연결리스트 -
원형연결리스트 및 양방향연결리스트

5주차-강의

남춘성

SUNG KYUN KWAN UNIVERSITY

원형연결리스트(Linked List)의 개념적 이해 - I

SUNG KYUN KWAN UNIVERSITY

- 단순 연결리스트의 문제점
  - 단순 연결리스 구현의 어려움 (느껴봤음!!)
  - 헤드를 통해 접근만 허용(모든 데이터를 읽을 경우, tail로는 불가)
  - 한번 지나간 Before 혹은 pred 와 같은 노드 접근 불가
    - → 순환을 통해 이를 해결할 수 있는 방법 : 원형 연결리스트

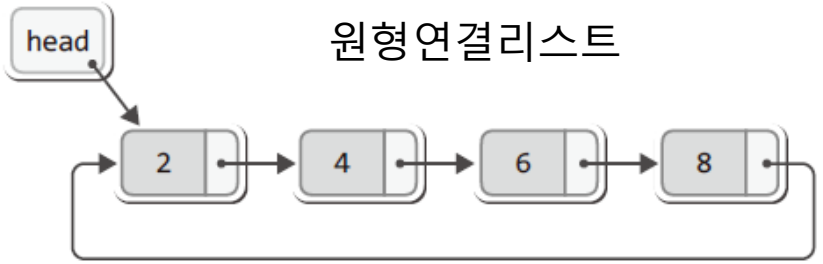
- 원형 연결리스트
  - 단순 연결 리스트에서 마지막 노드가 리스트의 첫 번째 노드를 가리키게 하여 리스트의 구조를 원형으로 만든 연결 리스트
    - 단순 연결 리스트의 마지막 노드의 링크 필드에 첫 번째 노드의 주소를 저장하여 구성
    - 링크를 따라 계속 순회하면 이전 노드에 접근 가능
    - 머리와 꼬리를 가리키는 포인터 변수 필요없음(하나의 포인터만으로도 해결 가능)

