

자료구조 & 알고리즘

for(A;B;C)
D;

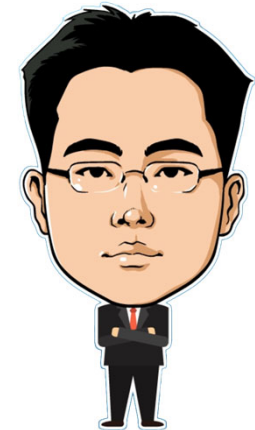


리스트
(List)

Seo, Doo-Ok

Clickseo.com

clickseo@gmail.com



목 차



- **선형 리스트**

- **연결 리스트**



선형 리스트



- 선형 리스트

- 선형 리스트 구현

- 연결 리스트



선형 리스트 (1/4)

● 리스트(List)

○ 목록, 대부분의 목록은 도표(Table) 형태로 표시

- 추상 자료형 리스트는 이러한 목록 또는 도표를 추상화한 것

이름 리스트	좋아하는 음식 리스트	오늘의 할 일 리스트
서두옥	김치찌개	자료구조 수업
홍길동	크림 스파게티	보고서 작성
이순신	불고기 피자	드라마 시청
이도	잡채	청소 하기
...

선형 리스트 (2/4)

● 선형 리스트(Linear List)

○ 순서 리스트(Ordered List)

- 리스트에서 나열한 원소들 간에 순서를 가지고 있는 리스트
- 원소들 간의 논리적인 순서와 물리적인 순서가 같은 구조(순차 자료구조)

이름 리스트		좋아하는 음식 리스트		오늘의 할 일 리스트	
1	서두옥	1	김치찌개	1	자료구조 수업
2	홍길동	2	크림 스파게티	2	보고서 작성
3	이순신	3	불고기 피자	3	드라마 시청
4	이도	4	잡채	4	청소 하기

선형 리스트 (3/4)

- **선형 리스트:** 원소 삽입

- 선형 리스트에서 원소 삽입

원소 삽입 전

0	1	2	3	4	5	6
10	20	40	50	60	70	

원소 삽입 후

0	1	2	3	4	5	6
10	20	40	50	60	70	

0	1	2	3	4	5	6
10	20	30	40	50	60	70

원소 30 삽입

선형 리스트 (4/4)

- 선형 리스트: 원소 삭제

- 선형 리스트에서 원소 삭제

원소 삭제 전

0	1	2	3	4	5	6
10	20	30	40	50	60	70

원소 삭제 후

0	1	2	3	4	5	6
10	20		40	50	60	70

원소 30 삭제

0	1	2	3	4	5	6
10	20	40	50	60	70	

선형 리스트

선형 리스트 구현



선형 리스트 구현 (1/2)

- 1차원 배열의 순차 표현

- 1차원 배열은 인덱스를 하나만 사용하는 배열

과 목	국어	영어	수학	총점
점 수	70	80	90	240

```
int arr[4] = {70, 80, 90, 240};
```

	[0]	[1]	[2]	[3]
arr	70	80	90	240

[학생 성적의 선형 리스트의 논리 구조]

0x0012ff70	...
0x0012ff74	70
0x0012ff78	80
0x0012ff7b	90
	240
	...

[학생 성적의 선형 리스트의 물리 구조]

선형 리스트 구현 (2/2)

- 2차원 배열의 순차 표현

- 행과 열의 구조로 나타내는 배열

- 메모리에 저장될 때에는 1차원의 순서로 저장

과목 \ 학생	국어	영어	수학	총점
1	70	80	90	240
2	50	60	70	180
3	60	70	80	210

```
int score[3][4] = {  
    {70, 80, 90},  
    {50, 60, 70},  
    {60, 70, 80}  
};
```

연결 리스트



- 선형 리스트
- 연결 리스트
 - 단순 연결 리스트
 - 원형 연결 리스트
 - 이중 연결 리스트



연결 리스트 (1/5)

● 순차 선형 리스트의 문제점

○ 리스트의 순서 유지를 위해 원소들의 삽입과 삭제가 어렵다.

- 삽입 또는 삭제 연산 후에 연속적인 물리 주소를 유지하기 위해서 원소들을 이동시키는 추가적인 작업과 시간이 소요된다.
 - 원소들의 빈번한 이동 작업으로 인한 오버헤드가 발생
 - 원소의 개수가 많고 삽입과 삭제 연산이 많이 발생하는 경우 더 많이 발생한다.

○ 메모리 사용의 비효율성

- 최대한의 크기를 가진 배열을 처음부터 준비해 두어야 하기 때문에 기억 장소의 낭비를 초래할 수 있다.

연결 리스트 (2/5)

● 순차 선형 리스트의 문제점: 파이썬 내장 리스트

○ 파이썬 내장 리스트

- 파이썬 리스트는 배열로 구현되어 있다.

insert()	
append()	
pop()	
remove()	
index()	
clear()	
count()	
extend()	
copy()	
reverse()	
sort()	

insert(i, x)	◀ x를 리스트의 i번 원소로 삽입한다. (맨 앞자리는 0번)
append(x)	◀ 원소 x를 리스트의 맨 뒤에 추가한다.
pop(i)	◀ 리스트의 i번 원소를 삭제하면서 알려준다.
remove(x)	◀ 리스트에서 (처음으로 나타나는) x를 삭제한다.
index(x)	◀ 원소 x가 리스트의 몇 번 원소인지 알려준다.
clear()	◀ 리스트를 깨끗이 청소한다.
count(x)	◀ 리스트에서 원소 x가 몇 번 나타나는지 알려준다.
extend(a)	◀ 리스트에 나열할 수 있는 객체(예 리스트) a를 풀어서 추가한다.
copy()	◀ 리스트를 복사한다.
reverse()	◀ 리스트의 순서를 역으로 뒤집는다.
sort()	◀ 리스트의 원소들을 정렬한다.

[이미지 출처: "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 문병로, 한빛아카데미, 2022.]

