Clickseo.com

C Programming

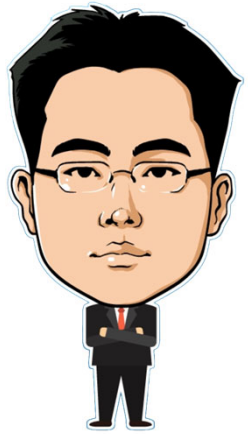
리스트

(List)

Seo, Doo-Ok

Clickseo.com

clickseo@gmail.com



목 차

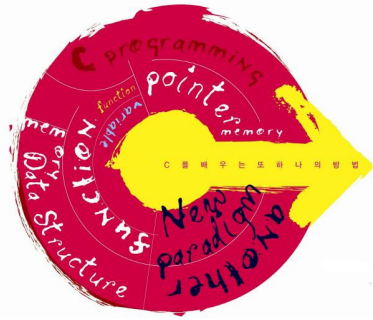
백문이불여일타(百聞而不如一打)

- 선형 리스트
- 연결 리스트
- 연결 리스트 구현

선형 리스트

백문이불여일타(百聞而不如一打)

- **선형 리스트**
  - 선형 리스트 구현
- 연결 리스트
- 연결 리스트 구현



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
홍길동	산채비빔밥	산책
이순신	활어회	수영
이도	잡채	글쓰기
강감찬	멧돼지 통구이	활쏘기
…	…	…

선형 리스트 (2/4)

● 선형 리스트(Linear List)

❍ 순서 리스트(Ordered List)

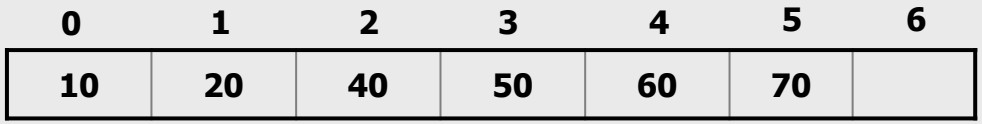
- 리스트에서 나열한 원소들 간에 순서를 가지고 있는 리스트
- 원소들 간의 논리적인 순서와 물리적인 순서가 같은 구조(순차 자료구조)

이름 리스트		좋아하는 음식 리스트		오늘의 할 일 리스트	
1	홍길동	1	산채비빔밥	1	산책
2	이순신	2	활어회	2	수영
3	이도	3	잡채	3	글쓰기
4	강감찬	4	멧돼지 통구이	4	활쏘기

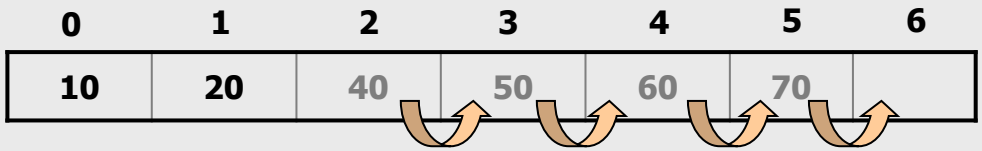
선형 리스트 (3/4)

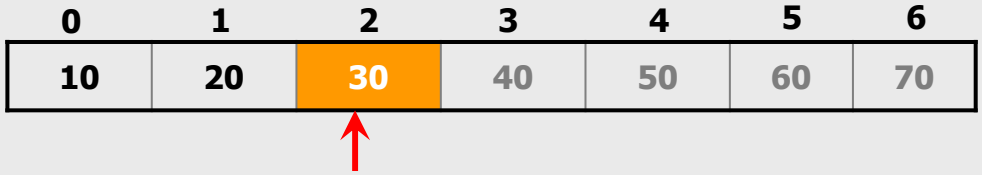
- **선형 리스트:** 원소 삽입
  - 선형 리스트에서 원소 삽입

원소 삽입 전



원소 삽입 후





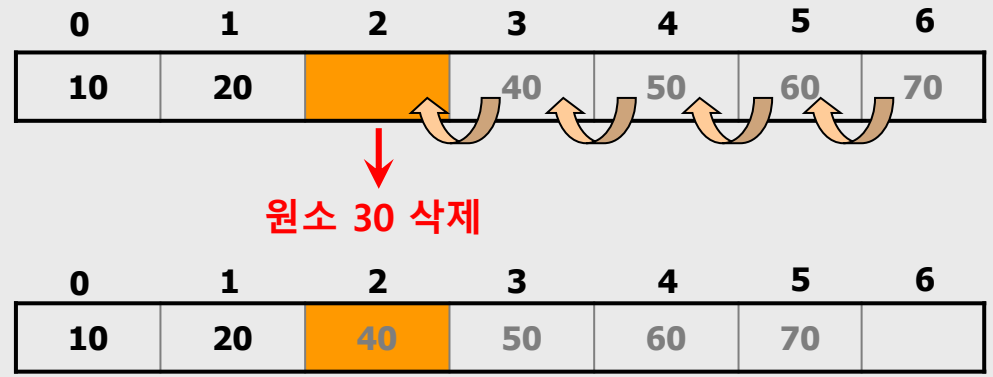
선형 리스트 (4/4)

- **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



선형 리스트

선형 리스트 구현

선형 리스트 구현 (1/2)

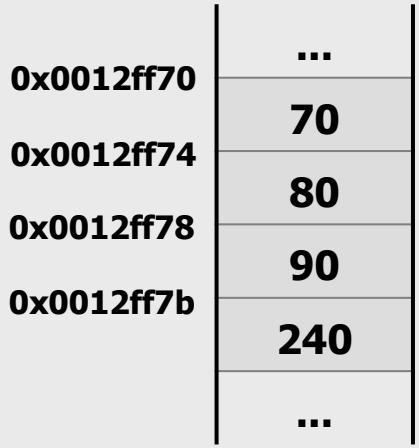
- **1차원 배열의 순차 표현**
  - 1차원 배열은 인덱스를 하나만 사용하는 배열

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]



[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

백문이불여일타(百聞而不如一打)

- **연결 리스트**
  - 단순 연결 리스트
  - 원형 연결 리스트
  - 이중 연결 리스트
- 연결 리스트 구현

연결 리스트 (1/4)

● 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.**

❍ 파이썬 내장 리스트

- **파이썬 리스트는 배열로 구현되어 있다.**

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

연결 리스트 (2/4)

- **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 (3/4)

- **자기 참조 구조체**
  - 자신의 구조체 자료형을 가리키는 포인터 멤버를 가질 수 있다.

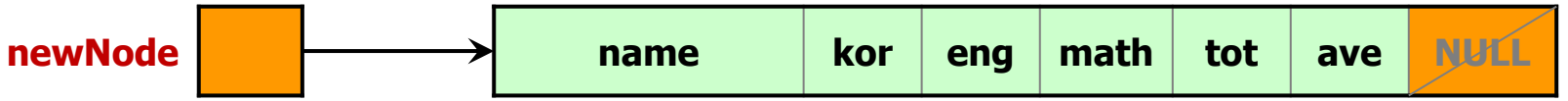
```
struct _score {
    char            name[12];
    int             kor, eng, math, tot;
    float           ave;
    struct _score   *link;
};
typedef struct _score SCORE;
```

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

연결 리스트 (4/4)

