 원형 연결리스트
2. 이중 연결리스트
3. 연결리스트 알고리즘





1. 원형 연결리스트(Circularly Linked Lists)

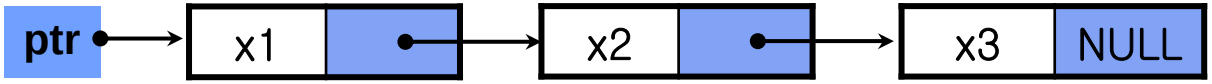
원형 연결리스트란?

연결리스트의 맨 끝 노드를 첫번째 노드와 연결시켜서 원형으로 만든 리스트

단순 연결리스트(Singly Linked List) 불편한 점

- 연결리스트의 노드 포인터를 알고 있을 때 첫번째 노드는 바로 찾아갈 수 있지만 마지막 노드는 리스트 전체를 따라가면서 끝을 찾아가야 한다(즉 n번의 링크를 따라가야 한다, O(n)).

마지막 데이터를 찾을 때, 마지막 다음에 데이터를 삽입할 때 등의 경우에 불편함이 있다.



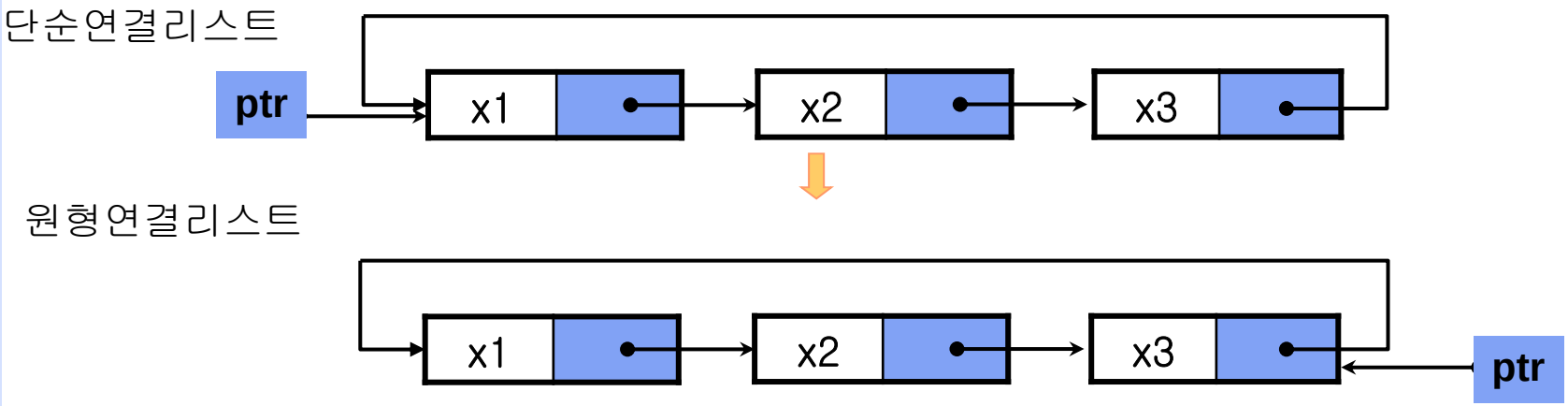
단순 연결리스트의 마지막 노드를 끝 노드와 연결시키면 ?

단순 연결리스트를 끝 노드와 연결시키면 상황은 마찬가지이지만 리스트의 포인터를 마지막 노드를 가리키면 바로 찾을 수 있다. 리스트의 첫 노드는 마지막 노드의 다음 노드이므로 리스트의 첫 노드도 바로 찾을 수 있다.(원형 연결리스트의 포인터는 항상 마지막 노드를 가리켜야 한다는 점을 주의하라.)

연결리스트 처음과 마지막에 노드 삽입/삭제

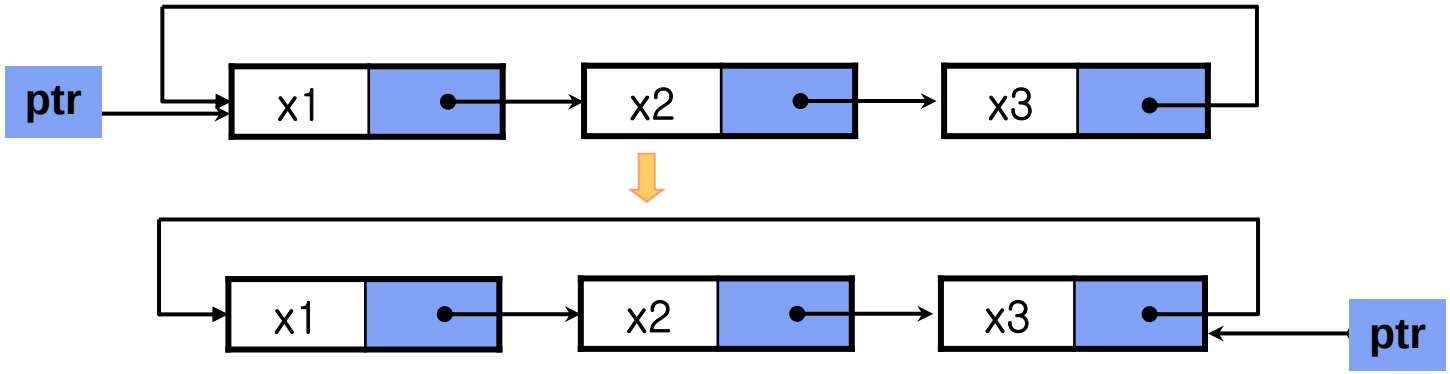
연결리스트의 맨처음과 맨마지막에 노드를 삽입, 삭제할 경우 노드수 n과 관계 – 노드를 삭제하려면 앞노드의 주소를 알아야한다.