연결 리스트 (3/5)

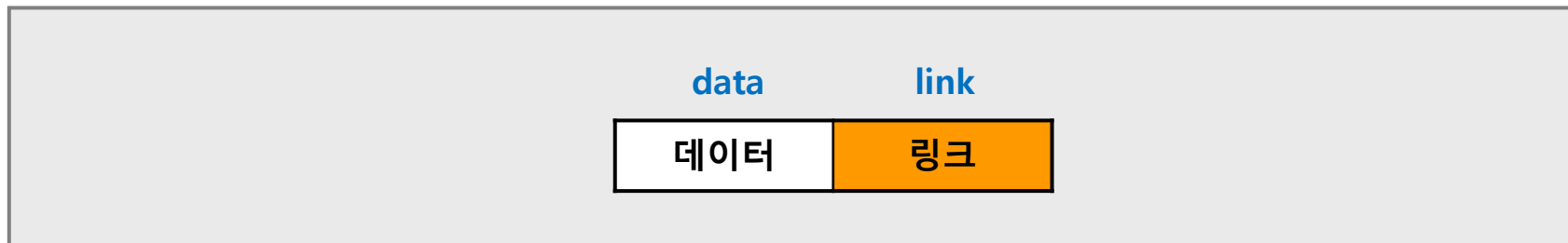
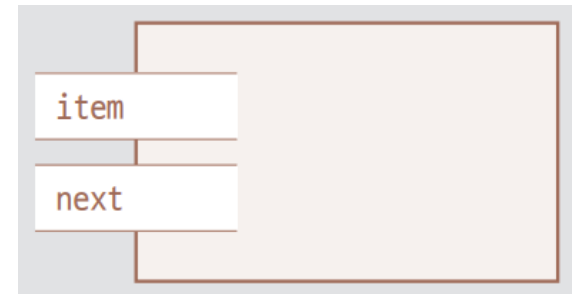
- **연결 리스트**(Linked List)

- 순차 자료구조에서의 연산 시간에 대한 문제와 저장 공간에 대한 문제를 개선한 자료 표현 방법

- 연결 자료구조(Linked Data Structure)
- 비 순차 자료구조(Nonsequential Data Structure)
- 데이터 아이템을 줄줄이 엮은(Link, Chain) 것

- **노드(Node):** <원소, 주소> 단위로 저장

- 데이터 필드(Data Field): 원소의 값을 저장
- 링크 필드(Link Field): 노드의 주소를 저장



연결 리스트 (4/5)

● 자기 참조 구조체

- 자신의 구조체 자료형을 가리키는 포인터 멤버를 가질 수 있다.

```
struct _score {  
    char        name[12];  
    int         kor, eng, math, tot;  
    float       ave;  
    struct _score* link;  
};  
typedef struct _score SCORE;
```

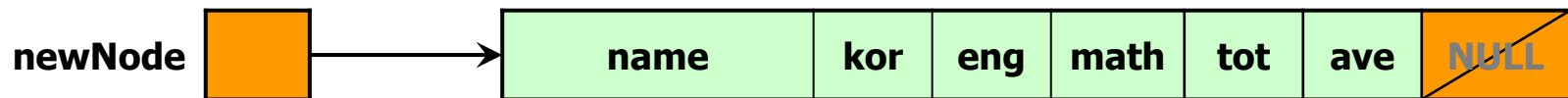
- link 멤버는 자신과 같은 구조의 구조체 주소를 저장하고 있다가 필요 시 저장된 주소의 구조체에 접근하는 것을 목표로 한다.

연결 리스트 (5/5)

- 자기 참조 구조체: 구조체 노드

- 구조체 노드의 생성

```
// struct score *head, *new_Node;  
SCORE*  head, *newNode;  
head = NULL;  
  
// SCORE 크기의 메모리 할당  
newNode = (SCORE*)malloc(sizeof(SCORE));  
if (newNode == NULL) {  
    printf("메모리 할당 실패!!! \n");  
    exit(100);  
}
```



연결 리스트

단순 연결 리스트



단순 연결 리스트 (1/9)

- **단순 연결 리스트**(Singly linked List)

- 선형 연결 리스트(linear linked list)

- 단순 연결 선형 리스트(singly linked linear list)

head



```
typedef struct _SNode {           // C
    int data;
    struct _SNode* link;
}SNode;
SNode* head;
```

```
class SNode : // Python
```

```
    self.__data
```

```
    self.__link
```

```
class SLinkedList:
```

```
    self.__head
```

```
class SNode {           // C++
```

```
private:
```

```
    int data;
```

```
    SNode* link;
```

```
    friend class SLinkedList; // SNode* head;
```

```
};
```

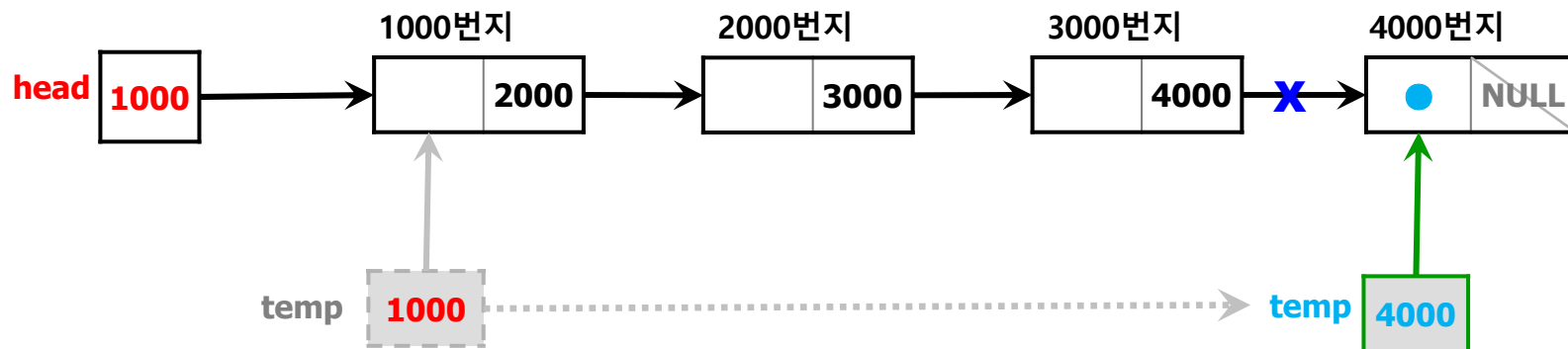
```
SLinkedList s = SLinkedList();
```

단순 연결 리스트 (2/9)

- 단순 연결 리스트: 탐색 알고리즘

- 리스트에서 조건을 만족하는 데이터를 가진 노드 탐색 알고리즘

```
searchSNode(head, data)
temp ← head;
while (temp != NULL) do
{
    if (temp.data = data) then
        return temp;
    temp ← temp.link;
}
if (temp = NULL) then
    return NULL;
end searchSNode()
```



단순 연결 리스트 (3/9)

- 단순 연결 리스트: 삽입 알고리즘

- 리스트의 첫 번째 노드 삽입 알고리즘

```
insertFirstNode(head, data)
```

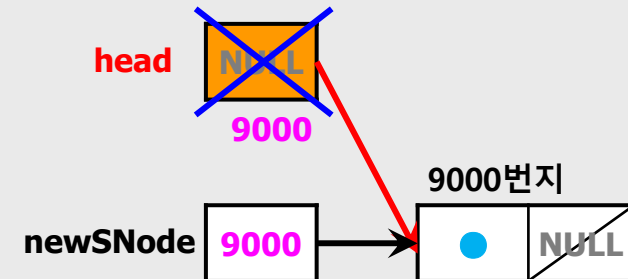
```
newSNode ← makeSNode(data);
```

```
newSNode.link = head;
```

