연결리스트 - 단순연결리스트

4주차-강의

남춘성

SUNG KYUN KWAN UNIVERSITY

연결리스트(Linked List)의 개념적 이해 - I

SUNG KYUN KWAN UNIVERSITY

- 배열(순차 자료구조) 자료형의 문제점
  - 삽입연산이나 삭제연사 후에 연속적인 물리 주소를 유지하기 위해서 원소들을 이동시키는 추가적인 작업과 시간 소요
    - 원소들의 이동 작업으로 인한 오버헤드는 원소의 개수가 많고 삽입 및 삭제 연산이 많이 발생하는 겨우 성능상의 문제 발생
  - 순차 자료구조는 배열을 이용하여 구현하기 대문에 배열이 갖고 있는 메모리 사용의 비효율성 문제를 그대로 가짐 (길이 변경 불가)
    - 즉, 배열의 길이를 넘어서거나 필요한 메모리의 크기에 유연하게 대체하지 않음
  - 순차 자료구조에서의 연산 시간에 대한 문제와 저장 공간에 대한 문제를 개선한 자료 표현 방법 필요

연결리스트(Linked List)의 개념적 이해 - II

SUNG KYUN KWAN UNIVERSITY

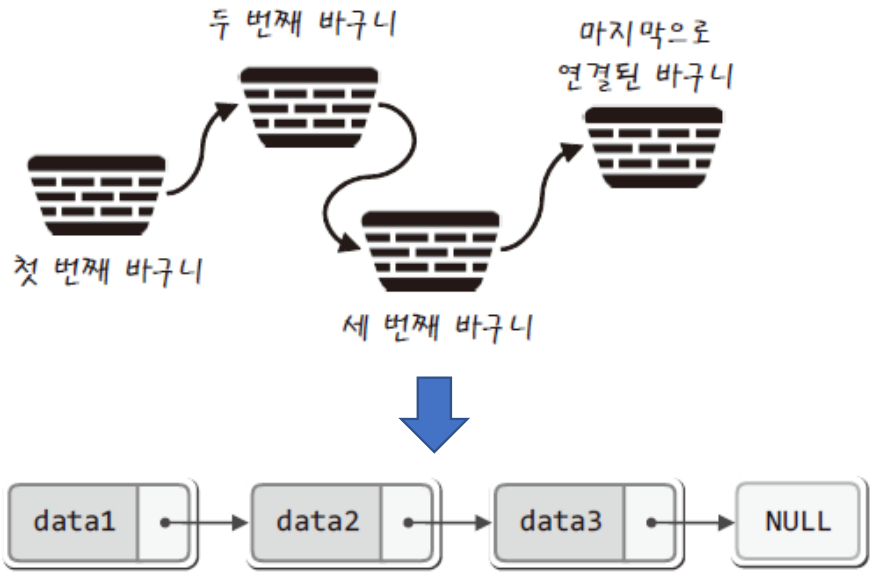
- 연결 자료구조
  - 자료의 논리적인 순서와 물리적인 순서가 일치하지 않는 자료구조
    - 각 원소에 저장되어 있는 다음 원소의 주소에 의해 순서가 연결되는 방식
      - **물리적인 순서를 맞추기 위한 오버헤드가 발생하지 않음**
    - 여러 개의 작은 공간을 연결하여 하나의 전체 자료구조를 표현
      - **크기 변경이 유연하고 더 효율적으로 메모리를 사용**
  - 연결 리스트
    - 리스트를 연결 자료구조로 표현한 구조
    - 연결하는 방식에 따라
      - **단순 연결 리스트**
      - **원형 연결 리스트**
      - **이중 연결 리스트**
      - **이중 원형 연결 리스트**

연결리스트(Linked List)의 개념적 이해 - III

SUNG KYUN KWAN UNIVERSITY

- 연결은 주소!
  - 바구니가 다음 바구니를 가르키는 형태
    - 바구니에 있는 원소 : 데이터
    - 바구니를 가리키는 원소 : 주소

```
typedef struct _node {

  int data;                 //원소
  struct _node * next;      // 연결 주소

} Node;
```



연결리스트(Linked List) 삽입 및 삭제 - I

SUNG KYUN KWAN UNIVERSITY

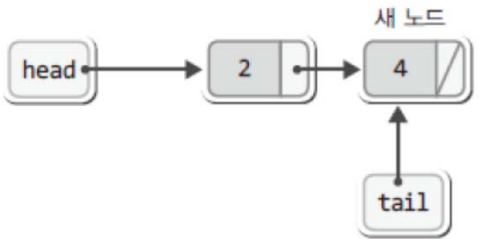
- head, tail, cur
  - head : 연결리스트의 시작
  - Tail : 연결리스트이 마지막
  - Cur : 연결리스트의 현재

```
Int main(void)
{
        Node * head = NULL:
        Node * tail = NULL:
        Node * cur = NULL;

        Node * newNode = NULL;
        int readData;

        ….


}
```