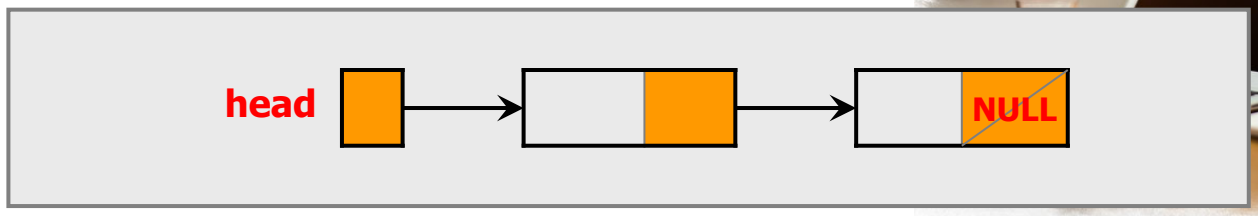
- **자기 참조 구조체:** 구조체 노드
  - 구조체 노드의 생성

```
// struct _score *head = NULL;
SCORE    *head = NULL;

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



연결 리스트

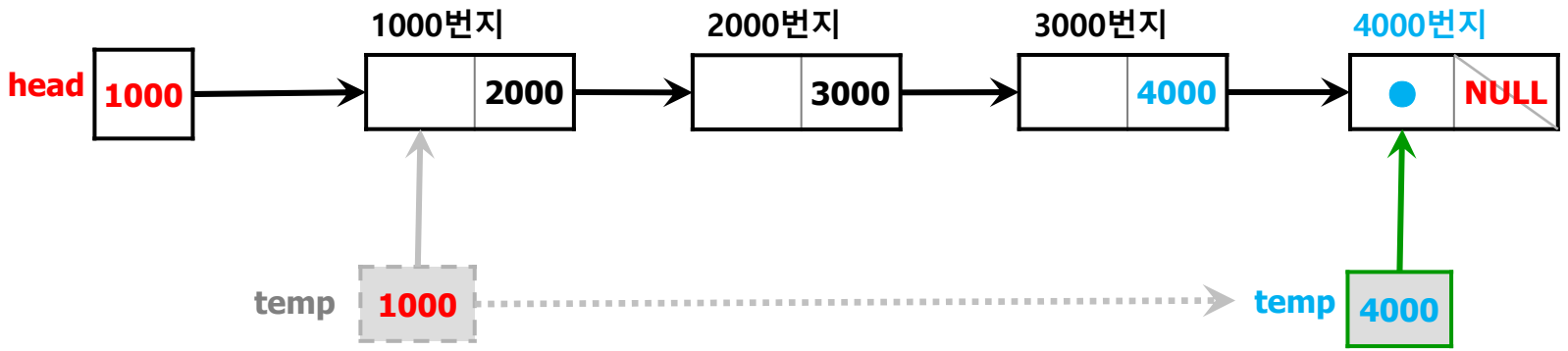
단순 연결 리스트: 알고리즘



단순 연결 리스트 (1/8)

- **단순 연결 리스트: 검색 알고리즘**
  - 리스트에서 조건을 만족하는 데이터를 가진 노드 탐색 알고리즘

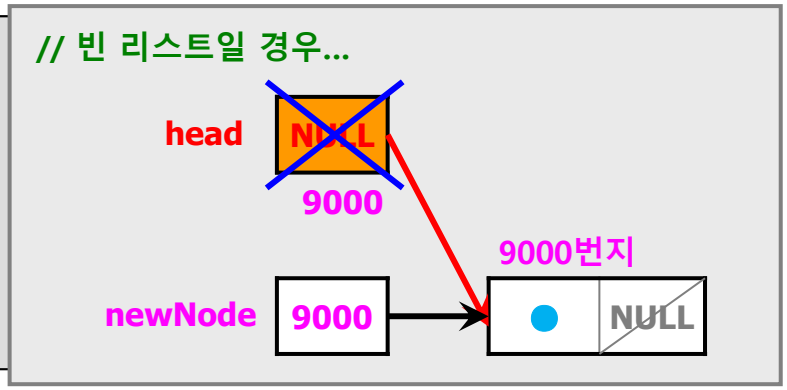
```
searchSNode(head, data)
    temp ← head;
    while (temp != NULL) do
    {
          if (temp.data = data) then
                return temp;
          temp ← temp.link;
    }
    if (temp = NULL)  then
          return NULL;
end searchSNode()
```



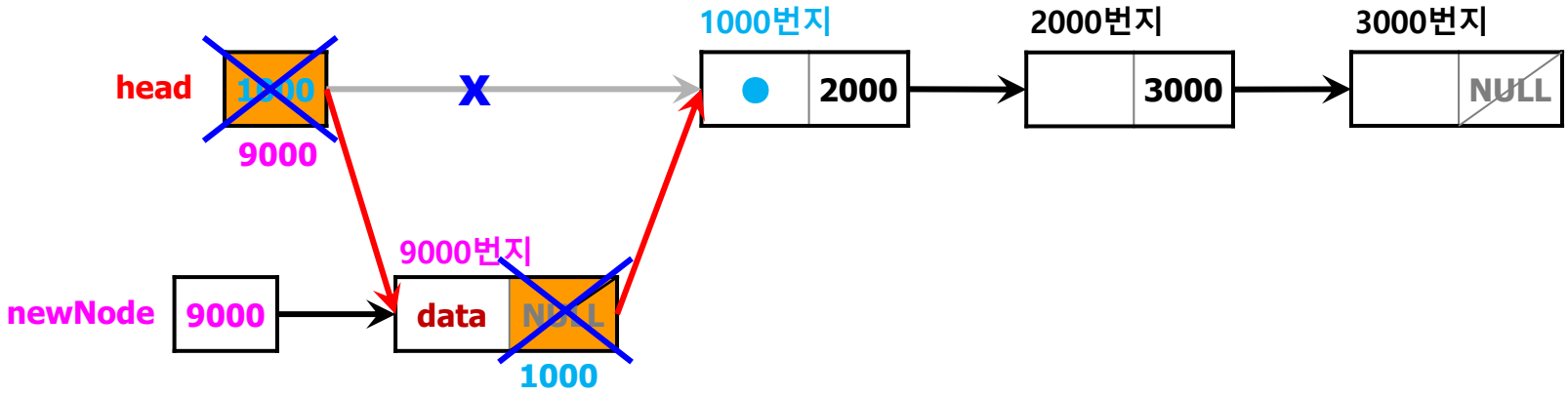
단순 연결 리스트 (2/8)

- **단순 연결 리스트: 삽입 알고리즘**
  - 리스트의 첫 번째 노드 삽입 알고리즘