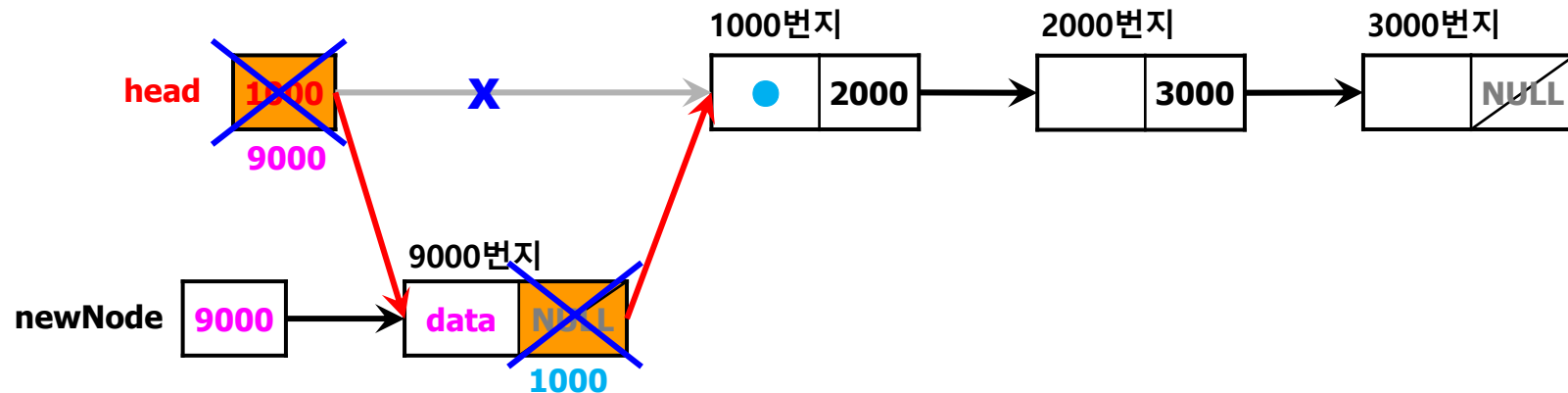
```
head ← newSNode;
```

```
end insertFirstNode()
```

// 빈 리스트일 경우...



// 빈 리스트가 아닐 경우...

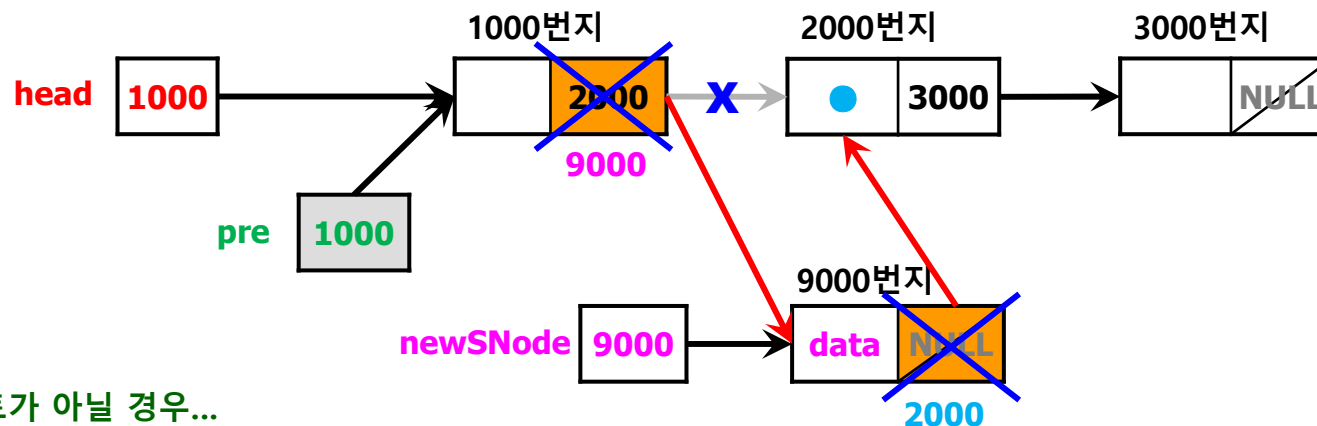


단순 연결 리스트 (4/9)

- 단순 연결 리스트: 삽입 알고리즘

- 리스트의 중간 노드 삽입 알고리즘

```
insertMiddleNode(head, pre, data)
  newNode ← makeSNode(data);
  if (head = NULL) then
    head ← newNode;
  else {
    newNode.link ← pre.link;
    pre.link ← newNode;
  }
end insertMiddleNode()
```

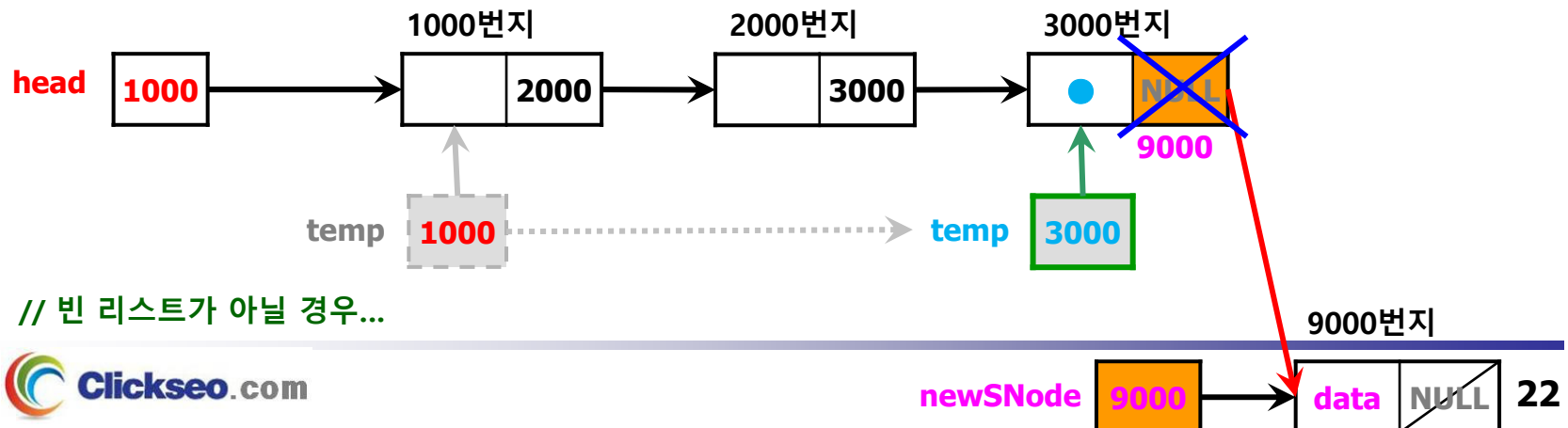


// 빈 리스트가 아닐 경우...