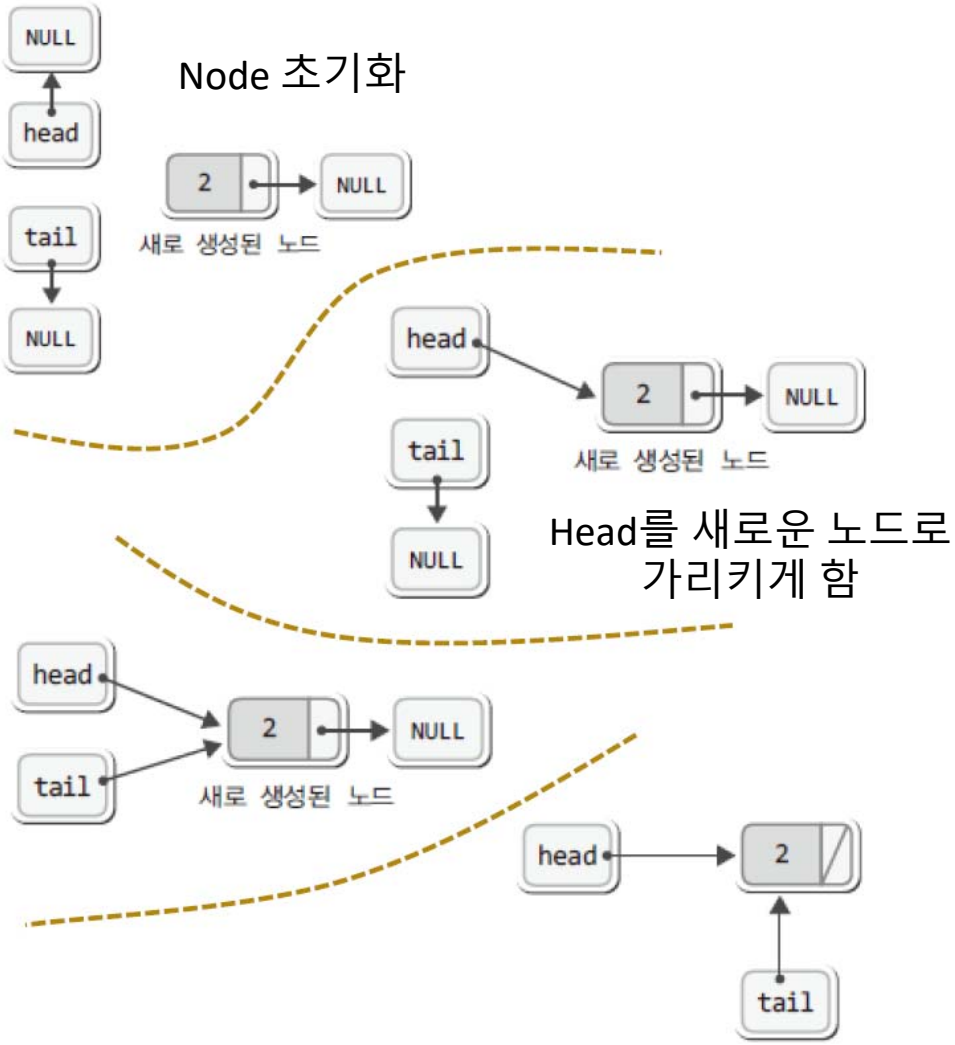
연결리스트(Linked List) 삽입 및 삭제 – II

SUNG KYUN KWAN UNIVERSITY

- 삽입시
  - 첫번째 원소인지?
  - 그 이후 원소인지?



새노드를 tail을 뒤에 taill을
맨 마지막을 가리키게 함



Tail을 새로운 노드로
가리키게 함

연결리스트(Linked List) 삽입 및 삭제 – III

SUNG KYUN KWAN UNIVERSITY

- 삽입과정 코드화

```
//데이터를 입력 받는 과정//
while (1){
	printf("자연수 입력 : ");
	scanf_s("%d", &readData);
	if (readData < 1)
		break;

	newNode = (Node *)malloc(sizeof(Node));
	newNode->data = readData;
	newNode->next = NULL;

	if (head == NULL)
		head = newNode;
	else
		tail->next = newNode;

	tail = newNode;
}
```

Node 생성

첫 Node 삽입 : 첫 번째 원소 → head가 첫 원소 가리킴

다음 Node 삽입 : 첫 번째 원소 이후

tail을 삽입된 원소로 가리킴

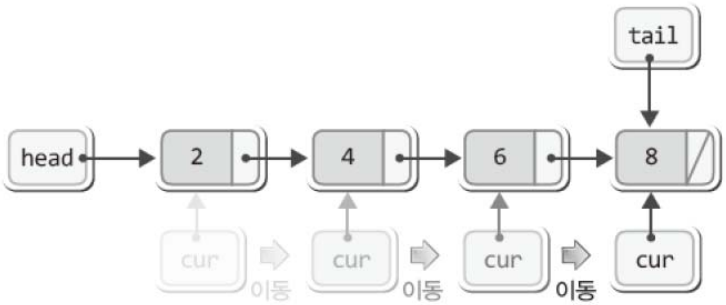
연결리스트(Linked List) 삽입 및 삭제 – IV

- 연결리스트 읽기

```
printf("입력 받은 데이터의 전체출력! \n");
if (head == NULL) {
        printf("저장된 자연수가 존재하지 않습니다. \n");
}
else {
        cur = head;
        printf("%d ", cur->data);

        while (cur->next != NULL) {
                cur = cur->next;
                printf("%d ", cur->data);
        }
}
```