```
insertFirstSNode(head, data)
    newNode ← makeNode(data);
    newNode.link = head;
    head ← newNode;
end insertFirstSNode()
```



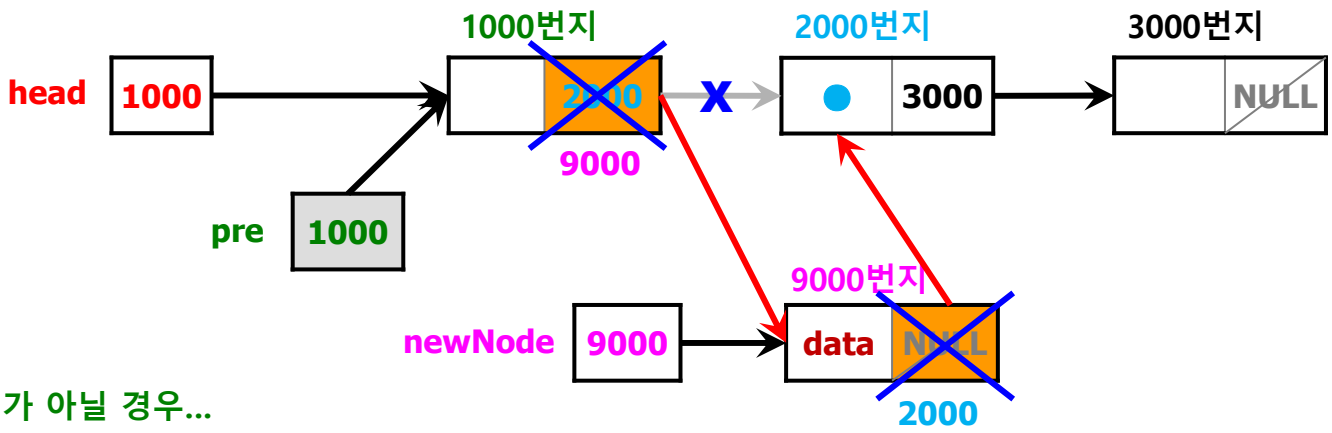
// 빈 리스트가 아닐 경우...



단순 연결 리스트 (3/8)

- **단순 연결 리스트:** 삽입 알고리즘
  - 리스트의 중간 노드 삽입 알고리즘

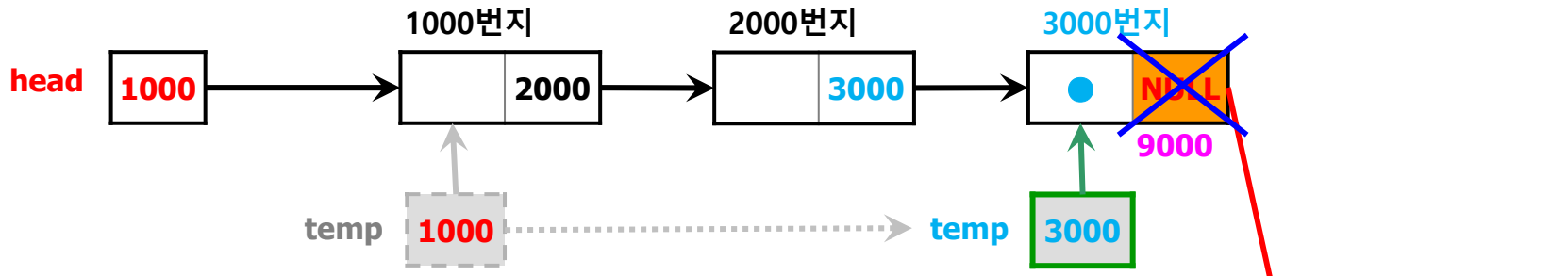
```
insertMiddleSNode(head, pre, data)
    newNode ← makeNode(data);
    if (head = NULL) then
          head ← newNode;
    else {
          newNode.link ← pre.link;
          pre.link ← newNode;
    }
end insertMiddleSNode()
```



단순 연결 리스트 (4/8)

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

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



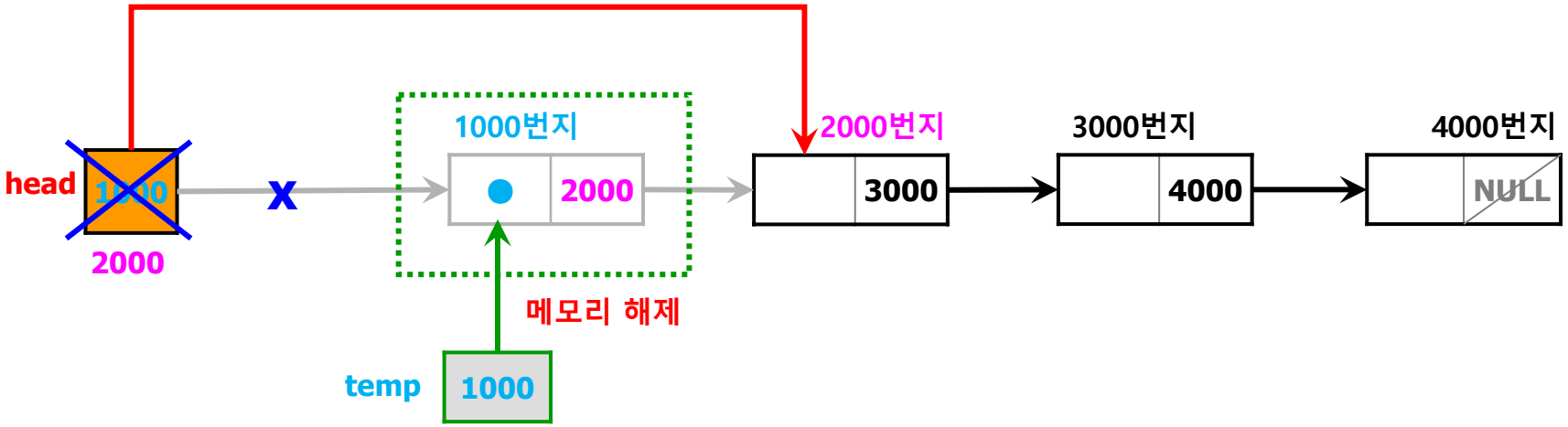
단순 연결 리스트 (5/8)

- **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 (6/8)

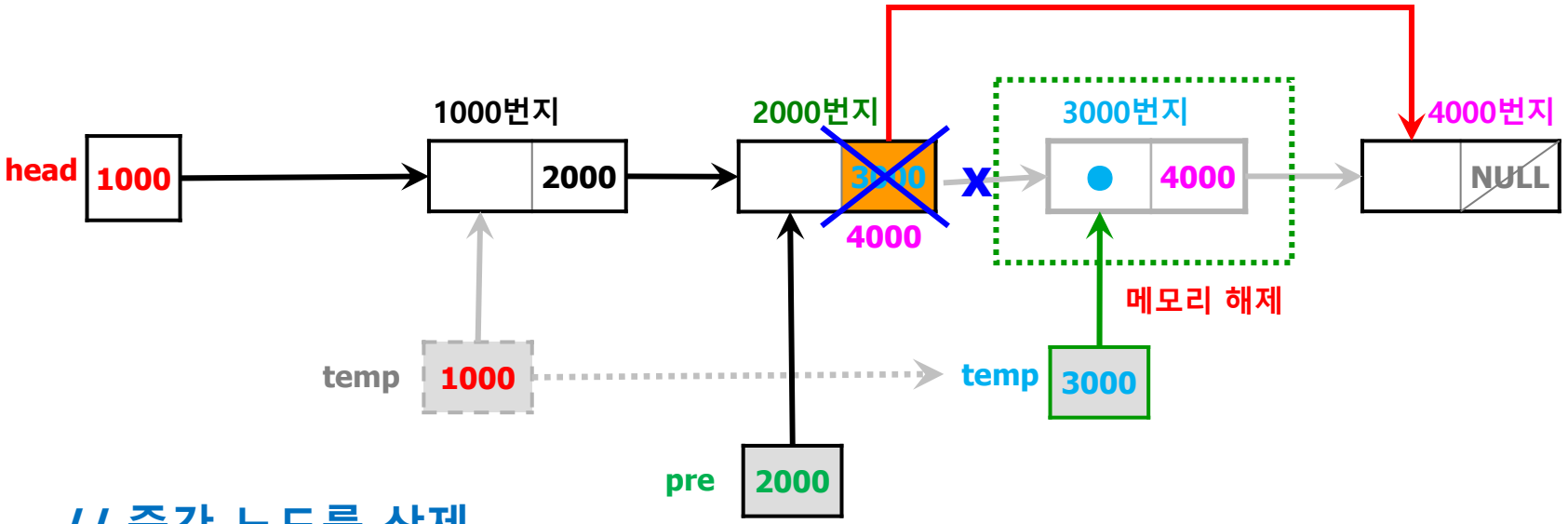
- **단순 연결 리스트:** 삭제 알고리즘
  - 리스트의 첫 번째 노드를 삭제
    - 삭제할 노드(temp)의 다음 노드(temp.link)를 head로 연결한다.




```
// 첫 번째 노드를 삭제
// deleteFirstSNode();
```

단순 연결 리스트 (7/8)

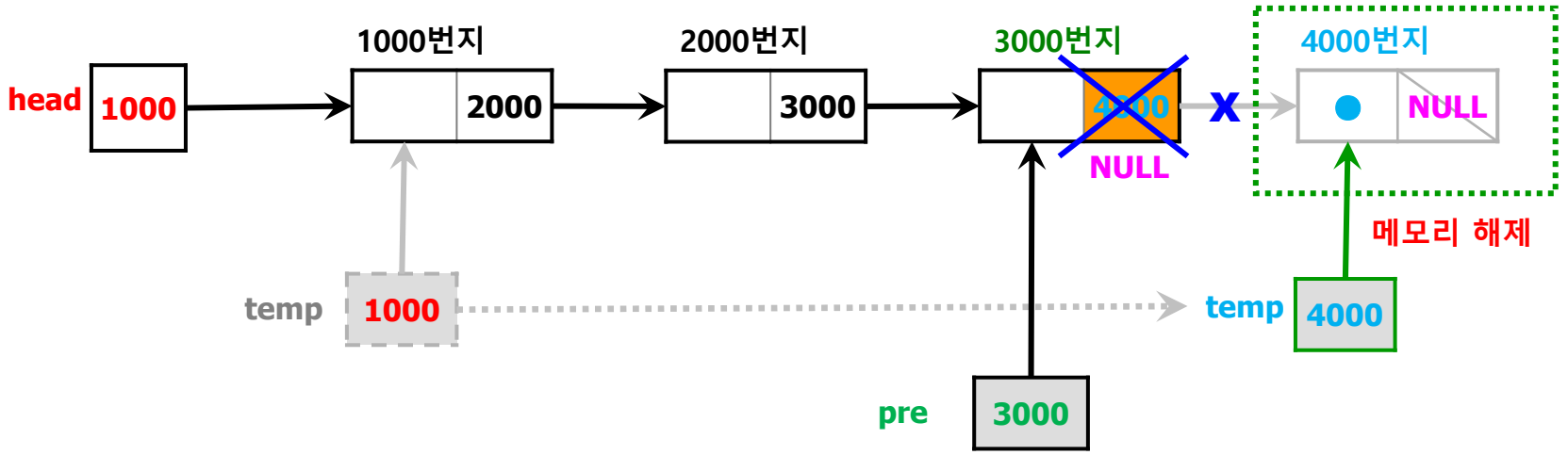
- **단순 연결 리스트:** 삭제 알고리즘
  - 리스트의 중간 노드를 삭제
    - 삭제할 노드(temp) 탐색 후 다음 노드(temp.link)를 이전 노드(pre)의 다음 노드(pre.link)로 연결한다.