단순/원형 연결리스트	삽입	삭제
맨처음	O(1)	O(1)
맨마지막	O(n)/O(1)	O(n)



단순 원형 연결리스트(singly circular linked list)의 예

리스트를 원형으로 만든 다음 리스트의 포인터를 마지막을 가리킨다.
그런 후 리스트의 처음과 마지막에 노드를 삽입하는 경우를 살펴보자



(1) 리스트의 처음에 삽입 – O(1) : 수행 시간이 상수시간이 걸린다.

x3 다음에 삽입한다.

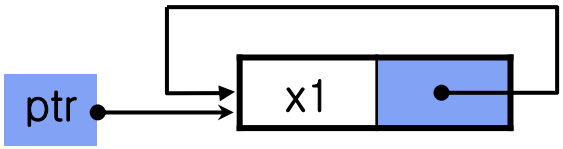
(2) 리스트의 마지막에 삽입 – O(1) : 수행 시간이 상수시간이 걸린다.

x3 다음에 삽입한다. ptr 값을 새로 삽입한 노드를 가리킨다.

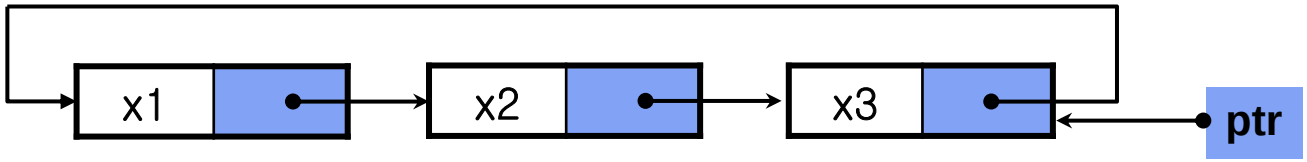
단순 원형 연결리스트(singly circular linked list)의 여러 가지 모습

(1)

(2)



(3)



수행시간 표기법

수행시간이 데이터 개수에 따라 어떻게 변하는 가를 나타내는 표기로 O-표기법을 사용한다.

O(1) - 상수시간 : 항상 똑 같을 때

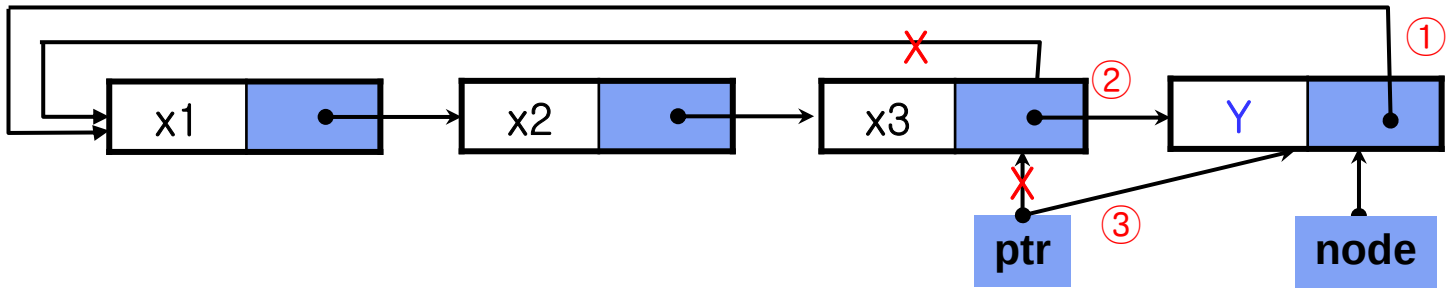
O(n) – 데이터 개수 n의 1차 함수보다 크지 않을 때

$O(n^2)$ – 데이터 개수 n의 이차함수(n^2)보다 크지 않을 때

즉 O(f(n)) 수행시간이라 함은 수행시간이 많이 걸려야 f(n) 함수보다는 작거나 같다는 의미이다.