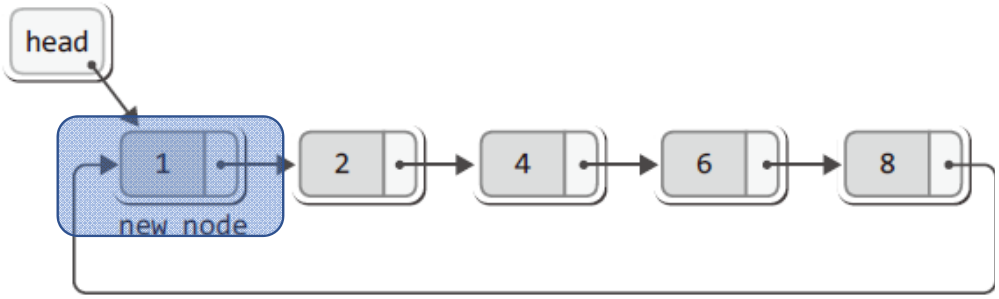
원형연결리스트(Linked List)의 개념적 이해 - II

SUNG KYUN KWAN UNIVERSITY

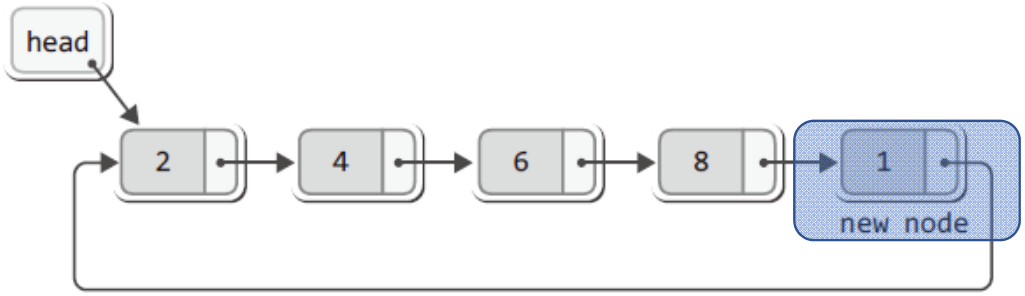
- 단순 연결리스트와 원형 연결리스트 비교 예

단순연결리스트





원형연결리스트 헤드삽입

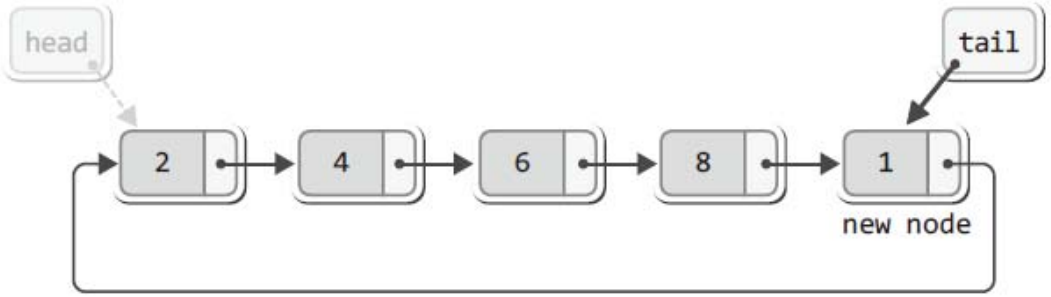


원형연결리스트 꼬리삽입

원형연결리스트(Linked List)의 개념적 이해 - III

SUNG KYUN KWAN UNIVERSITY

- 변형된 원형 연결 리스트
  - 결국 tail이 head와 연결되어 tail을 통해 head가 접근 가능함.



Tail은 tail
Head는 tail의 다음 원소

원형연결리스트(Linked List)의 개념적 이해 - IV

SUNG KYUN KWAN UNIVERSITY

- 변형된 원형 연결 리스트
  - 구현비교
    - Lfirst : 이전 단순 연결리스트와 기능이 동일
    - **Lnext** : 원형 연결리스트를 순환하는 형태이기 때문에 변경
    - **Lremove** : 이전 단순 연결리스트와 기능이 동일 (더미가 없는 경우 새로 구현)
    - **Linsert** : head와 tail에 삽입 가능하도록 두 개의 함수 정의
    - Sort : 모두 삭제
    - Others : 이전 단순 연결리스트와 동일

원형연결리스트(Linked List)의 개념적 이해 - V

SUNG KYUN KWAN UNIVERSITY

- 변형된 원형 연결 리스트 자료형
  - 꼬리로 표현할 수 있기에 tail만 정의 : Node * tail
  - 현재의 위치 표현 : Node * cur;
  - 삽입 삭제를 위한 이전 노드 표현 : Node * before;
  - 자료의 수 표현 : numOfData

```
typedef struct _CLL {
        Node * tail;
        Node * cur;
        Node * before;
        int numOfData;
}CList;
```

원형연결리스트(Linked List)의 ADT 및 구현 - 초기화

SUNG KYUN KWAN UNIVERSITY

- 원형 리스트 초기화 : void ListInit(List * plist);
  - tail 값 설정 : NULL
  - cur 값 설정 : NULL
  - before 값 설정 : NULL
  - numOfData 값 설정 : 0

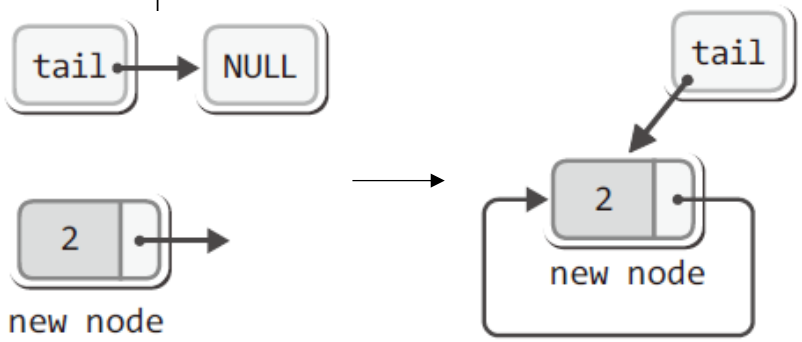
```
void ListInit(List * plist) {
        plist->tail = NULL;
        plist->cur = NULL;
        plist->before = NULL;
        plist->numOfData = 0;
}
```