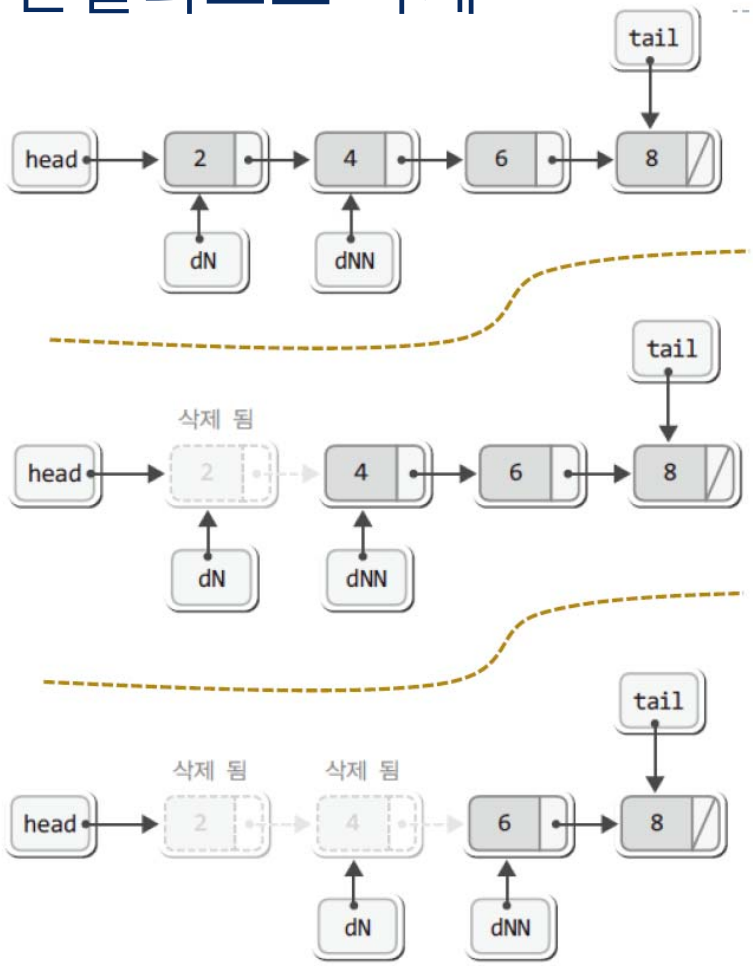
Cur의 첫 위치를 head가 가리키는 원소로 지정

Cur의 다음 원소가 NULL이 아니면, 원소가 있으므로 출력



연결리스트(Linked List) 삽입 및 삭제 – V

- 연결리스트 삭제



노드 접근을 위해 delNode(dN : 삭제할 노드), delNextNode(dNN: 삭제할 노드가 가르키는 노드) 를 설정 (delNode만 삭제하면 그 다음 노드에 접근 불가능)

dN 삭제(free(delNode)를 통해 완전삭제)

dN 을 dNN으로 설정해주고, dN을 통해 dNN새로 설정
Tail이 올때까지 혹은 dNN의 다음 값이 NULL 될때까지 삭제

연결리스트(Linked List) 삽입 및 삭제 – VI

SUNG KYUN KWAN UNIVERSITY

- 연결리스트 삭제

```
if (head == NULL) {
        return 0;
}
else {
        Node * delNode = head;
        Node * delNextNode = head->next;

        printf("%d을(를) 삭제합니다. \n", head->data);
        free(delNode);

        while (delNextNode != NULL) {
                delNode = delNextNode;
                delNextNode = delNextNode->next;

                printf("%d을(를) 삭제합니다. \n", delNode->data);
                free(delNode);
        }
}
```

dN, dNN 설정

dN 위치 노드 삭제

dN을 이동하며 삭제

단순 연결 리스트의 ADT 설정 및 구현 - I

SUNG KYUN KWAN UNIVERSITY

- ADT 설정 (3주차 참조)
  - 리스트 초기화 : void ListInit(List * plist);
    - 리스트 생성 후 제일 먼저 호출되는 함수
    - 초기화할 리스트의 주소 값을 인자로 전달
  - 리스트에 데이터 저장(삽입) : void Linsert(List * plist, Ldata data);
    - 리스트에 데이터 자장, 삽입할 매개변수 data를 리스트에 저장
  - 저장된 데이터의 탐색 및 탐색 초기화 : int Lfirst(List * plist, LData * pdata);
    - 데이터의 참조를 위해 초기화가 진행
    - 현재의 위치를 새로 설정(탐색의 첫번째 과정)
    - 첫 번째 데이터가 pdata가 가리키는 메모리에 저장하여 main에서 사용
    - 참조 성공시 TRUE(1), 실패 시 FALSE(0) 반환