단순 연결 리스트 (5/9)

- 단순 연결 리스트: 삽입 알고리즘
 - 리스트의 마지막 노드 삽입 알고리즘

```
insertLastSNode(head, data)
  newSNode ← makeSNode(data);
  if (head = NULL) then
    head ← newSNode;
  else {
    // 맨 마지막 노드 탐색
    temp ← head;
    while (temp.link != NULL) do
      temp ← temp.link;
    temp.link ← newSNode;
  }
end insertLastSNode()
```



단순 연결 리스트 (6/9)

- 단순 연결 리스트: 삭제 알고리즘

- 리스트에서 조건을 만족하는 노드 삭제 알고리즘

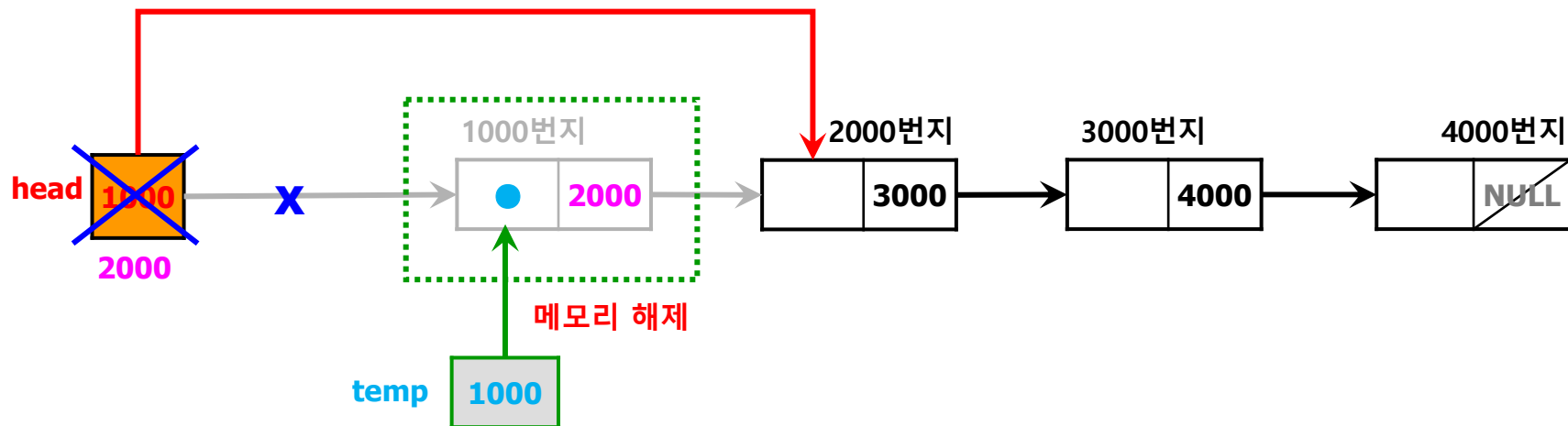
```
deleteSNode(head, data)
  if (head = NULL) then error;
  else {
    temp ← head;
    while (temp != NULL) {
      if (temp.data = data) then {
        if (temp = head) then deleteFirstNode();
        else if (temp = NULL) then deleteLastNode();
        else deleteMiddleNode();
      }
      pre ← temp;
      temp ← temp.link;
    }
  }
end deleteSNode()
```

단순 연결 리스트 (7/9)

- 단순 연결 리스트: 삭제 알고리즘

- 리스트의 첫 번째 노드를 삭제

- 삭제할 노드(old)의 다음 노드(old.link)를 head로 연결한다.



// 첫 번째 노드를 삭제

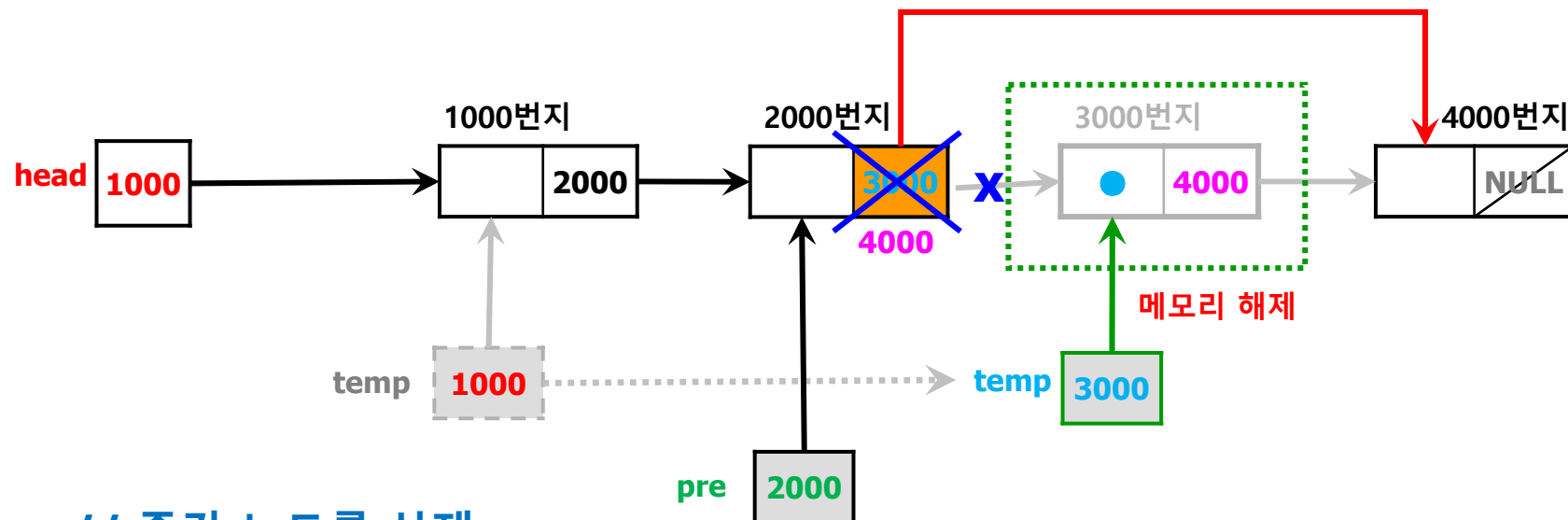
// deleteFirstSNode();

단순 연결 리스트 (8/9)

- 단순 연결 리스트: 삭제 알고리즘

- 리스트의 중간 노드를 삭제

- 삭제할 노드(old) 탐색 후 다음 노드(old.link)를 이전 노드(pre)의 다음 노드(pre.link)로 연결한다.