원형연결리스트(Linked List)의 ADT 및 구현 - 삽입I

SUNG KYUN KWAN UNIVERSITY

- 원형 연결리스트 삽입 : void Linsert(List * plist, Data data)
  - 노드가 처음이라면
    - 새 노드 추가 후 자기자신을 가르키도록 함
  - 노드가 처음이 아니라면
    - 새 노드 추가를 위해 tail의 위치를 조정하여 추가

```
void LInsert(List * plist, Data data) { //tail에 노드 추가 (tail은 뒤쪽)
        Node * newNode = (Node *)malloc(sizeof(Node));
        newNode->data = data;

        if (plist->tail == NULL) {// 첫번째 노드라면
                plist->tail = newNode;
                newNode->next = newNode;
        } else {
                .....
        }

        (plist->numOfData)++;
}
```

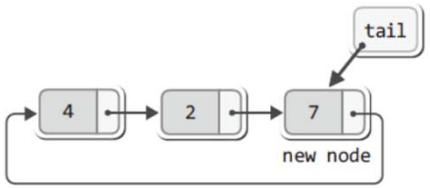


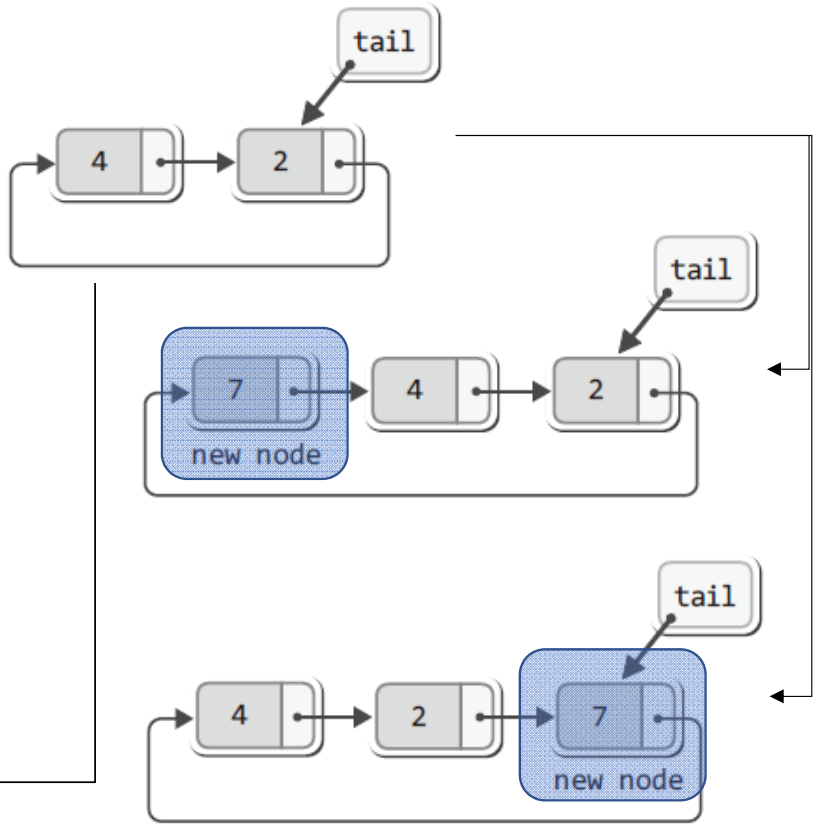
원형연결리스트(Linked List)의 ADT 및 구현 - 삽입II

SUNG KYUN KWAN UNIVERSITY

- Tail에 삽입하는 경우와 head에 삽입하는 경우
  - 머리에 삽입 : void LInsertFront(List * plist, Data data);
  - 꼬리에 삽입 : void Linsert(List * plist, Data data)

```
else { //LInsertFront
        newNode->next = plist->tail->next;
        plist->tail->next = newNode;
}

vs

else { //LInsert
        newNode->next = plist->tail->next;
        plist->tail->next = newNode;.
        plist->tail = newNode;// 즉, tail의 위치만 조정하면 됨.

}
```