단순 연결 리스트의 ADT 설정 및 구현 - II

SUNG KYUN KWAN UNIVERSITY

- ADT 설정 (3주차 참조)
  - 다음 데이터의 참조(반환) : int LNext(List * plist, LData * pdata);
    - 순차적인 참조를 위해 반복 호출(리스트에 포함된 데이터 수)
    - 참조된 데이터의 다음 데이터가 pdata가 가리키는 메모리에 저장
    - LFirst함수 이후에 호출하여 다음 데이터 참조
    - 참조 성공 시 TRUE(1), 실패 시 FALSE(0) 반환

  - 데이터 삭제 (바로 이전에 참조(반환)가 이루어진) : LData LRemove(List * plist);
    - LFrist 혹은 LNext 함수의 마지막 반환 데이터를 삭제(호출 : 특정 데이터 매칭시)
    - 삭제된 데이터는 반환
    - 반복 호출을 허용하지 않음
    - 리스트 원소의 위치 조정 및 개수 조정

  - 현재 저장되어 있는 데이터 수를 반환 : int LCount(List * plist);
    - 리스트에 저장되어 있는 데이터의 수를 반환

단순 연결 리스트의 ADT 설정 및 구현 - III

SUNG KYUN KWAN UNIVERSITY

- ADT 설정 (추가 : 정렬)
  - 데이터 정렬 : void SetSortRule(List * plist, int (*comp)(Ldata d1, Ldata d2));
    - 리스트에 정렬의 기준이 되는 함수를 등록하여 사용
    - 함수 포인터를 사용하여 함수를 지정
    - 반환형이 int이고, Ldata 인자를 두 개 전달받고, 함수의 주소값을 전달하는 역할(상황에 따라 프로그래머가 이를 새롭게 정의할 수 있음)

```
int WhoIsPrecede(int d1, int d2) {
        if (d1 < d2)
                return 0;
        else
                return 1;
}
```

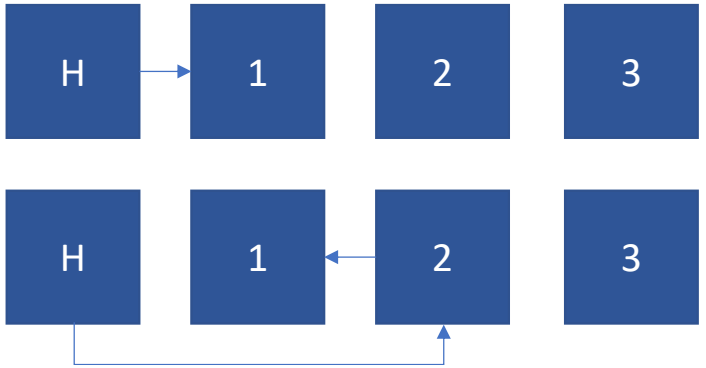
위와 같은 함수가 인자로 전달될 수 있음 :

0이면 d1이 head에 더 가까운 값 : 첫 번째 인자 data가 순서상 앞서서 head에 더 가까움
(즉, 첫 번째 인자를 두 번째 인자 앞에 삽입)_
1이면 d2가 head에 더 가까운 값 : 두 번째 인자가 정렬 순서상 앞서서 head 에 더 가까움
(즉, 첫 번째 인자를 두 번째 인자 뒤에 값을 삽입)

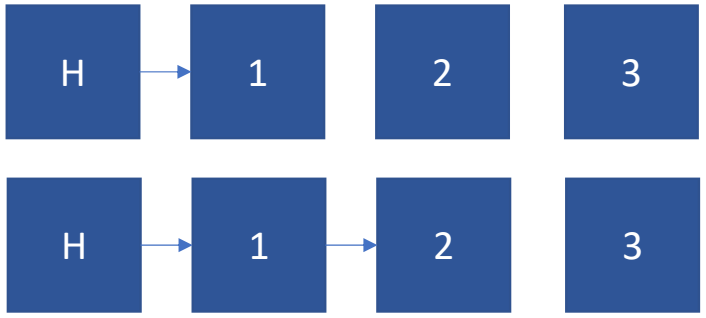
단순 연결 리스트의 ADT 설정 및 구현 - IV

SUNG KYUN KWAN UNIVERSITY

- 연결 리스트에서 새 노드 추가 위치
  - 새노드를 머리에 추가하는 경우
    - 장점 : Tail이 불필요함(항상 head)
    - 단점 : 저장된 순서를 유지하지 않음

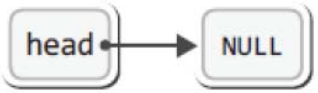


  - 새노드를 꼬리에 추가하는 경우
    - 장점 : 저장된 순서가 유지
    - 단점 : 포인터 변수 tail이 필요(항상 tail)

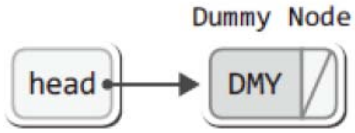


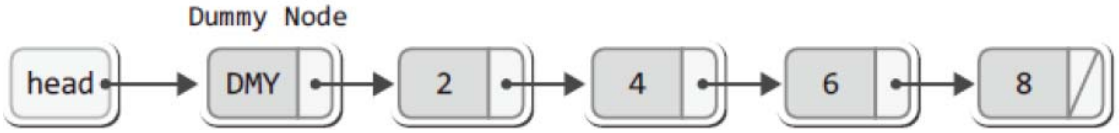
단순 연결 리스트의 ADT 설정 및 구현 - V

- 연결 리스트에서 새 노드 추가시 더미 노드
  - 더미 노드가 없는 경우 (일반적)
    - 특징 : 첫 번째 노드 와 그 이후 추가 및 삭제 방식이 다를 수 있음



  - 더미 노드가 있는 경우 (편의성)
    - 특징 : 노드의 추가 및 삭제 방식이 일정함.