```
// 중간 노드를 삭제
// deleteMiddleSNode();
```

단순 연결 리스트 (8/8)

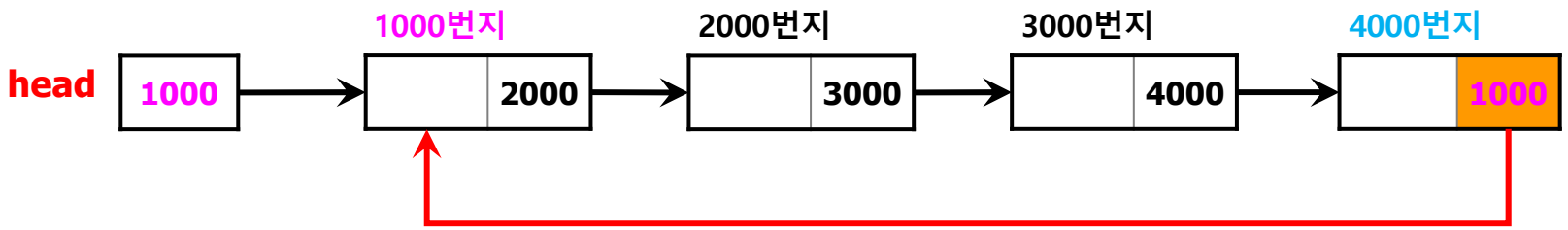
- **단순 연결 리스트:** 삭제 알고리즘
  - 리스트의 마지막 노드를 삭제
    - 삭제할 노드(old) 탐색 후 이전 노드(pre)의 링크 필드(pre.link)를 NULL로 만든다.




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

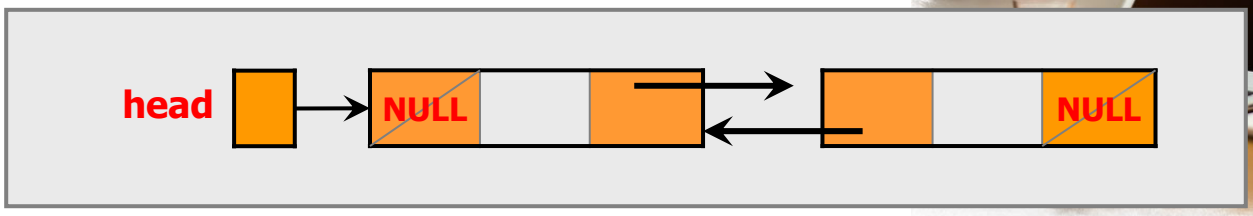
원형 연결 리스트

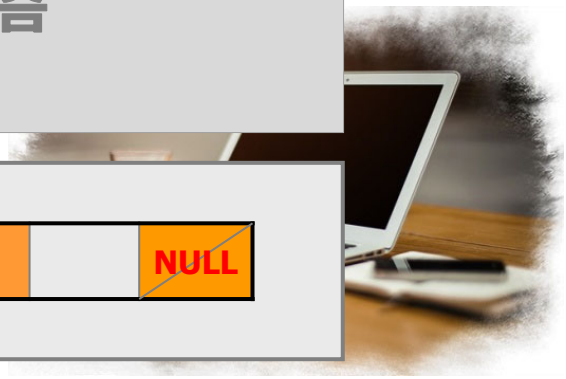
- **원형 연결 리스트**(Circular linked List)
  - 단순 연결 리스트에서 마지막 노드가 리스트의 첫 번째 노드를 가리키게 하여 구조를 원형으로 만든 연결 리스트



연결 리스트

이중 연결 리스트: 알고리즘

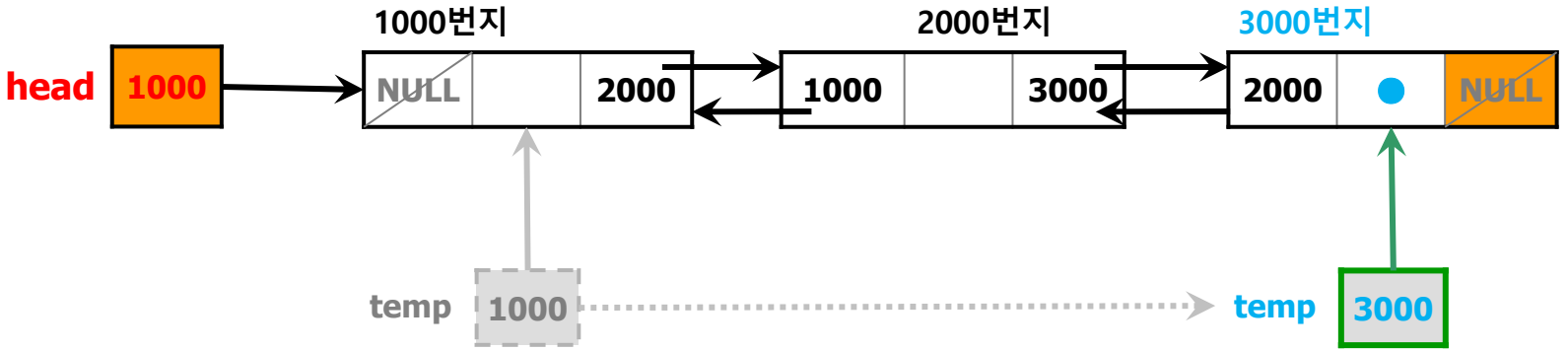




이중 연결 리스트 (1/8)

- **이중 연결 리스트:** 검색 알고리즘
  - 리스트에서 조건을 만족하는 데이터를 가진 노드 검색 알고리즘

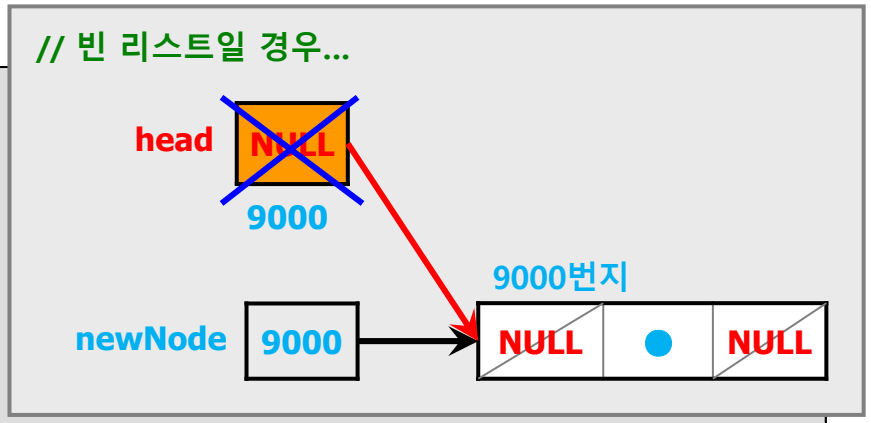
```
searchDNode(head, data)
    temp ← head;
    while (temp != NULL) do
    {
          if (temp.data = data) then
                return temp;
          temp ← temp.Rlink;
    }
    if (temp = NULL)   then
          return NULL;
end searchDNode()
```



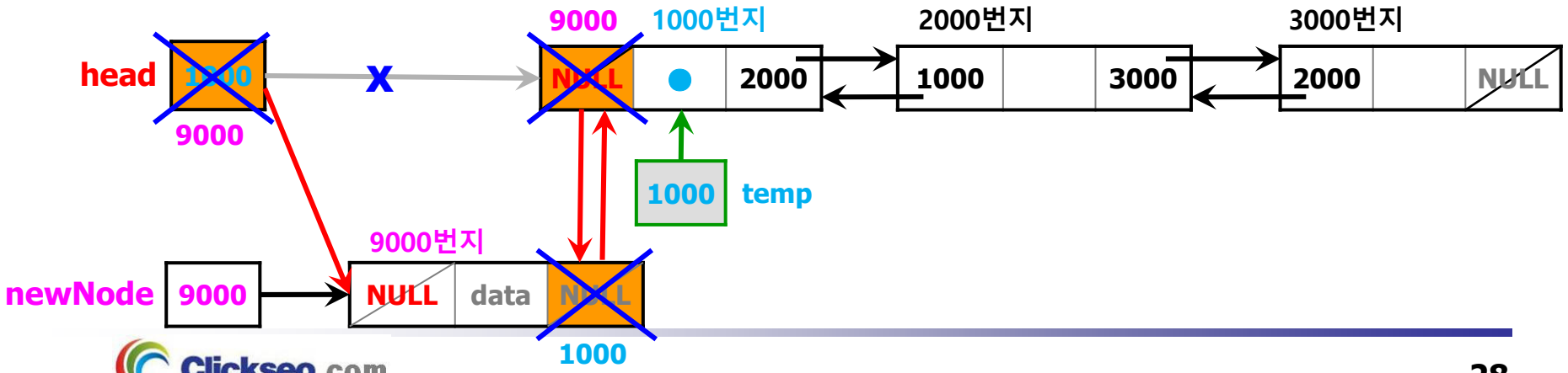
이중 연결 리스트 (2/8)

- **이중 연결 리스트:** 삽입 알고리즘
  - 리스트의 첫 번째 노드로 삽입