// 중간 노드를 삭제

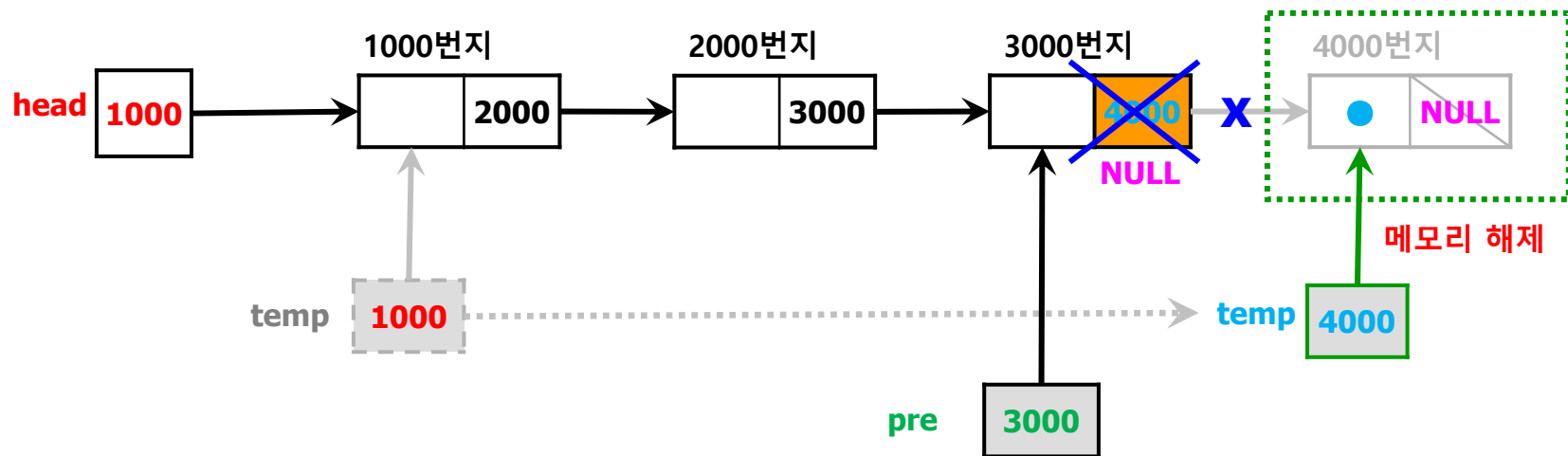
// deleteMiddleSNode();

단순 연결 리스트 (9/9)

- 단순 연결 리스트: 삭제 알고리즘

- 리스트의 마지막 노드를 삭제

- 삭제할 노드(old) 탐색 후 이전 노드(pre)의 링크 필드(pre.link)를 NULL로 만든다.



```
// 마지막 노드를 삭제  
// deleteLastSNode();
```

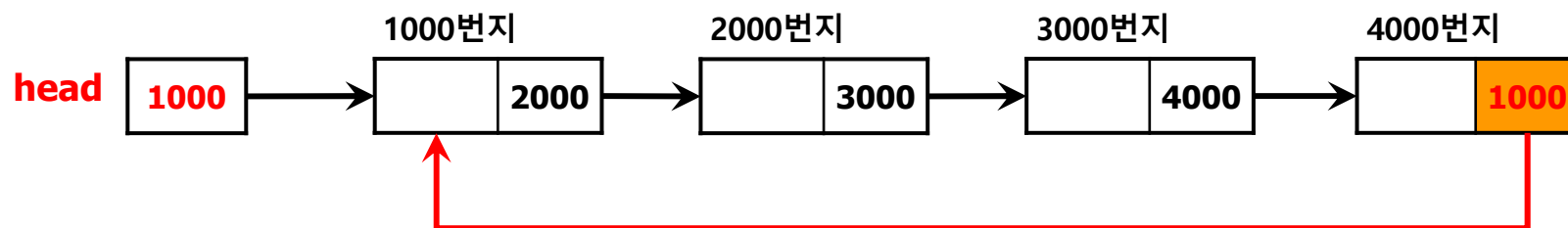
연결 리스트

원형 연결 리스트



원형 연결 리스트

- 원형 연결 리스트(Circular linked List)
 - 단순 연결 리스트에서 마지막 노드가 리스트의 첫 번째 노드를 가리키게 하여 구조를 원형으로 만든 연결 리스트



연결 리스트

이중 연결 리스트



이중 연결 리스트 (1/9)

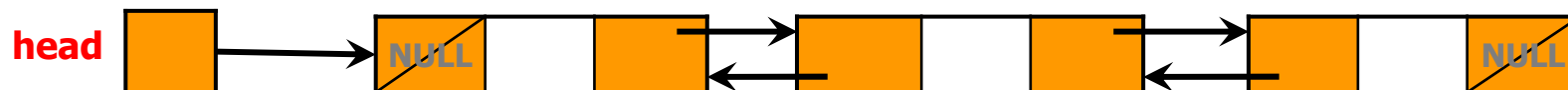
- 이중 연결 리스트(Doubly linked List)

- 원형 연결 리스트의 문제점

- 현재 노드의 바로 이전 노드를 접근하려면 전체 리스트를 한 바퀴 순회해야 한다.

```
typedef struct _DNode
{
    struct _Dnode*  Llink;
    int             data;
    struct _DNode*  Rlink;
}DNode;

DNode*  head;
```

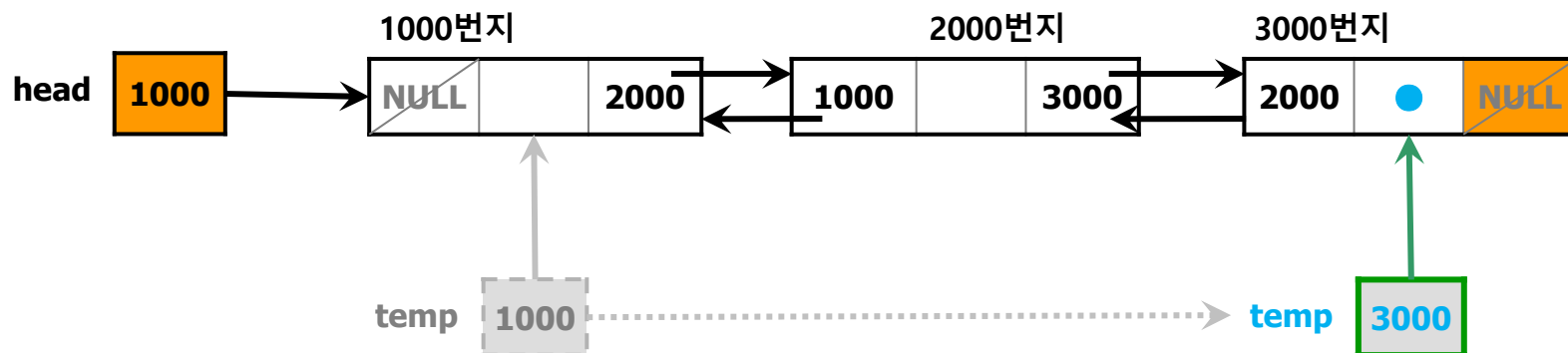


이중 연결 리스트 (2/9)