단순 연결 리스트의 ADT 설정 및 구현 - VI

SUNG KYUN KWAN UNIVERSITY

- 단순 연결 리스트의 구조체 표현
  - 더미 노드를 가정하여
    - 더미 노드를 가리키기 위한 head : Node
    - 참조 및 삭제를 위한 cur : Node
    - 삭제를 돕는 before : Node (한 방향)
    - 저장된 데이터의 수 : numOfData
    - 정렬 기준 등록하기 위한 int (*comp)(Ldata d1, Ldata d2)

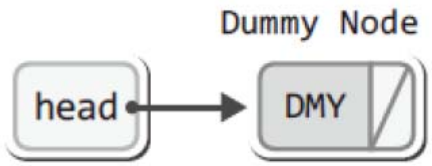
```
typedef struct _linkedList {
        Node * head;
        Node * cur;
        Node * before;
        int numOfData;
        int(*comp)(LData d1, LData d2);
}LinkedList;
```

단순 연결 리스트의 ADT 설정 및 구현 - 초기화

SUNG KYUN KWAN UNIVERSITY

- 단순 리스트 초기화 : void ListInit(List * plist);
  - Head 값 설정 : 더미노드를 사용하여
  - Head의 next 값 설정 : NULL
  - Comp 주소 값 설정 : NULL
  - numOfData 값 설정 : 0

```
void ListInit(List * plist) {
        plist->head = (Node *)malloc(sizeof(Node));
        plist->head->next = NULL;
        plist->comp = NULL;
        plist->numOfData = 0;
}
```



단순 연결 리스트의 ADT 설정 및 구현 - 삽입 I

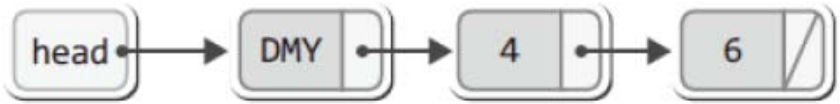
SUNG KYUN KWAN UNIVERSITY

- 리스트에 데이터 저장(삽입) : void Linsert(List * plist, Ldata data)
  - 정렬기준이 없는 경우 : Finsert
    - void Finsert(List * plist, Ldata data); → 머리에 노드를 추가하는 방식
  - 정렬기준이 있는 경우 : Sinsert – 다음에 구현
    - void Sinsert(List * plist, Ldata data);

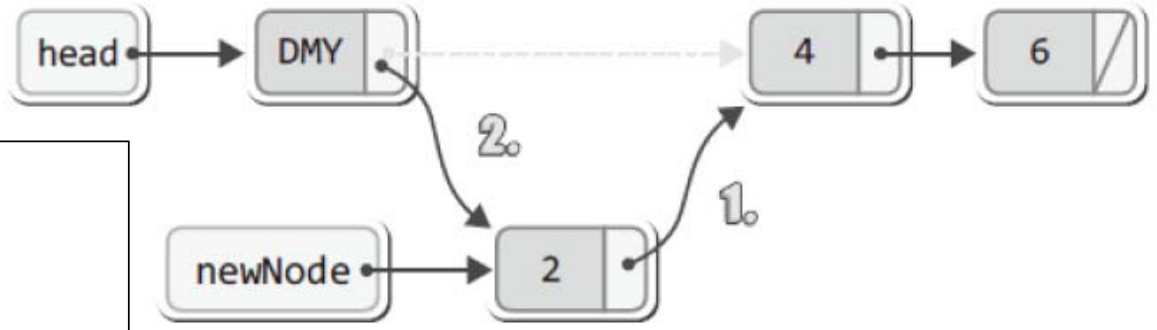
단순 연결 리스트의 ADT 설정 및 구현 - 삽입 II

SUNG KYUN KWAN UNIVERSITY

- 정렬 기준 없이 머리에 노드 추가 방법




```
Node * newNode = (Node *)malloc(sizeof(Node));
newNode->data = data;
```