```
insertFirstDNode(head, data)
    newNode ← makeNode(data);
    if (head = NULL) then
          head = newNode;
    else
    {
          head.Llink = newNode;
          newNode.Rlink = head;
          head = newNode;
    }
end insertFirstDNode()
```



// 첫 번째 노드로 삽입

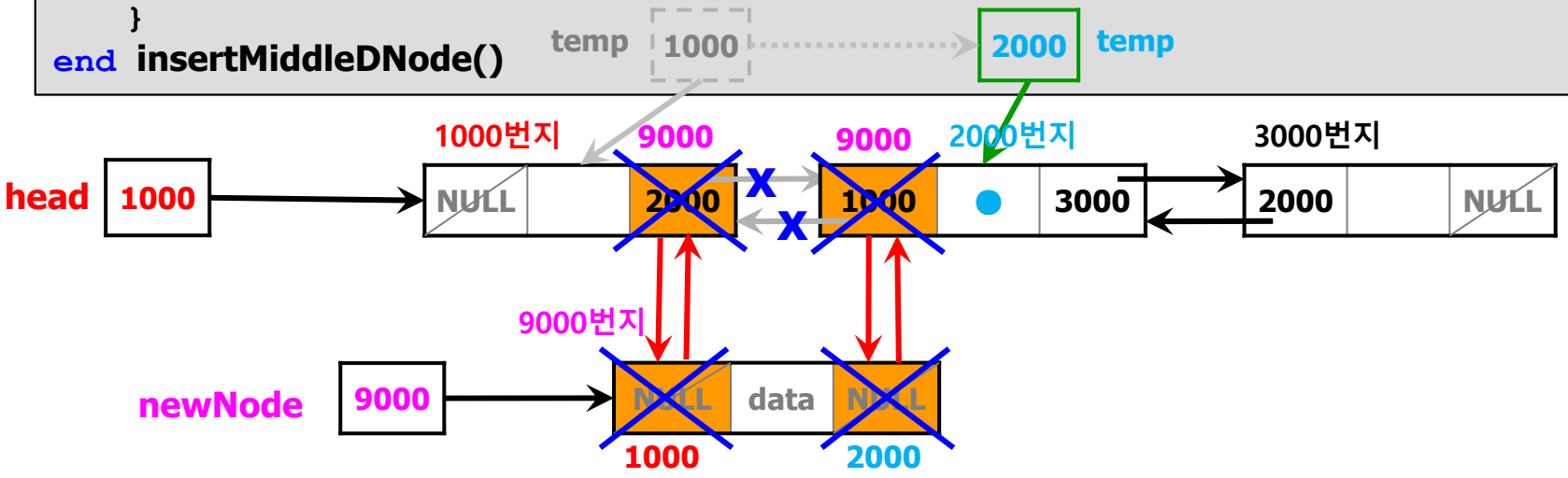


이중 연결 리스트 (3/8)

- **이중 연결 리스트:** 삽입 알고리즘
  - 리스트의 중간 노드로 삽입

```
insertMiddleDNode(head, temp, data)
    newNode ← makeNode(data);
    if (head = NULL) then    head = newNode;
    else
    {
          newNode.Llink = temp.Llink;
          newNode.Rlink = temp;
          temp.Llink.Rlink = newNode;
          temp.Llink = newNode;
    }
end insertMiddleDNode()
```

// 중간 노드로 삽입

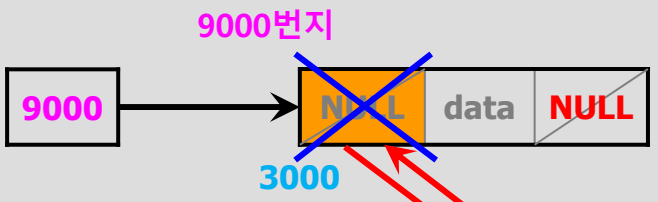


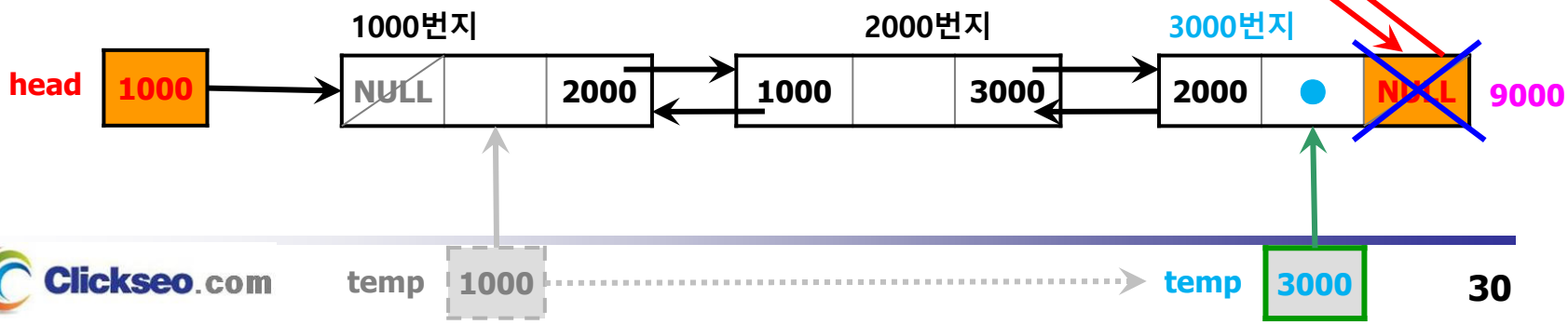
이중 연결 리스트 (4/8)

- **이중 연결 리스트: 삽입 알고리즘**
  - 리스트의 마지막 노드로 삽입

```
insertLastDNode(head, data)
    newNode ← makeNode(data);
    if (head = NULL) then
          head = newNode;
    else
    {
          temp = head;
          while (temp.Rlink != NULL)
                temp = temp.Rlink;