즉 프로그램 수행이 얼마나 오래 걸릴 것인지에 대한 척도이다.

```
/* 원형 연결리스트의 맨 뒤에 노드를 삽입하는 알고리즘 */
void insert_last(list_ptr *ptr, list_ptr node)
{
    if(IS_EMPTY(*ptr))
    { /* 빈 리스트 경우 */  *ptr = node; node->link = node;}
    else
    {    node->link = (*ptr)->link; /* ① */
         (*ptr)->link = node; /* ② */
         /* 노드 연결 */
         *ptr = node; /* ③ */
         /* 리스트의 맨 처음에 삽입할 경우 이 문장을 생략 */
    }
}
```

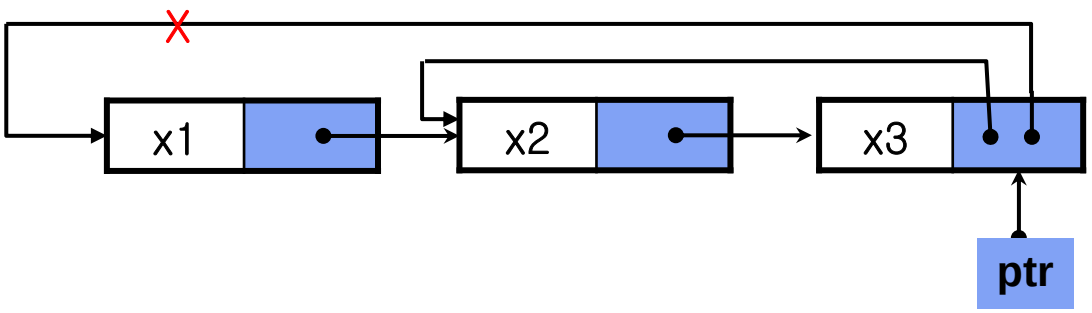


Q/A

(실험 1) circularlist.c 프로그램 실행

(실험 2) 원형 연결리스트의 맨 앞에 노드를 삭제하는 알고리즘 작성

```
void  delete_first(list_ptr *ptr,list_ptr node)
{
        if(IS_EMPTY(*ptr))
        { printf("원소없음…₩n");  /* 빈 리스트 경우 */  }
        else
        {
                /* 여기 작성 */
        }
}
```



2. 이중 연결리스트(Doubly Linked List)

- 이중 연결리스트란?

  연결리스트의 각 노드에 링크를 2개를 만들어 하나는 뒷 노드를 하나는 앞 노드를 가리키는 연결리스트이다.

  단순 연결리스트(Singly Linked List)는 다음과 같은 경우 불편한 점이 있다. 단순연결리스트의 노드 포인터를 알고 있을 때 노드의 다음 노드는 찾아갈 수 있지만, 노드의 앞 노드는 찾아 갈 수 없다. 앞 노드를 찾아가려면 리스트의 처음부터 따라가야 한다.
  (즉 노드의 위치 만큼 링크를 따라가야 한다 O(n)).

  즉, 임의의 노드의 앞 노드를 찾아가는 데 시간이 걸린다.

  단순 연결리스트의 각 노드에 앞 노드의 링크를 저장하면 ?

  단순 연결리스트의 각 노드에 두개의 노드 포인터를 두어서 하나를 앞쪽 노드를 하나는 뒤 쪽 노드를 가리키어 앞 뒤 어느 방향으로나 갈 수 있도록 만든 리스트이다.

단순 연결리스트의 문제점

- 한 쪽 방향으로만 노드를 따라간다.(앞에서 뒤로)
- 앞 노드를 찾으려면 처음 부터 링크를 따라와야 한다.
- 임의의 노드를 지우기 어렵다
  (노드를 지우려면 앞, 뒤 노드 포인터를 알아야 한다.)

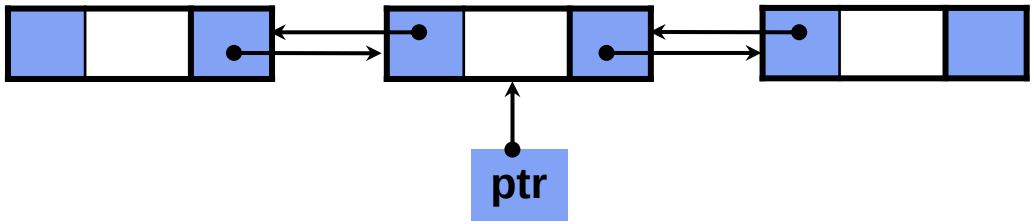
해결 방법

- 원형 연결리스트
- 이중 연결리스트
- 이중 원형 연결리스트(doubly linked circular list)

```
/* 이중 연결리스트를 위한 노드의 선언 */
struct dnode {
      struct dnode *llink;
      element item;
      struct dnode *rlink;
};

typedef struct dnode node;
typedef node  *node_ptr;
```

ptr이 이중 연결리스트의 임의의 노드를 가리킨다고 가정하자
이중 연결리스트에서는 다음의 식이 성립한다.

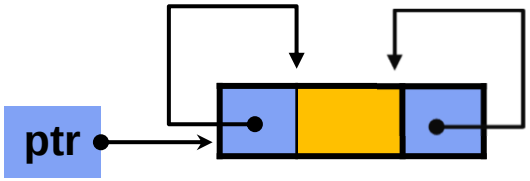
ptr = ptr->llink->rlink = ptr->rlink->llink



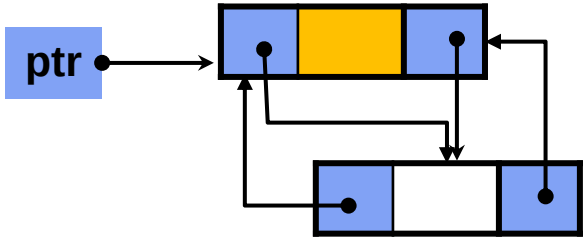
- 이중 연결리스트의 처음 시작 때는(노드가 하나도 없는 경우) 가상의 노드에서 부터 시작한다. 이 데이터가 없는 가상의 노드를 dummy node 라고 한다. 보통은 head node(머리 노드)라고 한다.