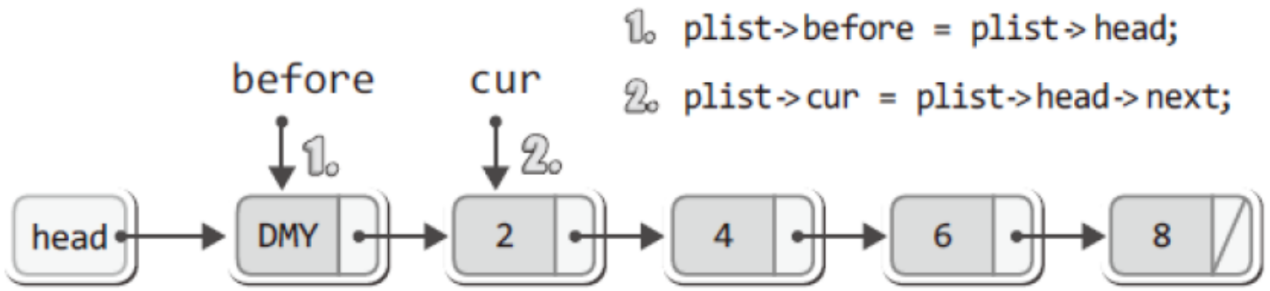



```
newNode->next = plist->head->next;
plist->head->next = newNode;

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

단순 연결 리스트의 ADT 설정 및 구현 - 참조(출력) I

- 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->head;
        plist->cur = plist->head->next;

        *pdata = plist->cur->data;

        return TRUE;
}
```

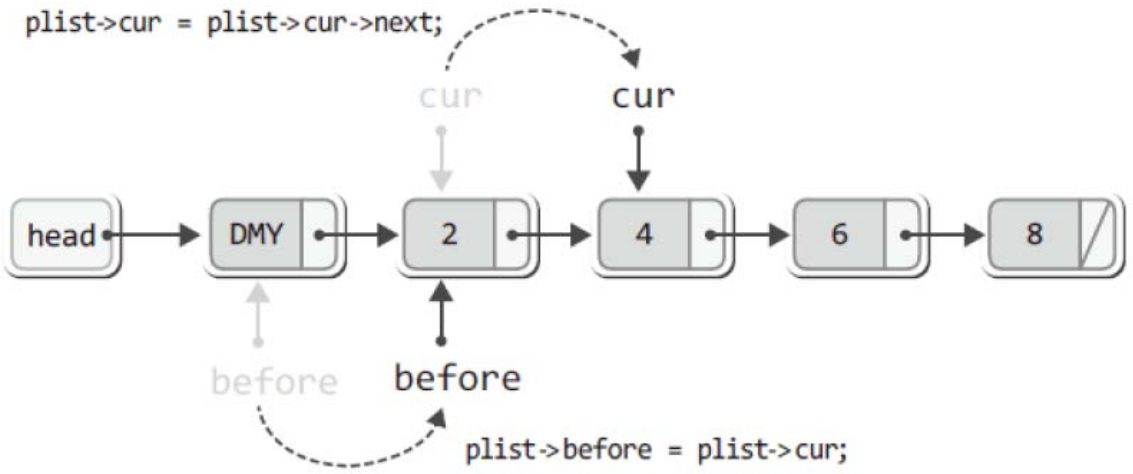


단순 연결 리스트의 ADT 설정 및 구현 - 참조(출력)II

SUNG KYUN KWAN UNIVERSITY

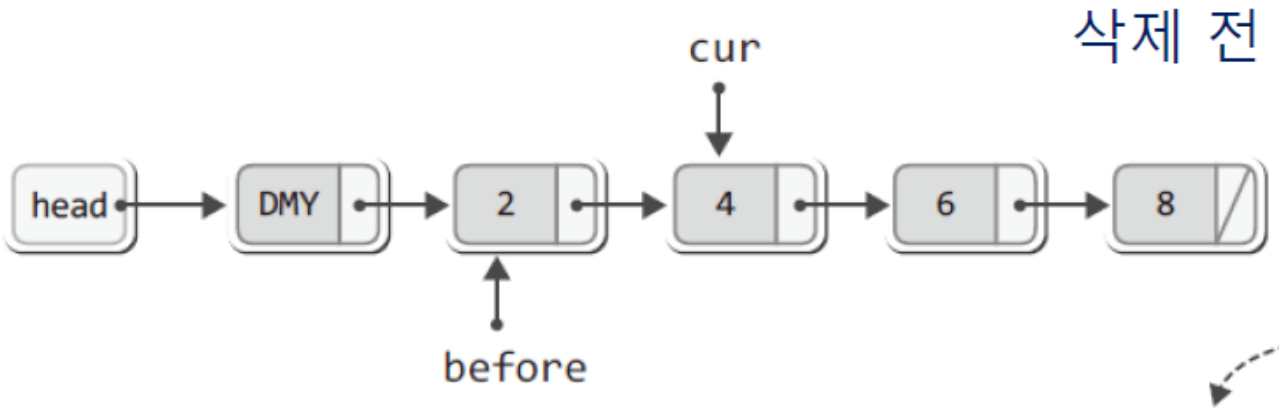
- 다음 데이터의 참조(반환) : int LNext(List * plist, LData * pdata);
  - Cur가리키던 것을 before가 가리킴
  - Cur는 다음 노드를 가리킴
  - 노드 데이터 전달

```
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;
}
```