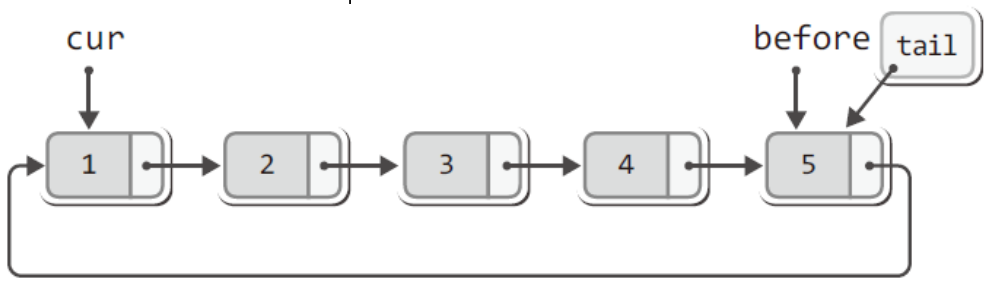


원형연결리스트(Linked List)의 ADT 및 구현 - 조회I

SUNG KYUN KWAN UNIVERSITY

- 저장된 데이터의 탐색 초기화 : int Lfirst(List * plist, LData * pdata);
  - Before는 더미 노드를 가리키게 함
  - Cur는 첫 번째 노드를 가리키게 함
  - 첫 번째 노드 데이터 전달

```
int LFirst(List * plist, LData * pdata) {
        if (plist->head->next == NULL)
                return FALSE;

        plist->before = plist->tail;  // head를 tail로만 변경
        plist->cur = plist->tail->next; // head를 tail로만 변경


        *pdata = plist->cur->data;

        return TRUE;
}
```



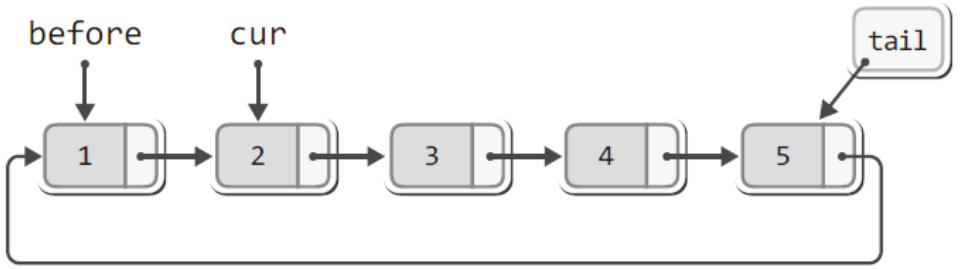
원형연결리스트(Linked List)의 ADT 및 구현 - 조회II

SUNG KYUN KWAN UNIVERSITY

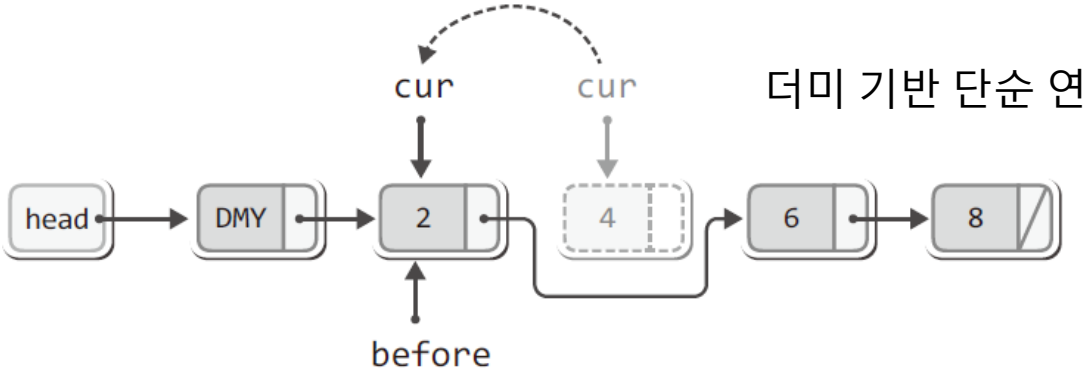
- 다음 데이터의 참조(반환) : int LNext(List * plist, LData * pdata);
  - Cur가리키던 것을 before가 가리킴
  - Cur는 다음 노드를 가리킴
  - 노드 데이터 전달 (단순연결리스트와 같음)