- 이중 연결 리스트: 탐색 알고리즘

- 리스트에서 조건을 만족하는 데이터를 가진 노드 탐색 알고리즘

```
searchDNode(head, data)
temp ← head;
while (temp != NULL) do
{
    if (temp.data = data) then
        return temp;
    temp ← temp.Rlink;
}
if (temp = NULL) then
    return NULL;
end searchDNode()
```



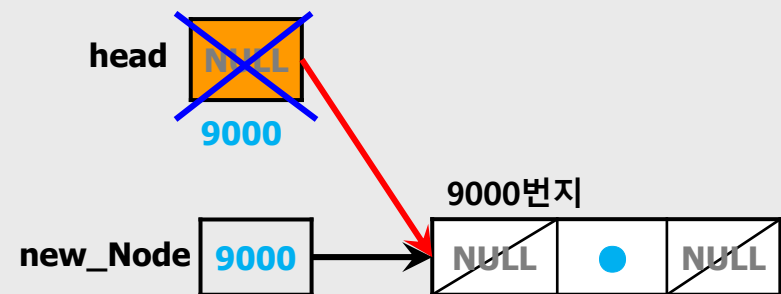
이중 연결 리스트 (3/9)

● 이중 연결 리스트: 삽입 알고리즘

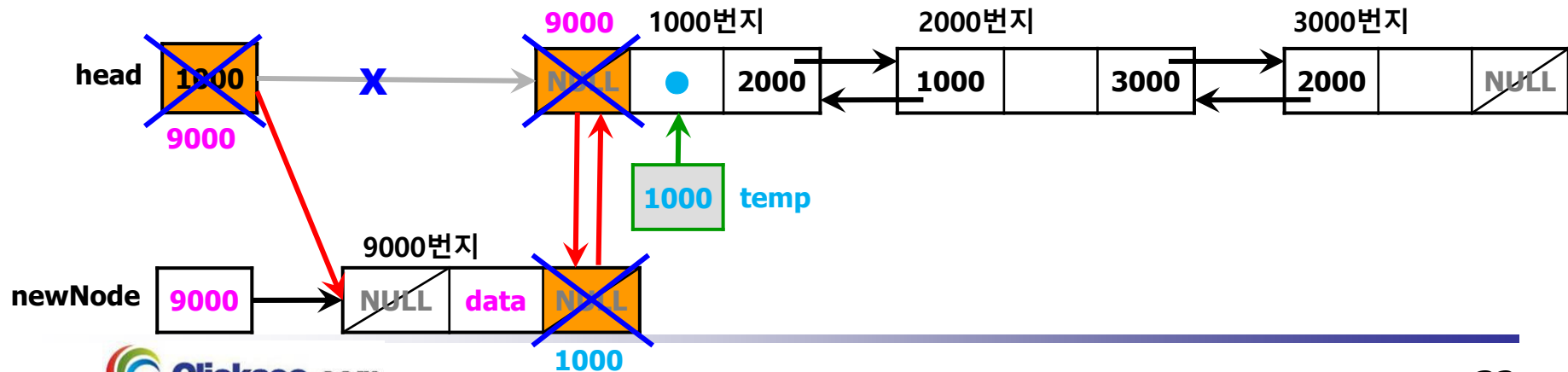
○ 리스트의 첫 번째 노드로 삽입

```
insertFirstDNode(head, data)
  newDNode ← makeDNode(data);
  if (head = NULL) then
    head = newDNode;
  else
  {
    head.Rlink = newDNode;
    newDNode.Rlink = head;
    head = newDNode;
  }
end insertFirstDNode()
```

// 빈 리스트일 경우...



// 첫 번째 노드로 삽입



이중 연결 리스트 (4/9)

- 이중 연결 리스트: 삽입 알고리즘

- 리스트의 중간 노드로 삽입

```
insertMiddleDNode(head, temp, data)
```

```
newDNode ← makeDNode(data);
```

```
if (head = NULL) then head = newDNode;
```

```
else
```

```
{
```

```
    newDNode.Llink = temp.Llink;
```

```
    newDNode.Rlink = temp;
```

```
    temp.Llink.Rlink = newDNode;
```

```
    temp.Llink = newDNode;
```

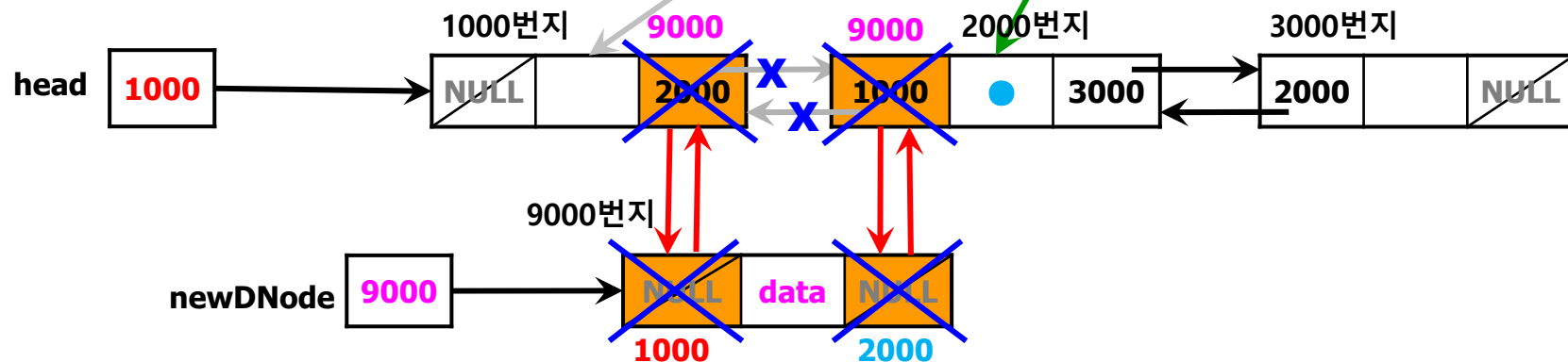
```
}
```

```
end insertMiddleDNode()
```

temp 1000

2000 temp

// 중간 노드로 삽입



이중 연결 리스트 (5/9)

● 이중 연결 리스트: 삽입 알고리즘

○ 리스트의 마지막 노드로 삽입

```
insertLastDNode(head, data)
```

```
  newDNode ← makeDNode(data);
```

```
  if (head = NULL) then
```

```
    head = newDNode;
```

```
  else
```

```
  {
```

```
    temp = head;
```

```
    while (temp.Rlink != NULL)
```

```
      temp = temp.Rlink;
```