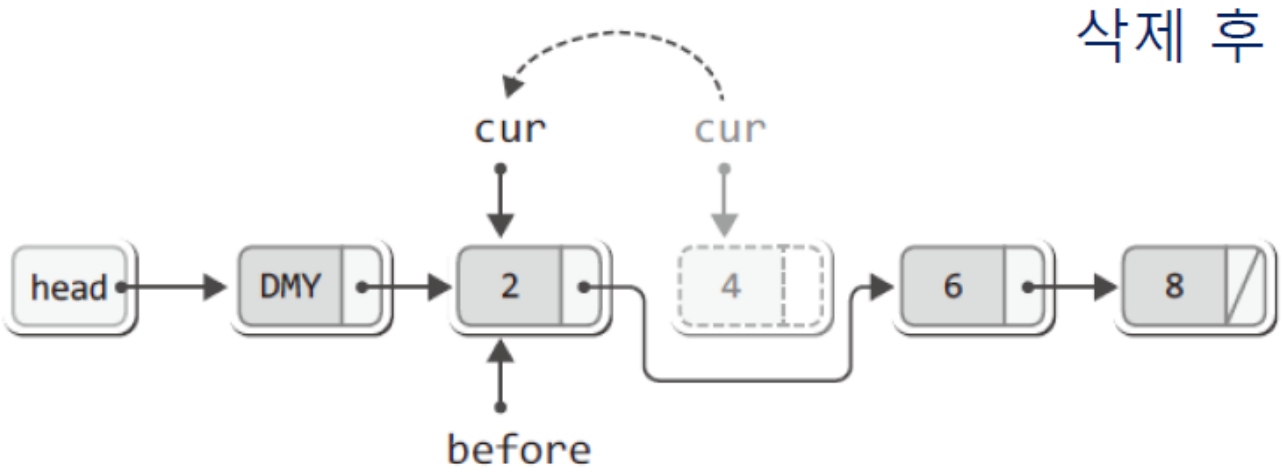


단순 연결 리스트의 ADT 설정 및 구현 - 삭제 I

- 삭제 과정
  - Lnext 혹은 Lfirst 를 통해 특정한 값이 발견되면 이 값을 삭제

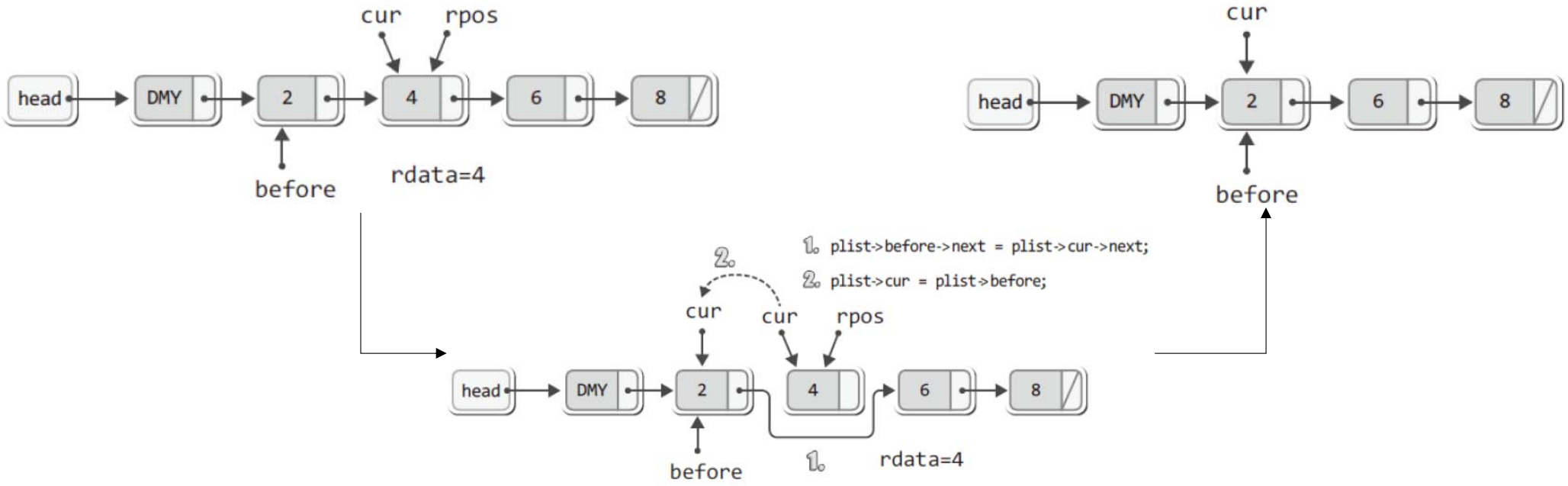


Before 는 Lnext와 Lfirst시 재설정되므로 재조정이 불필요.
Cur는 삭제후 재조정



단순 연결 리스트의 ADT 설정 및 구현 - 삭제 II

- 삭제 과정
  - 삭제할 노드를 가리키는 주소 설정 : Node * rpos
  - 삭제할 노드의 데이터 설정 : Ldata rdata
  - 삭제할 노드의 링크를 제외 (이전 노드를 다음노드로 가리킴, cur->before)
  - 노드 삭제 및 데이터 수 줄임