다만 원형 연결리스트이 끝을 검사하는 코드를 구현해야 함.

```
int LNext(List * plist, LData * pdata) {
        if (plist->cur->next == NULL)
                return FALSE;

        plist->before = plist->cur;
        plist->cur = plist->cur->next;

        *pdata = plist->cur->data;

        return TRUE;
}
```

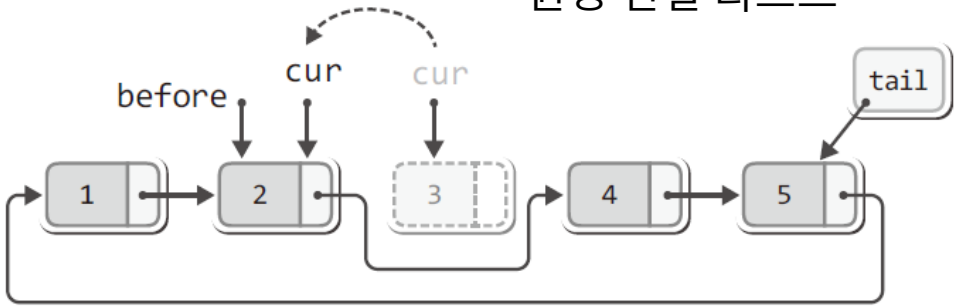


원형연결리스트(Linked List)의 ADT 및 구현 - 삭제I



더미 기반 단순 연결리스트

원형 연결 리스트



위와 다른 상황
- 삭제할 노드를 tail이 가리키는 경우 발생
  1) 그 노드가 마지막 노드인 경우
  2) 그 노드가 마지막이 아닌 경우

원형연결리스트(Linked List)의 ADT 및 구현 - 삭제I

SUNG KYUN KWAN UNIVERSITY

- 데이터 삭제 (바로 이전에 참조(반환)가 이루어진) : LData LRemove(List * plist) 과 비교

```
LData LRemove(List * plist) {
	Node * rpos = plist->cur;
	LData rdata = rpos->data;

	plist->before->next = plist->cur->next;
	plist->cur = plist->before;

	free(rpos);
	(plist->numOfData)--;

	return rdata;

}
```