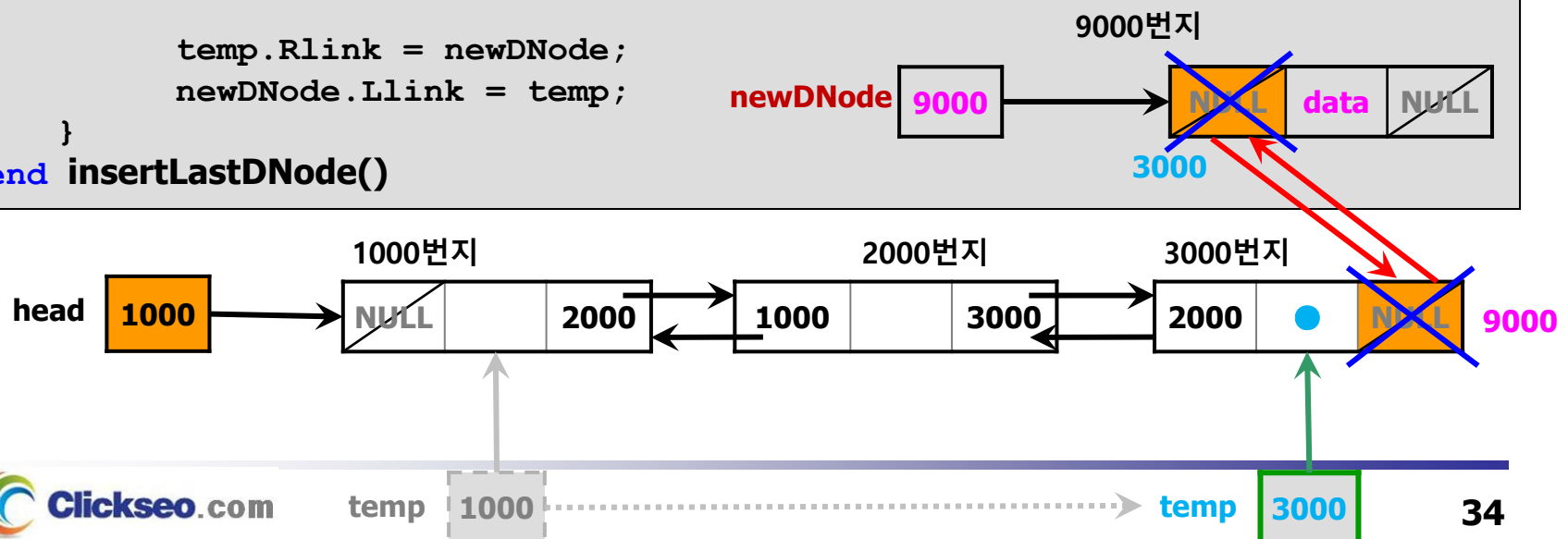
```
    temp.Rlink = newDNode;
```

```
    newDNode.Llink = temp;
```

```
  }
```

```
end insertLastDNode()
```

// 마지막 노드로 삽입



이중 연결 리스트 (6/9)

- 이중 연결 리스트: 삭제 알고리즘

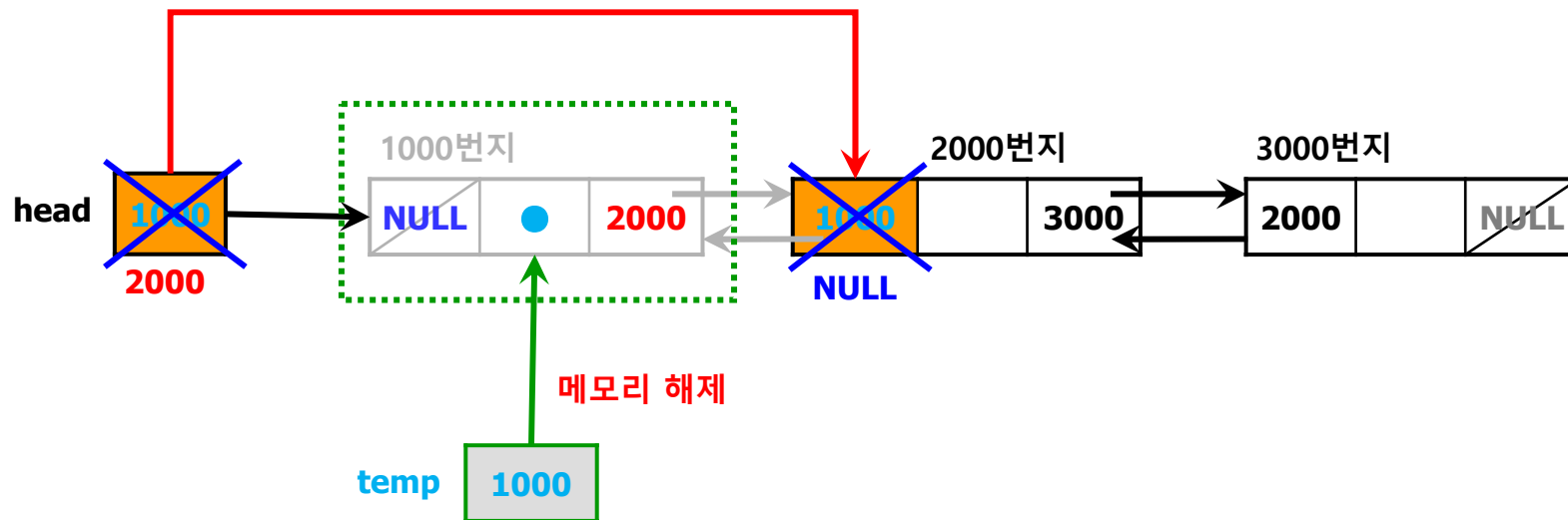
- 리스트에서 조건을 만족하는 노드 삭제 알고리즘

```
deleteDNode(head, data)
  if (head = NULL) then error;
  else {
    temp ← head;
    while (temp != NULL)
    {
      if (temp.data = data) then
        break;
      temp ← temp.link;
    }
    if (temp = head) then deleteFirstNode();
    else if (temp = NULL) then deleteLastNode();
    else deleteMiddleNode();
  }
end deleteDNode()
```

이중 연결 리스트 (7/9)