  - 비어있는 리스트 경우 head node만 있다.
  - 프로그램의 편의를 위하여 만들어 놓은 것이다.
  - 노드와 같은 자료구조이나 데이터 값이 없다.

비어있는 이중 연결 리스트

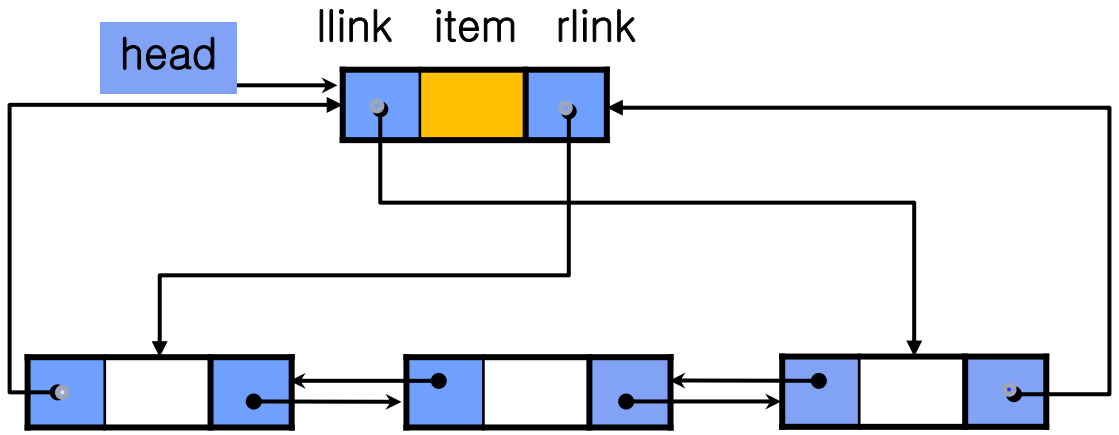


노드가 한 개 추가된 후의 리스트

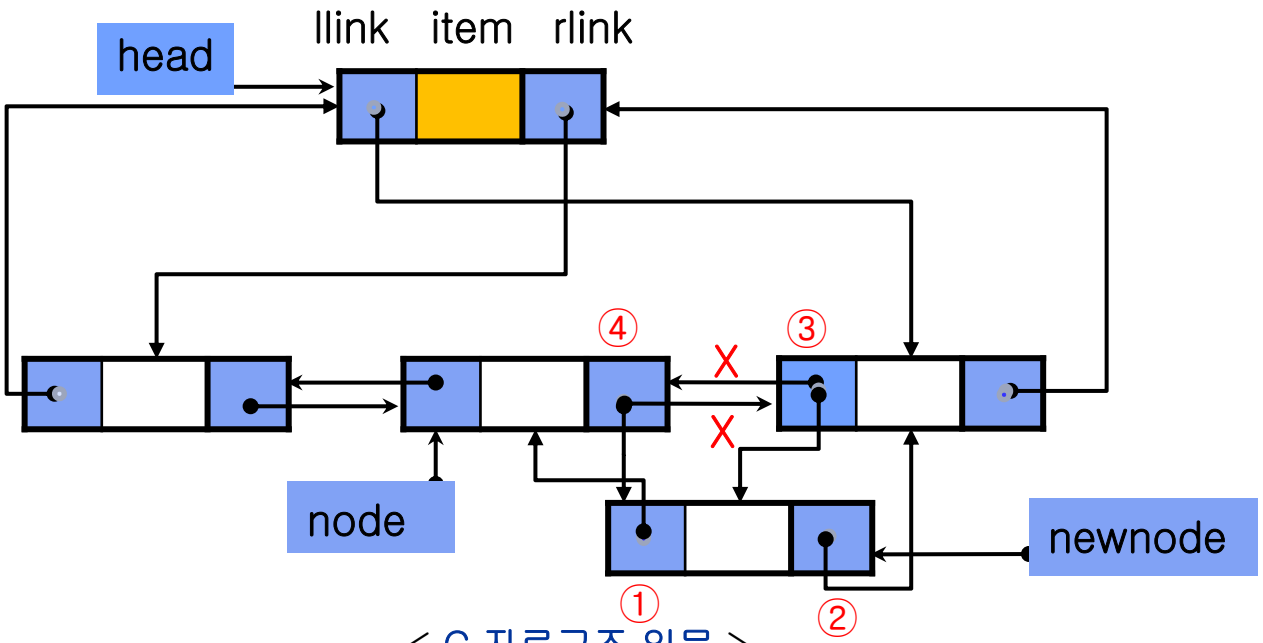


머리노드를 가진 이중 원형 연결리스트의 모양

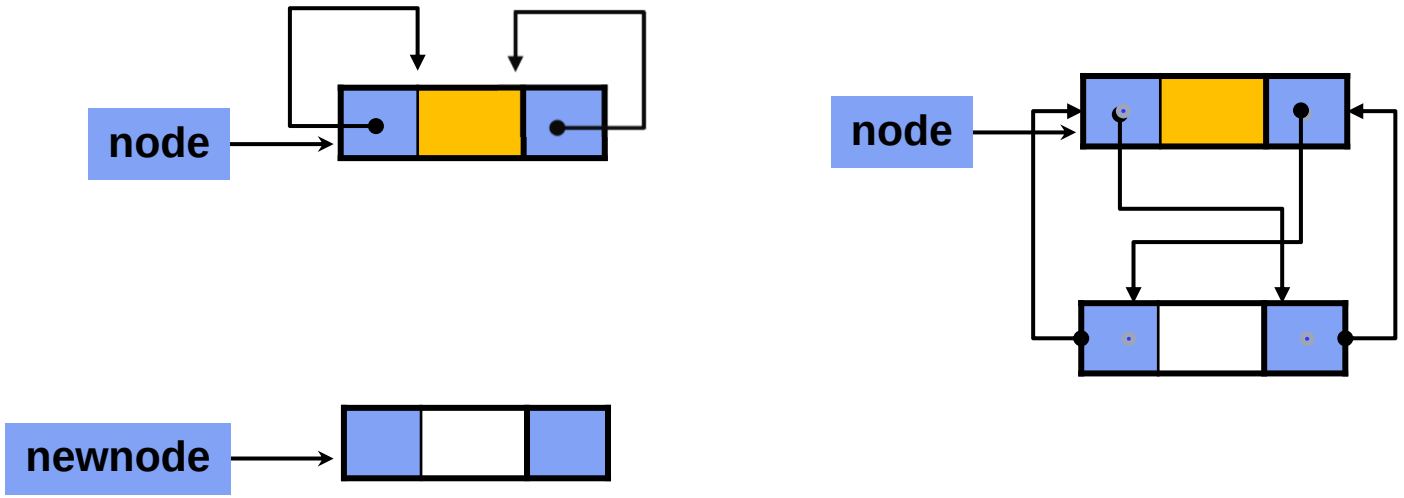
- 헤드노드의 llink 는 앞노드(리스트의 마지막 노드를 가리킨다)
  rlink는 다음 노드(리스트의 첫 노드를 가리킨다)



```
/* 이중 원형 연결리스트에 노드를 삽입하는 프로그램
삽입할 노드의 바로 앞 노드만 알면 된다.*/

void dinsert(node_ptr node,node_ptr newnode) {
        /* node의 오른쪽에 newnode를 삽입 */
        newnode->llink = node; /* ① */
        newnode->rlink = node->rlink; /* ② */