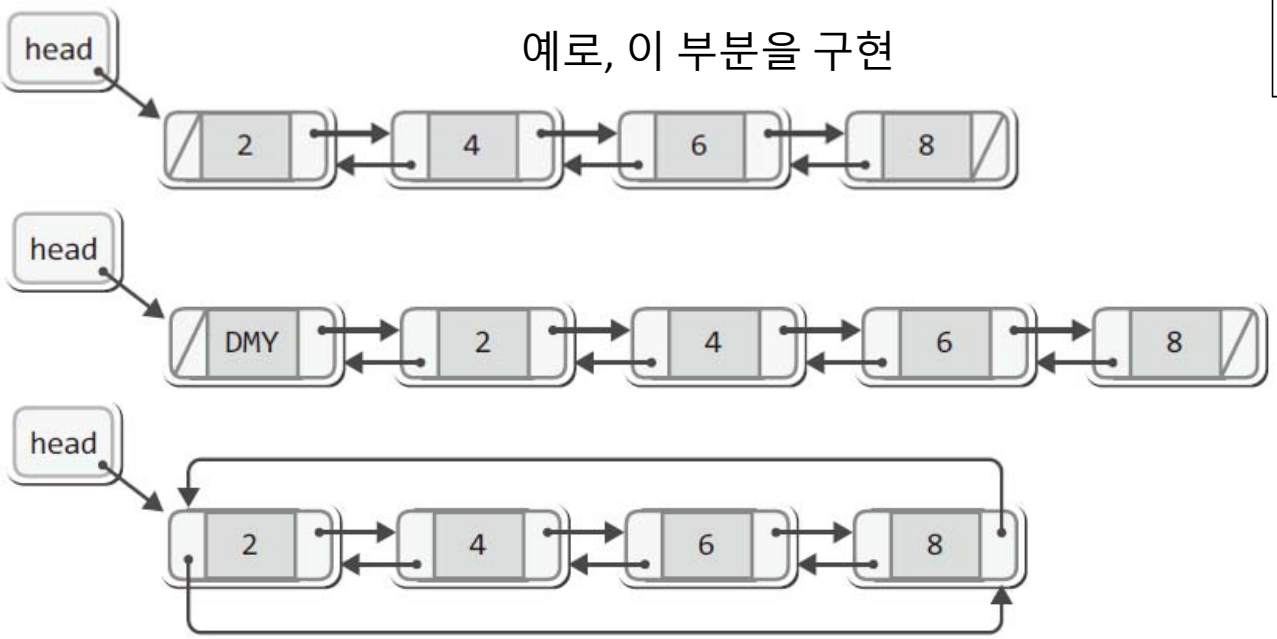
```
Data LRemove(List * plist) {
	Node * rpos = plist->cur;
	Data rdata = rpos->data;

	if (rpos == plist->tail) { //삭제 Node가 tail인 경우
		if (plist->tail == plist->tail->next) //tail이 마지막
			plist->tail = NULL;
		else  //tail만 변경하는 경우
			plist->tail = plist->before;
	}
	plist->before->next = plist->cur->next;
	plist->cur = plist->before;

	free(rpos);
	(plist->numOfData)--;

	return rdata;
}
```

양방향 연결리스트(Linked List)의 개념적 이해 - I

SUNG KYUN KWAN UNIVERSITY

- 양방향 연결리스트(Doubly Linked list)
  - 양쪽 방향으로 순회할 수 있도록 노드를 연결한 리스트
    - Before를 유지할 필요가 없음.
    - 덜 복잡한 과정을 통해 함수 구현
      - 현재 노드 중심으로 생각

```
typedef struct _node{
        Data data;
        struct _node * next;
        struct _node * prev;
}Node;
```



예로, 이 부분을 구현

양방향 연결리스트(Linked List)의 ADT 및 구현 - I

SUNG KYUN KWAN UNIVERSITY

- 이전 노드 탐색 (추가) : int Lprevious(List * plist, Data * pdata);
  - 실제 Lprevious. 는 Lnext와 이동 방향만 다름
  - 또한, Lfirst와 Lnext도 같음

```
int LFirst(List * plist, Data * pdata) {
        if (plist->head == NULL) {
                return FALSE;
        }

        plist->cur = plist->head;
        *pdata = plist->cur->data;

        return TRUE;
}
```

```
int LNext(List * plist, Data * pdata) {
        if (plist->cur->next == NULL) {
                return FALSE;
        }

        plist->cur = plist->cur->next;
        *pdata = plist->cur->data;

        return TRUE;
}
```

```
int LPrevious(List * plist, Data * pdata) {
        if (plist->cur->prev == NULL) {
                return FALSE;
        }

        plist->cur = plist->cur->prev;
        *pdata = plist->cur->data;

        return TRUE;
}
```

ADT를 통한 비교

양방향 연결리스트(Linked List)의 ADT 및 구현 - II

SUNG KYUN KWAN UNIVERSITY

- 새 노드 삽입 : void Linsert(List * plist, Data data); - 더미 없을 경우
  - 첫 노드 삽입 하는 경우
  - 그 이후 삽입 하는 경우

```
void LInsert(List * plist, Data data) {
        Node * newNode = (Node*)malloc(sizeof(Node));
        newNode->data = data;
// plist->head 가 NULL 인 경우 즉, head 다음에 삽입하는 경우
// plist->head가 NULL이 아닌경우, 즉, head가 가리키는 곳에 삽입하는 경우
        newNode->next = plist->head

        if (plist->head != NULL)
                plist->head->prev = newNode;

        newNode->prev = NULL;
        plist->head = newNode;

        (plist->numOfData)++;
}
```