단순 연결 리스트의 ADT 설정 및 구현 - 삭제 III

SUNG KYUN KWAN UNIVERSITY

- 노드 삭제 구현

```
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;

}
```

단순 연결 리스트의 ADT 설정 및 구현 - 정렬삽입 I

SUNG KYUN KWAN UNIVERSITY

- 정렬기준 설정 방법
  - 정렬기준이 되는 함수를 등록하는 SetSortRule 함수 존재
  - SetSortRule 함수를 통해 전달된 함수정보 저장을 위한 연결리스트 멤버 comp
  - Comp에 등록된 정렬기준을 근거로 데이터를 저장하는 Sinsert 함수 수행

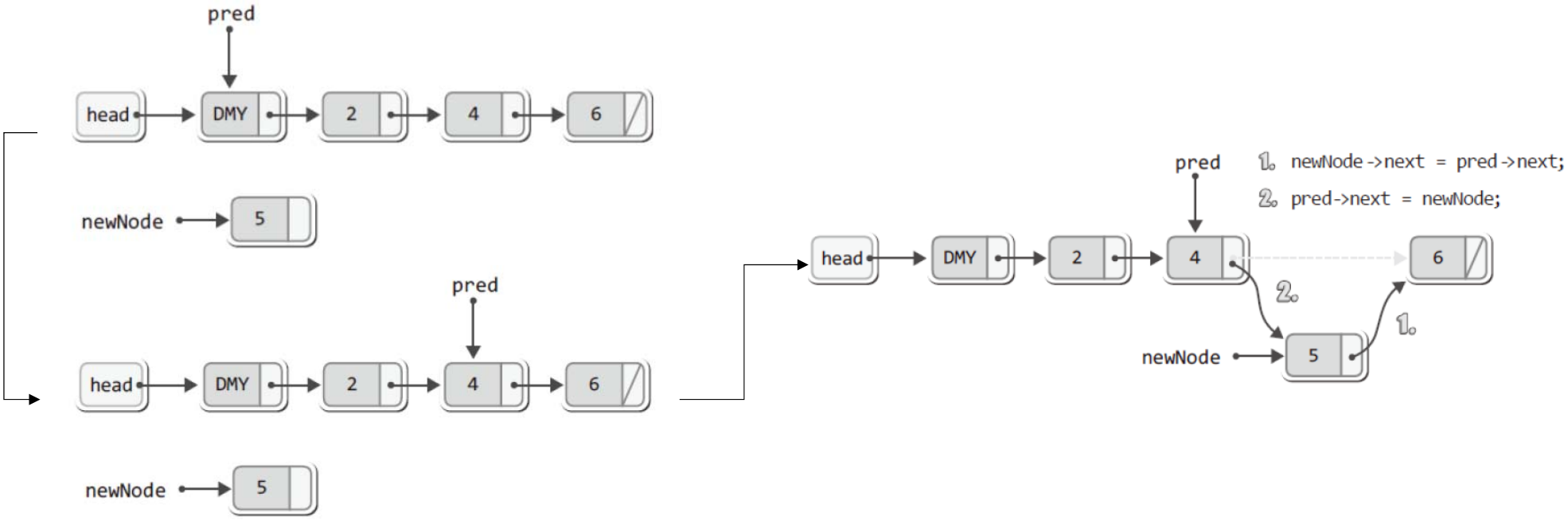
  - 즉, SetSortRule 함수가 호출되면 정렬의 기준이 리스트의 멤버 comp에 등록되면, SInsert함수 내에서는 comp에 등록된 정렬 기준을 근거로 데이터를 정렬함.

  - 정렬을 위해 Sinsert함수 만 정의하면 됨

단순 연결 리스트의 ADT 설정 및 구현 - 정렬삽입 II

SUNG KYUN KWAN UNIVERSITY

- 정렬을 위한 Insert함수 : void Sinsert(List * plist, Ldata data);
  - 새 노드 생성
  - 새 노드 들어갈 자리 찾기
  - 새 노드 들어갈 자리로 삽입
    - 이전 노드가 새노드를 가리킴, 새 노드 이전 노드가 가리키는 곳을 가리킴



단순 연결 리스트의 ADT 설정 및 구현 - 정렬삽입 III

SUNG KYUN KWAN UNIVERSITY

- 정렬을 위한 Insert함수 : void Sinsert(List * plist, Ldata data);

```
void SInsert(List * plist, LData data) {
	// 새 노드 생성
	Node * newNode = (Node*)malloc(sizeof(Node));
	Node * pred = plist->head;
	newNode->data = data;

	//새 노드 들어갈 위치 찾기, 노드 마지막과 우선순위가 0인 경우 필요없음
	while (pred->next != NULL && plist->comp(data, pred->next->data) != 0){
		pred = pred->next;
	}

	// 새 노드 들어갈 위치로 노드 삽입
	newNode->next = pred->next;
	pred->next = newNode;
	(plist->numOfData)++;
}
```

단순 연결 리스트의 ADT 설정 및 구현 - 정렬삽입 IV

SUNG KYUN KWAN UNIVERSITY

- 정렬을 기준 설정을 위함 함수: int WhoIsPrecede(int d1, int d2);

```
int WhoIsPrecede(int d1, int d2) {
        if (d1 < d2)
                return 0;       //d1이 정렬 순서상 앞섬 : pred 옮길 이유가 없음
        else
                return 1;       //d2가 정렬 순서상 앞섬 : pred를 옮김
}
```