        node->rlink->llink = newnode; /* ③ */
        node->rlink = newnode; /* ④ */
}
```



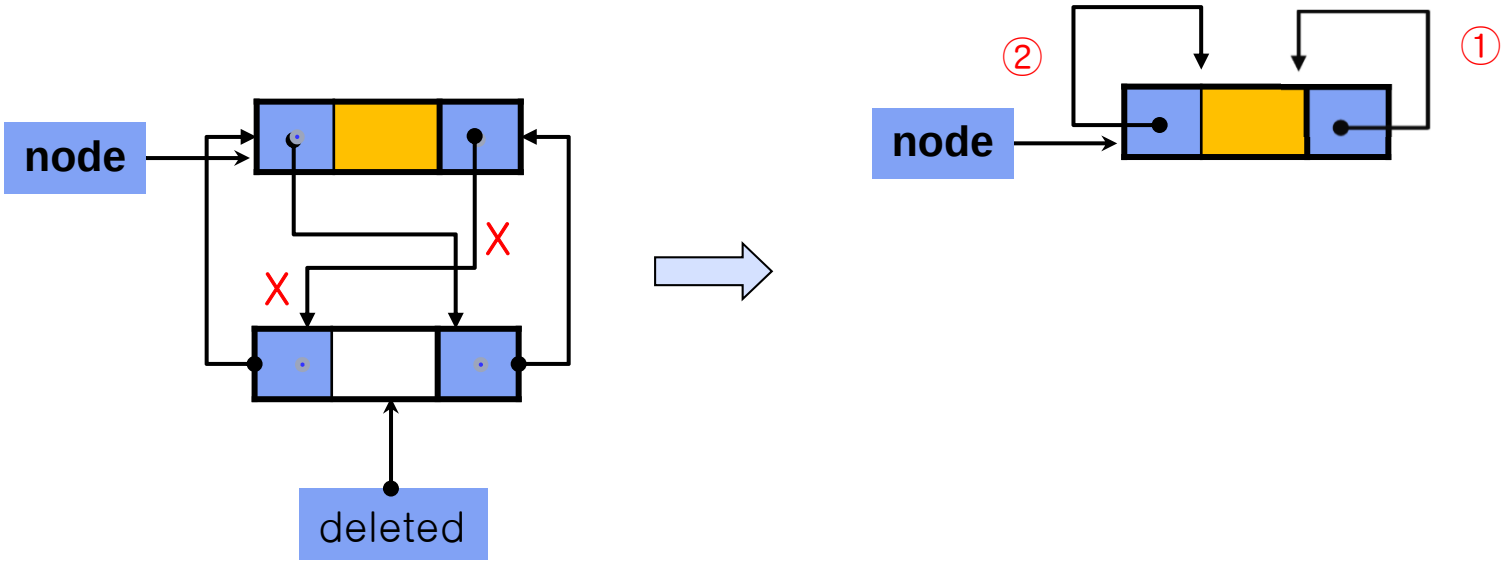
비어있는 이중 연결리스트에 노드의 삽입



이중 연결리스트에서 노드의 삭제

```
void ddelete(node_ptr node,node_ptr deleted) {
        /* 이중 연결리스트에서의 deleted 노드 삭제 */
        if(node == deleted)
                printf(" 머리노드 삭제 금지 ...₩n");
        else {
                deleted->llink->rlink = deleted->rlink; /* ① */
                deleted->rlink->llink = deleted->llink; /* ② */
                free(deleted); /* ③ */
        }
}
```