          temp.Rlink = newDNode;
          newNode.Llink = temp;
    }
end insertLastDNode()
```

// 마지막 노드로 삽입





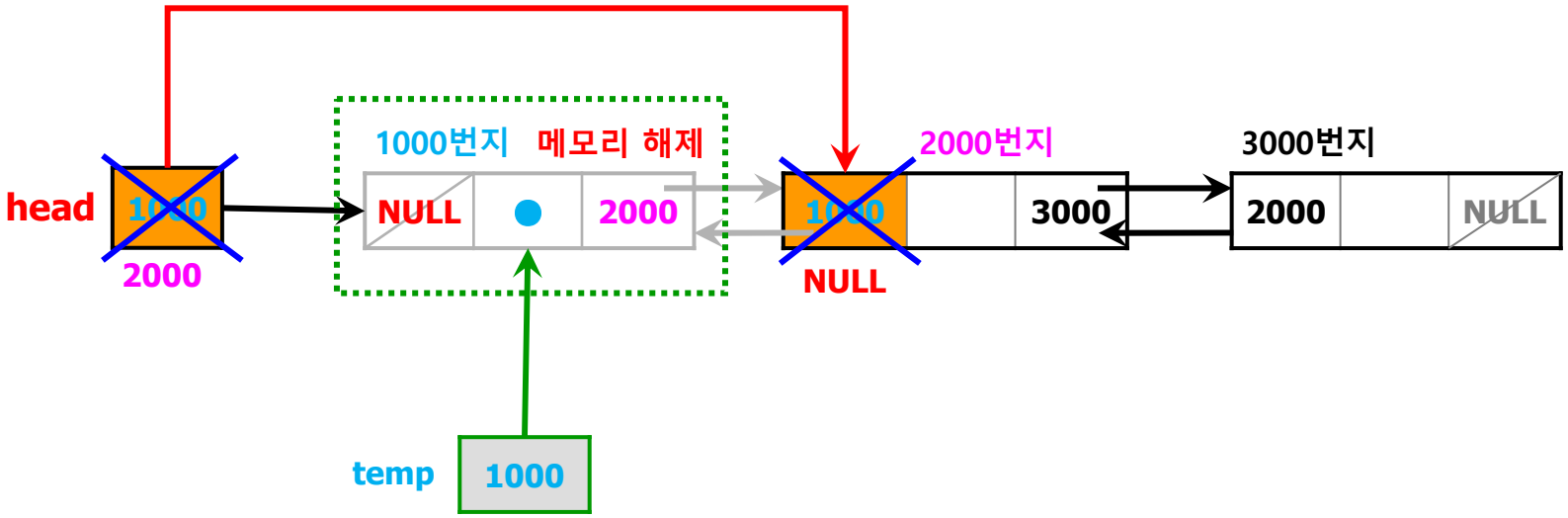
이중 연결 리스트 (5/8)

- **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 (6/8)

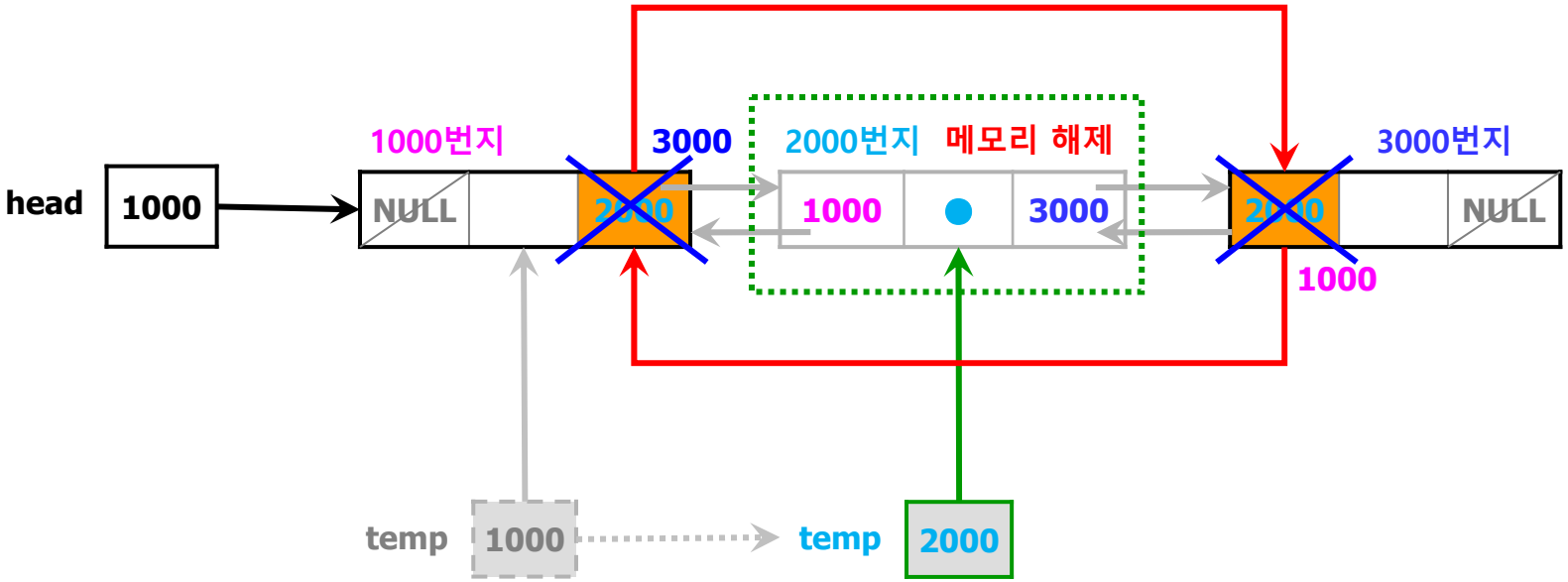
- **이중 연결 리스트:** 삭제 알고리즘
  - 리스트의 첫 번째 노드를 삭제




```
// 첫 번째 노드를 삭제
// deleteFirstDNode();
```

이중 연결 리스트 (7/8)

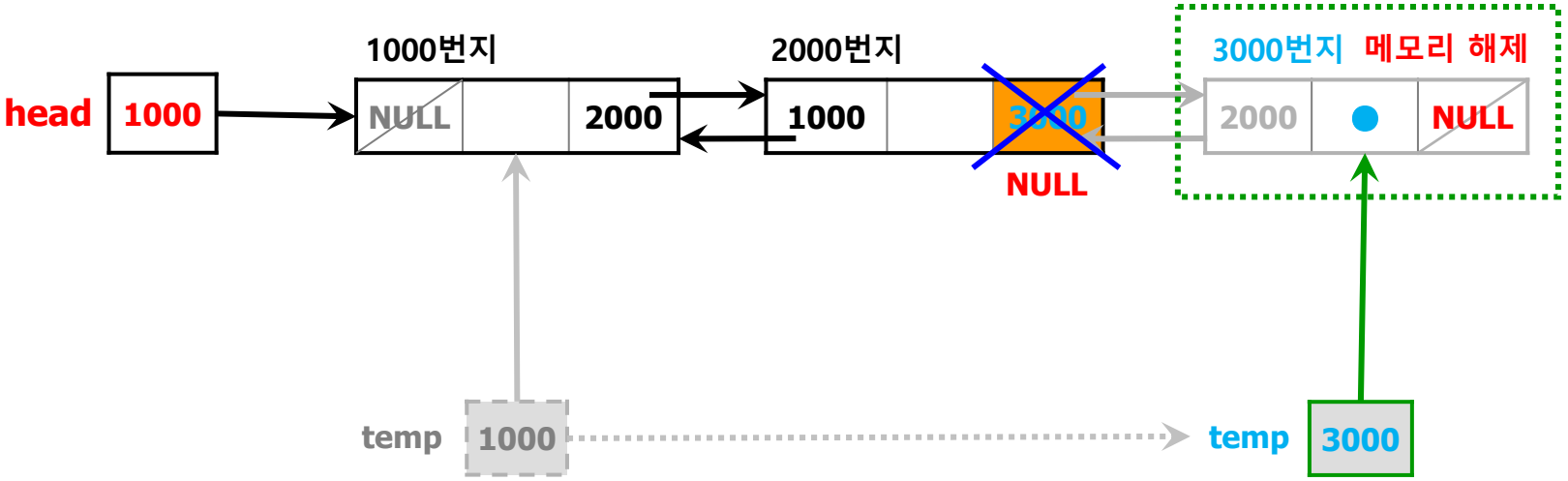
- **이중 연결 리스트:** 삭제 알고리즘
  - 리스트의 중간 노드를 삭제




```
// 중간 노드를 삭제
// deleteMiddleDNode();
```

이중 연결 리스트 (8/8)

- **이중 연결 리스트:** 삭제 알고리즘
  - 리스트의 마지막 노드를 삭제



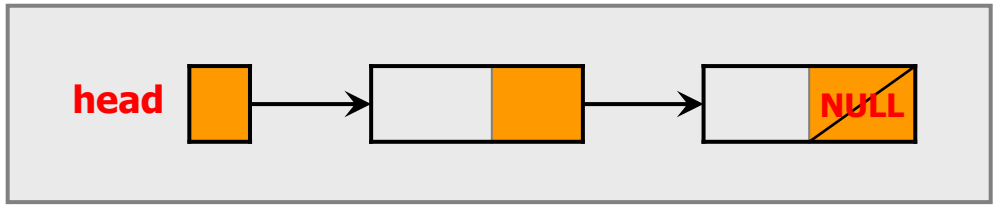