- 이중 연결 리스트: 삭제 알고리즘

- 리스트의 첫 번째 노드를 삭제



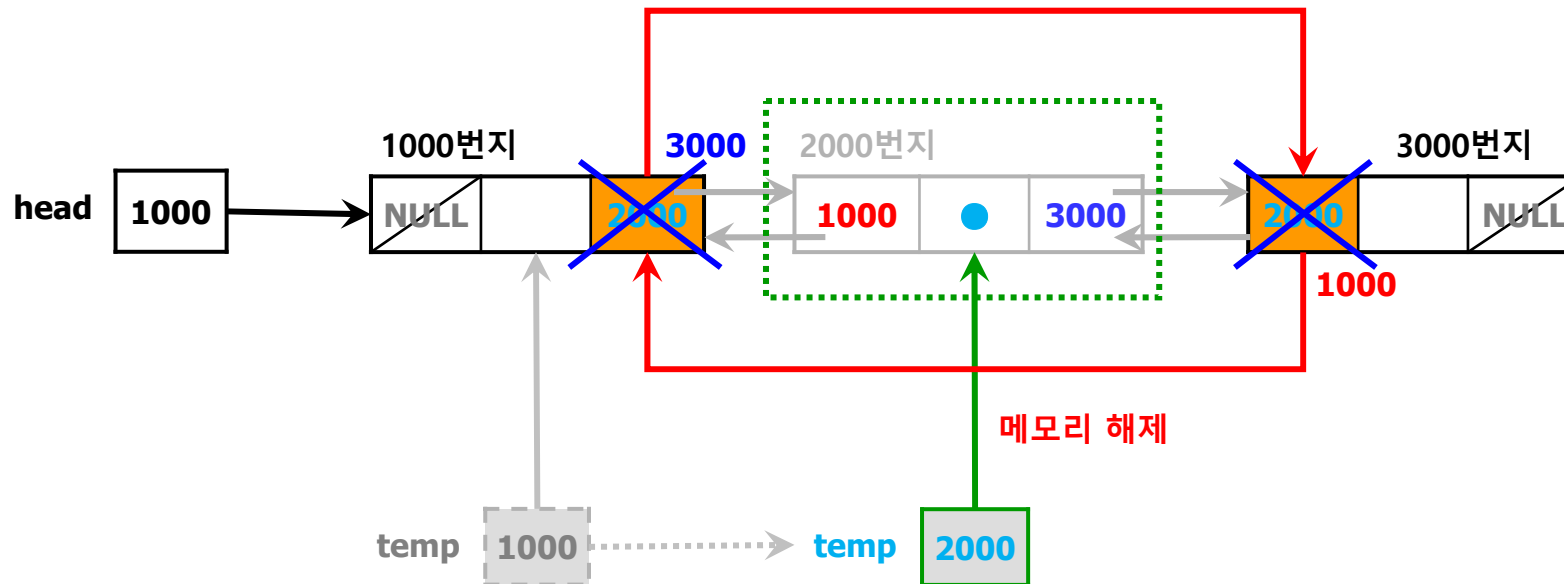
// 첫 번째 노드를 삭제

// deleteFirstDNode();

이중 연결 리스트 (8/9)

- 이중 연결 리스트: 삭제 알고리즘

- 리스트의 중간 노드를 삭제



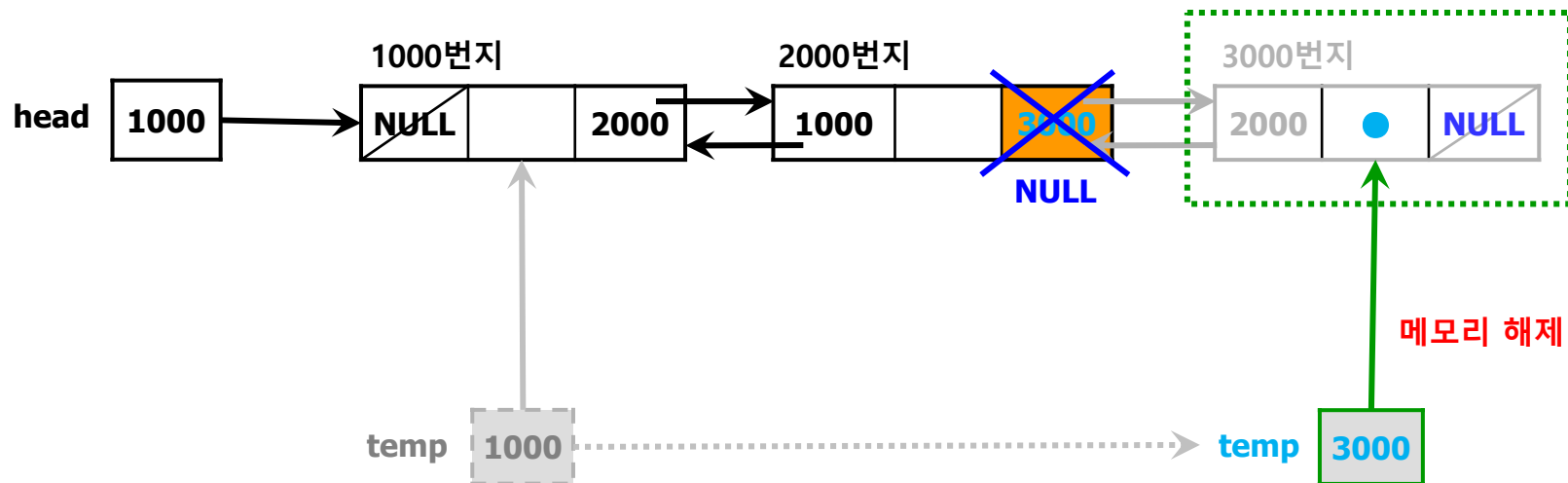
// 중간 노드를 삭제

// deleteMiddleNode();

이중 연결 리스트 (9/9)

- 이중 연결 리스트: 삭제 알고리즘

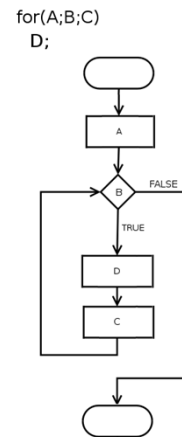
- 리스트의 마지막 노드를 삭제



```
// 마지막 노드를 삭제  
// deleteLastDNode();
```

참고문헌

- [1] Michael T. Goodrich 외 2인 지음, 김유성 외 2인 옮김, "C++로 구현하는 자료구조와 알고리즘", 한티에듀, 2020.
- [2] "프로그래밍 대회 공략을 위한 알고리즘과 자료 구조 입문", 와타노베 유타카 저, 윤인성 역, 인사이트, 2021.
- [3] "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 문병로, 한빛아카데미, 2022.
- [4] "이것이 취업을 위한 코딩 테스트다 with 파이썬", 나동빈, 한빛미디어, 2020.
- [5] 문병로, "IT CookBook, 쉽게 배우는 알고리즘: 관계 중심의 사고법"(개정판), 개정판, 한빛아카데미, 2018.
- [6] Richard E. Neapolitan, 도경구 역, "알고리즘 기초", 도서출판 홍릉, 2017.
- [7] 주우석, "IT CookBook, C · C++ 로 배우는 자료구조론", 한빛아카데미, 2019.
- [8] 이지영, "C 로 배우는 쉬운 자료구조", 한빛아카데미, 2022.



이 강의자료는 저작권법에 따라 보호받는 저작물이므로 무단 전제와 무단 복제를 금지하며,
내용의 전부 또는 일부를 이용하려면 반드시 저작권자의 서면 동의를 받아야 합니다.

Copyright © Clickseo.com. All rights reserved.