한 개의 노드를 가진 이중연결리스트에서 노드의 삭제



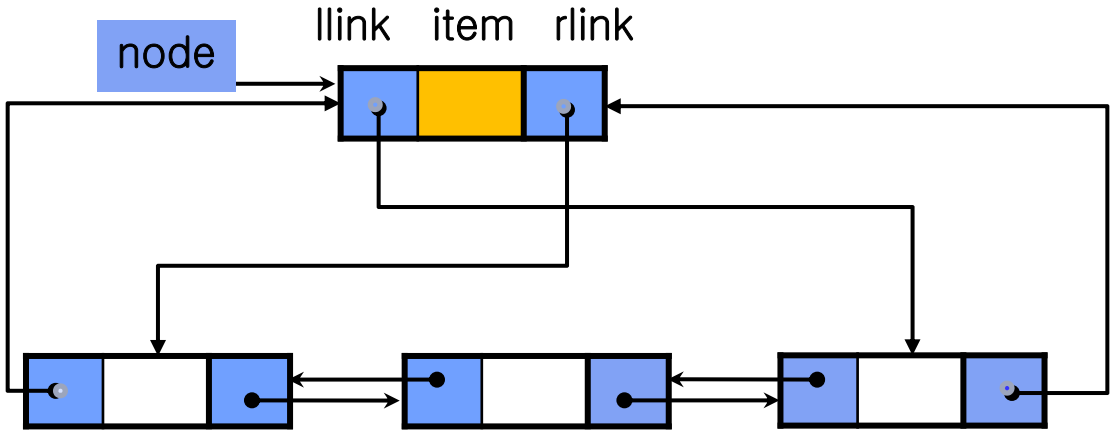
③ free() 노드를 반환

Q/A

(실험 1) dlinkedlist.c 프로그램 실행

(실험 2) 이중 연결리스트의 노드의 수를 세는 알고리즘 작성

```
int countdouble(node_ptr node)
{
    if(node->rlink==node) return 0;
    else
    {
        /* ? */
    }
}
```