```
// 마지막 노드를 삭제
// deleteLastDNode();
```

연결 리스트 구현

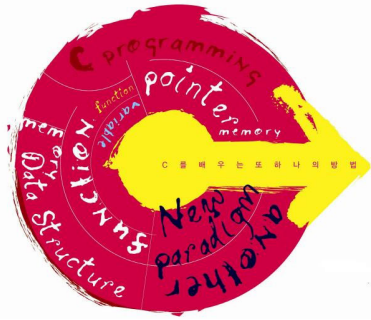
- 선형 리스트

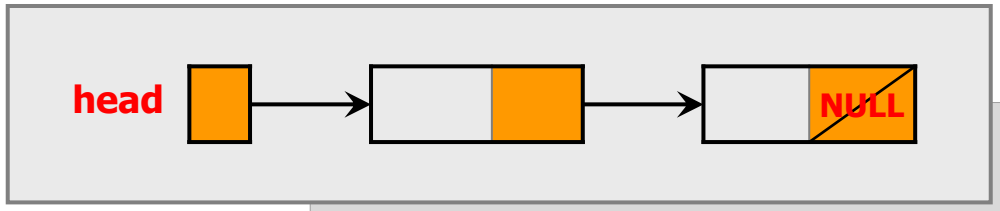
백문이불여일타(百聞而不如一打)

- 연결 리스트



- **연결 리스트 구현**
  - 단순 연결 리스트
  - 원형 연결 리스트
  - 이중 연결 리스트





연결 리스트

단순 연결 리스트: C

THE C PROGRAMMING LANGUAGE

단순 연결 리스트: C (1/12)

- **단순 연결 리스트: C**

```
// 파일명: SLinkedList(head).h
// #pragma once
#include "LinkedNode.h"    // SNode, makeSNode

// 구조체: SLinkedList
#ifndef __SLinkedList_H__
#define __SLinkedList_H__

// 단순 연결 리스트: SLinkedList
typedef struct _SLinkedList {
        SNode           *head;      // 첫 번째 노드
        // SNode        *tail;      // 맨 마지막 노드
        // int          count;      // 노드의 총 개수
} SLinkedList;

#endif

// 단순 연결 리스트 구현(C): 리스트 생성 및 조작 함수
SLinkedList   *sListCreate(void);
SLinkedList   *sListDestroy(SLinkedList *sList);
_Bool         sListEmpty(SLinkedList *sList);
int           countSNode(SLinkedList *sList);
SNode         *frontSNode(SLinkedList *sList);
SNode         *rearSNode(SLinkedList *sList);
void          sListAddRear(SLinkedList *sList, SNode *newNode);
void          sListRemoveFront(SLinkedList *sList);
void          printSLinkedList(SLinkedList *sList);
```

```
// 파일명: LinkedNode.h
// #pragma once
typedef     int             element;

#ifndef __SNode_H__
#define __SNode_H__

// 노드: SNode(data, link)
typedef struct _SNode {
        element             data;
        struct _SNode       *link;
} SNode;
#endif

SNode       *makeSNode(element data);
```

단순 연결 리스트: C (2/12)

예제 4-2: 단순 연결 리스트 SLinkedList(demo).c (1/2)

```c
#include <stdio.h>
#include <stdlib.h>                     // exit, malloc, free
#include <stdbool.h>                    // bool, true, false
#include "SLinkedList(head).h"          // SLinkedList >> head
// #include "SLinkedList(tail).h"       // SLinkedList >> head, tail, count
// #include "LinkedNode.h"              // SNode, makeSNode

int main(void)
{
        int             num;

        // head = NULL;
        SLinkedList     *sList = sListCreate();
        while (true) {
                printf("임의의 정수 입력(종료: 0): ");
                scanf_s("%d", &num);
                // scanf("%d", &num);
                if (num == 0)
                        break;

                SNode   *newNode = makeSNode(num);  // 새로운 노드 생성
                sListAddRear(sList, newNode);       // 맨 마지막 노드로 삽입
        }
```

```
Microsoft Visual Studio 디버그 콘솔
임의의 정수 입력(종료: 0): 1
임의의 정수 입력(종료: 0): 2
임의의 정수 입력(종료: 0): 3
임의의 정수 입력(종료: 0): 4
임의의 정수 입력(종료: 0): 5
임의의 정수 입력(종료: 0): 0

### 입력된 데이터 ###

 1 ->> 2 ->> 3 ->> 4 ->> 5 ->> NULL

노드의 총 개수: 5
첫 번째 노드의 데이터: 1
마지막 노드의 데이터 : 5
```

단순 연결 리스트: C (3/12)

예제 4-2: 단순 연결 리스트 SLinkedList(demo).c (2/2)

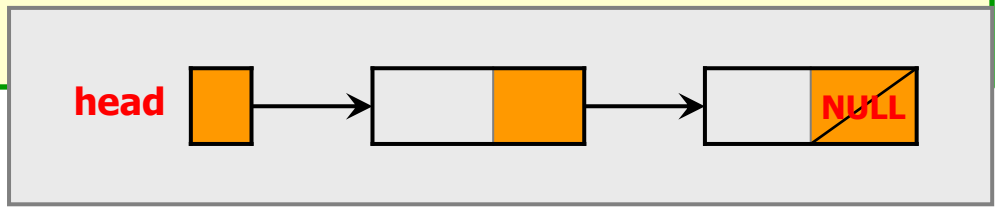
```
        // 리스트 전체 노드의 데이터 출력
        printSLinkedList(sList);

        // 노드의 총 개수
        printf("노드의 총 개수: %d\n", countSNode(sList));

        // 빈 리스트 여부 판단
        if (sListEmpty(sList)) {
                printf("입력된 데이터가 없습니다!!!\n");
        }
        else {
                // 첫 번째 노드와 마지막 노드의 데이터
                printf("첫 번째 노드의 데이터: %d\n", frontSNode(sList)->data);
                printf("마지막 노드의 데이터 : %d\n", rearSNode(sList)->data);
        }

        // sList = sListDestroy(sList);
        return 0;
}
```

```
Microsoft Visual Studio 디버그 콘솔
입의의 정수 입력(종료: 0): 1
입입의의의 정정수수 입입력력(종종료료: 0): 2
입입의의의 정정수수 입입력력(종종료료: 0): 3
입입의의의 정정수수 입입력력(종종료료: 0): 4
입입의의의 정정수수 입입력력(종종료료: 0): 5
입의의 정수 입력(종료: 0): 0

### 입력된 데이터 ###

 1 ->> 2 ->> 3 ->> 4 ->> 5 ->> NULL

노드의 총 개수: 5
첫 번째 노드의 데이터: 1
마지막 노드의 데이터 : 5
```

단순 연결 리스트: C (4/12)

예제 4-2: 단순 연결 리스트 LinkedNode.h

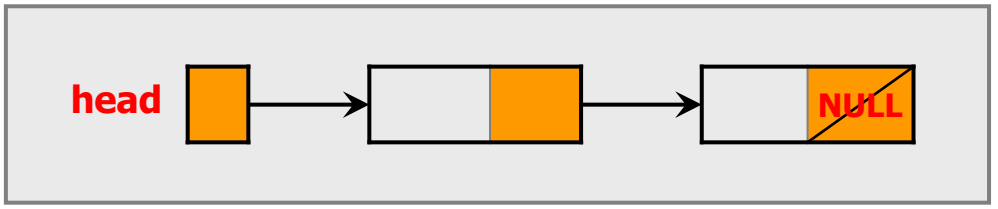
```
// #pragma once
typedef int    element;

// 단순 연결 리스트: SNode(data, link)
#ifndef __SNode_H__
#define __SNode_H__

// SNode(data, link)
typedef struct _SNode {
        element         data;
        struct _SNode   *link;
} SNode;

#endif

SNode   *makeSNode(element data);
```



단순 연결 리스트: C (5/12)

예제 4-2: 단순 연결 리스트 LinkedNode.c

```
#include <stdio.h>
#include <stdlib.h>                    // malloc, free
#include "LinkedNode.h"                // SNode

// 단순 연결 리스트: SNode(data, link)
// 새로운 노드(data, link) 생성
SNode  *makeSNode(element num) {
        SNode  *newNode = (SNode *)malloc(sizeof(SNode));
        if (newNode == NULL) {
                printf("노드 생성 실패!!! \n");
                exit(1);
        }
        newNode->data = num;
        newNode->link = NULL;
        return newNode;
}
```



단순 연결 리스트: C (6/12)

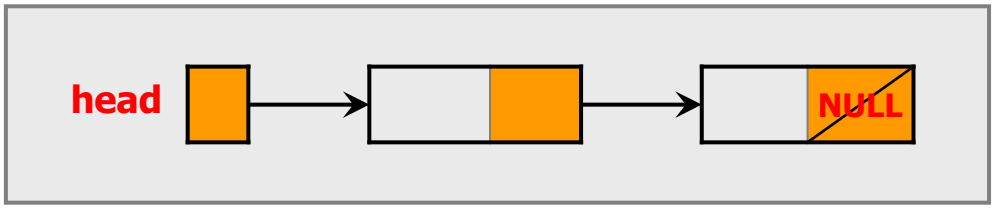
예제 4-2: 단순 연결 리스트 SLinkedList(head).h