3. 연결리스트 알고리즘들

3-1. 연결 리스트를 역순으로 만들기

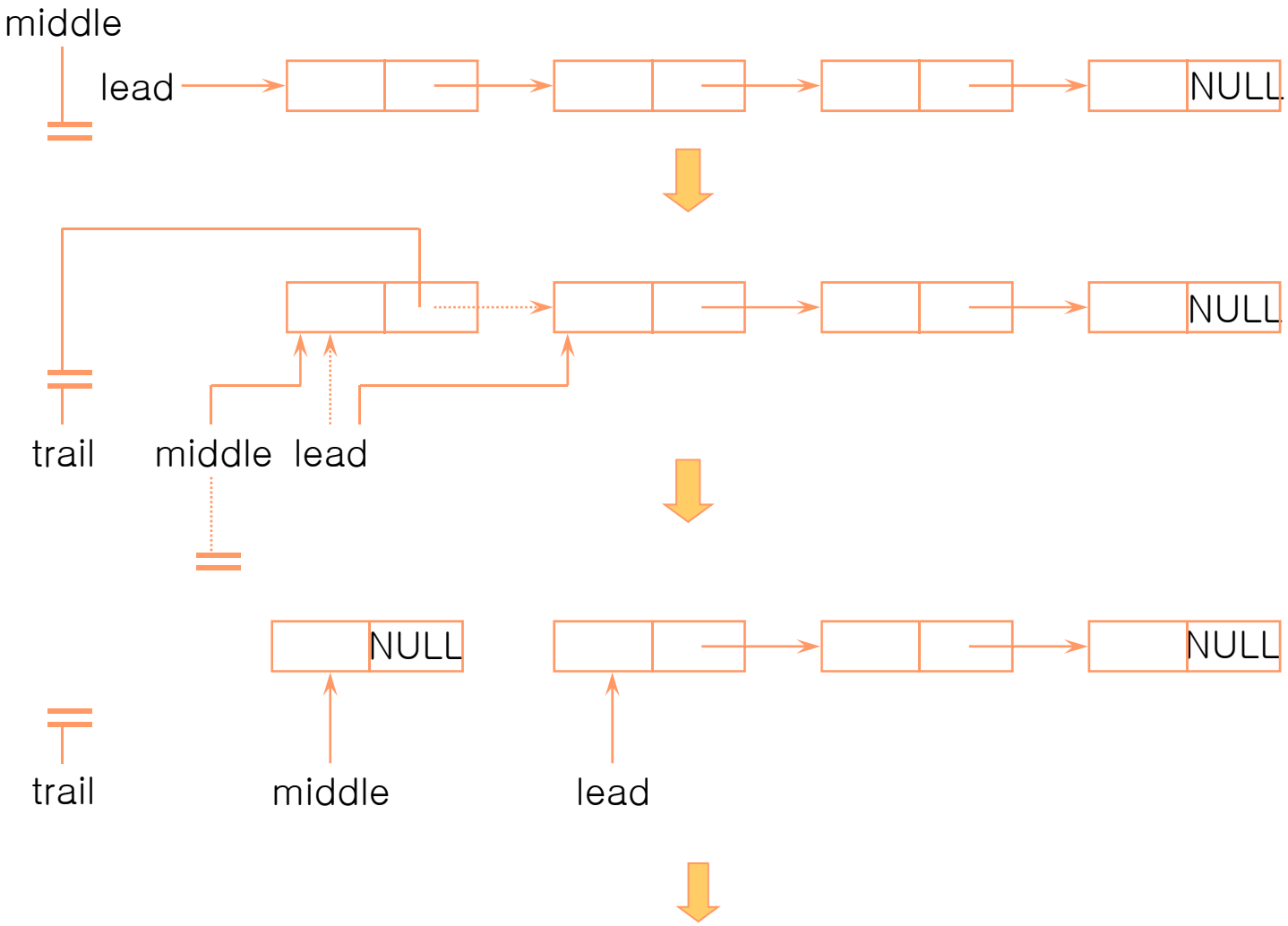
연결 리스트의 역 배열을 위한 프로그램

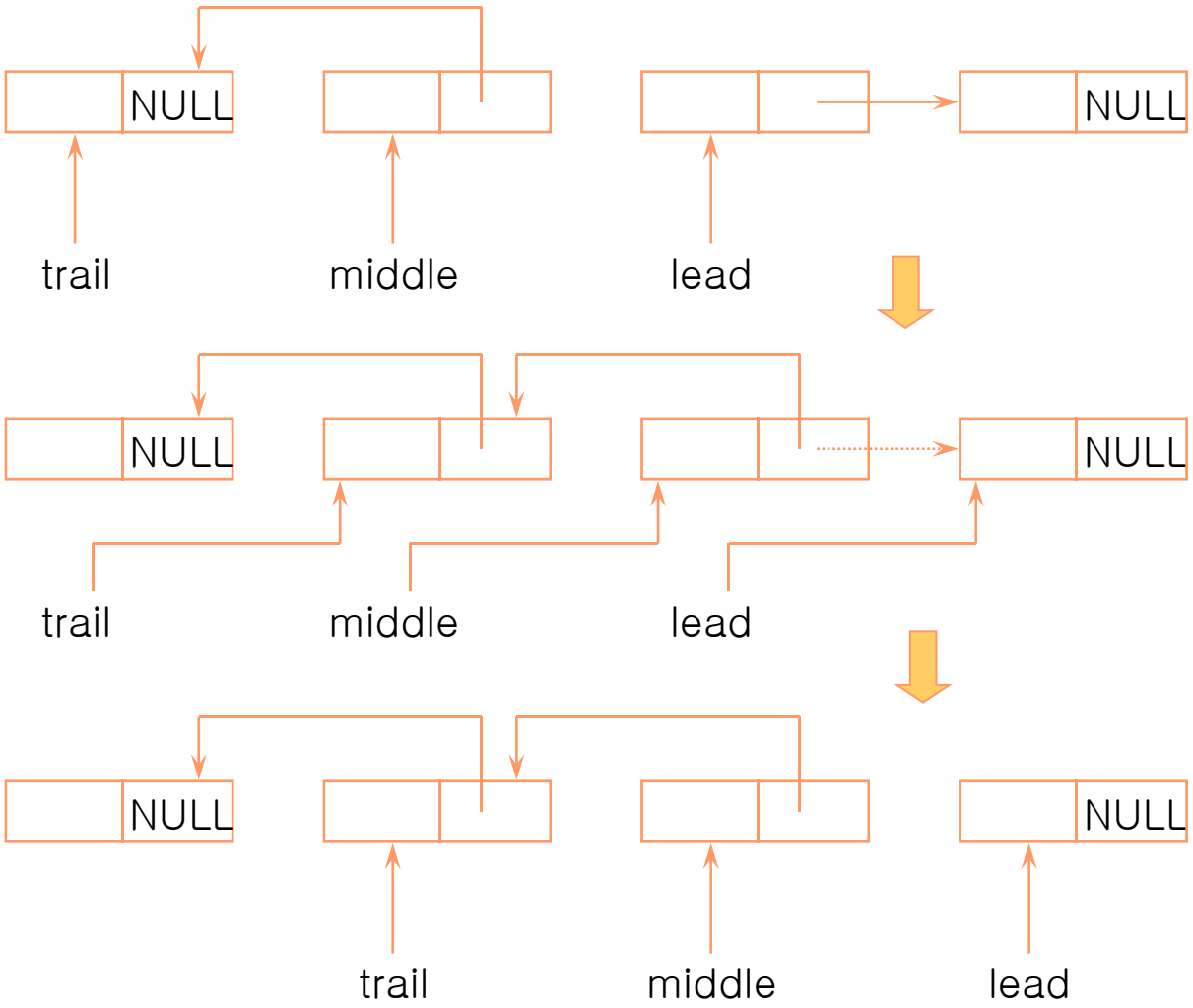
```
/* 연결리스트의 링크를 거꾸로 만드는 프로그램을 작성하여 본다 */
/* 리스트 노드 선언 부분 */
  struct node {
      char data;
      struct node * link;
  };

  typedef struct node list_node;
  typedef list_node * list_ptr;
```

연결 리스트의 역 배열을 위한 프로그램

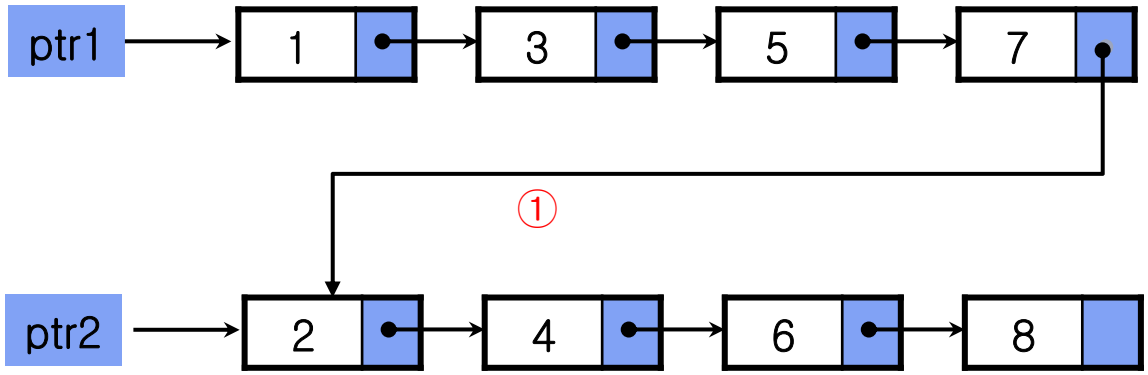
수행시간 : O(n)

```
list_ptr invert(list_ptr lead) {
        list_ptr middle, trail;
        middle = NULL;
        while(lead) {
                trail = middle;
                middle = lead;
                lead = lead->link;
                middle->link = trail;
        }
        return middle;
}
```


middle
lead
NULL
NULL
trail
middle
lead
NULL
NULL
trail
middle
lead


NULL
NULL
trail
middle
lead
NULL
NULL
trail
middle
lead
NULL
NULL
trail
middle
lead

3-2. 두 개의 연결리스트를 한 개의 연결리스트로 연결



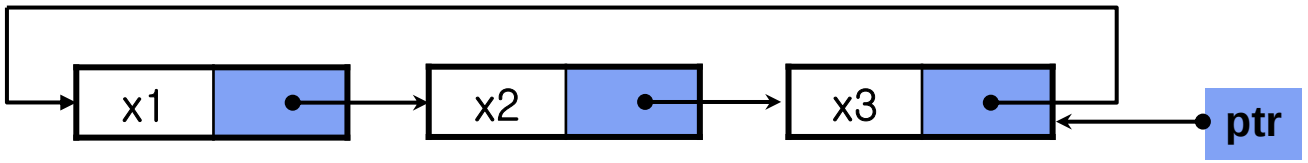
2개의 연결 리스트 연결 프로그램

```
/* 두개의 리스트를 연결해서 새로운 리스트 만든다 */
/* ptr1에 ptr2를 붙인다 */

list_ptr concat(list_ptr ptr1,list_ptr ptr2) {
        list_ptr temp;
        if( !ptr1 ) return ptr2;
        else {
                if( ptr2 ) {
                        for(temp=ptr1; temp->link; temp=temp->link);
                        temp->link = ptr2; ① }
                return ptr1;
        }
}
```

3-3. 원형연결리스트의 노드 개수 세기

```
/* 아래의 연결리스트의 ptr 값을 주면 결과는 3을 반환한다.*/
int length(list_ptr ptr) {
        list_ptr temp; int count = 0;
        if(ptr) {
                temp = ptr;
                do {
                        count++; temp = temp->link;
                } while(temp != ptr);
        }
        return count;
}
```

원형 리스트의 노드의 개수 세기



Review

⊙ 단순 연결리스트의 단점을 극복하는 리스트 구조는 다음과 같다.

(1) 연결리스트의 끝에서 작업하기 편리한 원형 연결리스트
연결리스트에서 마지막 노드가 끝 노드를 가리키게하며 리스트의 포인터를 맨끝 노드를 가리킨다.

(2) 임의의 노드의 앞뒤 작업을 편리하게 하기위한 이중 연결리스트
연결리스트의 각 노드에 앞과 뒤로가는 링크를 만들어 놓은 리스트이다.

(3) 원형과 이중 연결리스트의 두가지 장점을 동시에 갖는 이중 원형 연결리스트가 있다. 원형과 이중 연결리스트 두 가지 구조를 동시에 갖는다. 즉 각 노드에 두개의 링크를 두면서 마지막 노드와 첫 노드도 연결시킨다. 또 머리노드라는 가상의 노드를 운영한다.

⊙ 연결리스트 프로그래밍은 약간 복잡하다. 링크와 노드의 구조를 잘 이해하면 기계적이다.