```
// #pragma once
#include "LinkedNode.h"                 // SNode, makeSNode

// 구조체: SLinkedList
#ifndef __SLinkedList_H__
#define __SLinkedList_H__

typedef struct _SLinkedList {
        SNode           *head;          // 첫 번째 노드
        // SNode        *tail;          // 맨 마지막 노드
        // int          count;          // 노드의 총 개수
} SLinkedList;

#endif

// 단순 연결 리스트(C): 리스트 생성 및 조작 함수
SLinkedList    *sListCreate(void);
SLinkedList    *sListDestroy(SLinkedList *sList);
_Bool   sListEmpty(SLinkedList *sList);
int     countSNode(SLinkedList *sList);
SNode   *frontSNode(SLinkedList *sList);
SNode   *rearSNode(SLinkedList *sList);
void    sListAddRear(SLinkedList *sList, SNode *newNode);
void    sListRemoveFront(SLinkedList *sList);
void    printSLinkedList(SLinkedList *sList);
```

단순 연결 리스트: C (7/12)

예제 4-2: 단순 연결 리스트 SLinkedList(head).c (1/6)

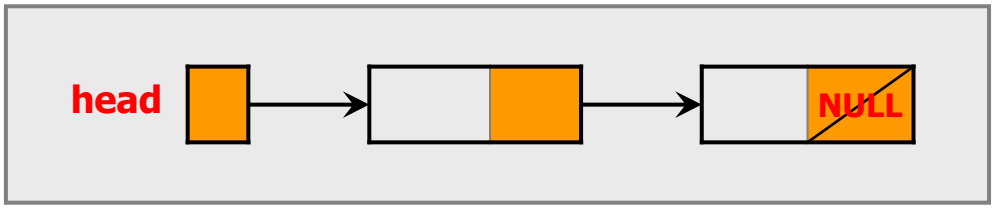
```
#include <stdio.h>
#include <stdlib.h>                    // exit, malloc, free
#include <stdbool.h>                   // bool, true, false
#include "SLinkedList(head).h"         // SLinkedList >> head
// #include "SLinkedList(tail).h"      // SLinkedList >> head, tail, count
// #include "LinkedNode.h"             // SNode, makeSNode

// 빈 리스트 생성
SLinkedList     *sListCreate(void) {
        SLinkedList     *sList = (SLinkedList *)malloc(sizeof(SLinkedList));
        if (sList == NULL) {
                printf("메모리 할당 실패!!! \n");
                exit(1);
        }
        sList->head = NULL;
        return sList;
}
```



단순 연결 리스트: C (8/12)

예제 4-2: 단순 연결 리스트 SLinkedList(head).c (2/6)

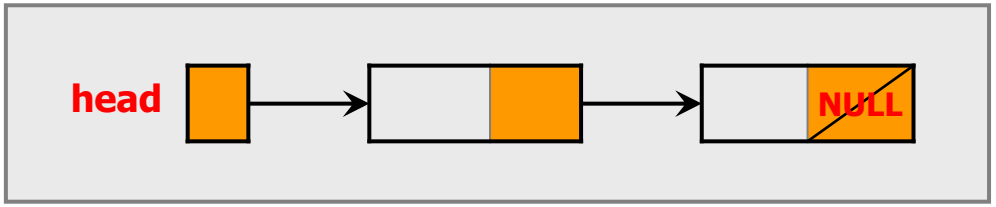
```
// 리스트 삭제: 리스트의 전체 노드 삭제
SLinkedList     *sListDestroy(SLinkedList *sList) {
        // while (!sListEmpty(sList))
        //      sListRemoveFront(sList);
        // free(sList);
        SNode   *tNode, *old;
        tNode = sList->head;
        while (tNode) {
                old = tNode;
                sList->head = tNode->link;
                free(old);
        }
        free(sList);
        return NULL;
}
```



단순 연결 리스트: C (9/12)

예제 4-2: 단순 연결 리스트 SLinkedList(head).c (3/6)

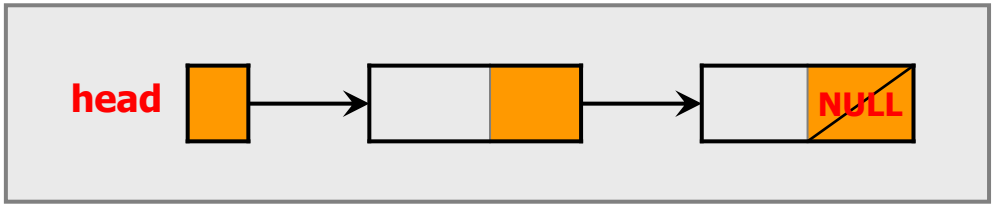
```
// 빈 리스트 여부 판단
_Bool    sListEmpty(SLinkedList *sList) {
         return sList->head == NULL;
}

// 탐색: 노드의 총 개수
int      countSNode(SLinkedList *sList) {
         int      count = 0;
         SNode    *rNode = sList->head;
         while (rNode) {
                  ++count;
                  rNode = rNode->link;
         }
         return count;
}
```



단순 연결 리스트: C (10/12)

예제 4-2: 단순 연결 리스트 SLinkedList(head).c (4/6)

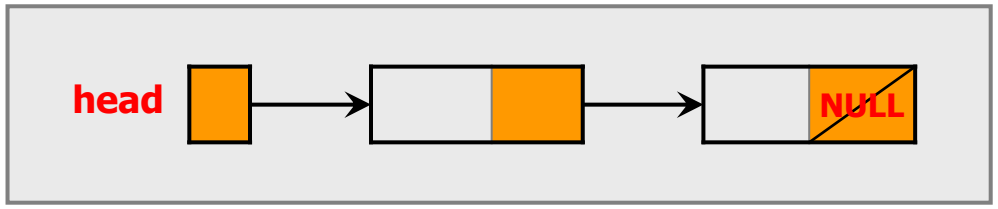
```
// 탐색: 리스트의 첫 번째 노드(head)
SNode   *frontSNode(SLinkedList *sList) {
        return sList->head;
}

// 탐색: 리스트의 맨 마지막 노드
SNode   *rearSNode(SLinkedList *sList) {
        if (sListEmpty(sList))
                return NULL;

        SNode   *rNode = sList->head;
        while (rNode->link)
                rNode = rNode->link;
        return rNode;
}
```



단순 연결 리스트: C (11/12)

예제 4-2: 단순 연결 리스트 SLinkedList(head).c (5/6)

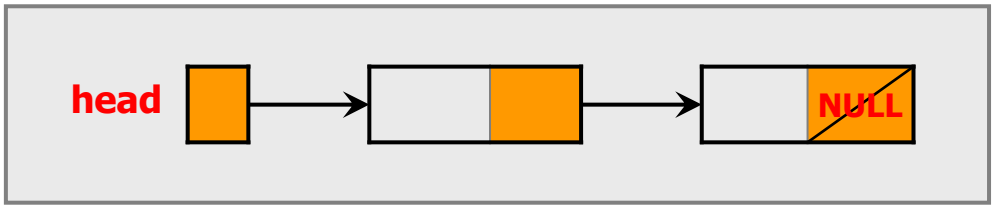
```
// 노드 삽입: 리스트의 맨 마지막 노드로 삽입한다.
void    sListAddRear(SLinkedList *sList, SNode *newNode) {
        if (sListEmpty(sList)) {
                sList->head = newNode;
        }
        else {
                SNode   *rNode = rearSNode(sList);
                rNode->link = newNode;
        }
}

// 노드 삭제: 리스트에서 첫 번째 노드를 삭제한다.
void    sListRemoveFront(SLinkedList *sList) {
        if (sListEmpty(sList))
                return;

        SNode   *old = sList->head;
        sList->head = old->link;
        free(old);
}
```



단순 연결 리스트: C (12/12)

예제 4-2: 단순 연결 리스트 SLinkedList(head).c (6/6)

```
// 출력: 리스트 전체 노드의 데이터
void    printSLinkedList(SLinkedList *sList) {
        if (sListEmpty(sList)) {
                printf("입력된 데이터가 없습니다... \n");
                return;
        }

        printf("\n ### 입력된 데이터 ### \n\n");
        SNode   *tNode = sList->head;
        while (tNode) {
                printf("%3d ->>", tNode->data);
                tNode = tNode->link;
        }
        printf(" NULL\n");
}
```

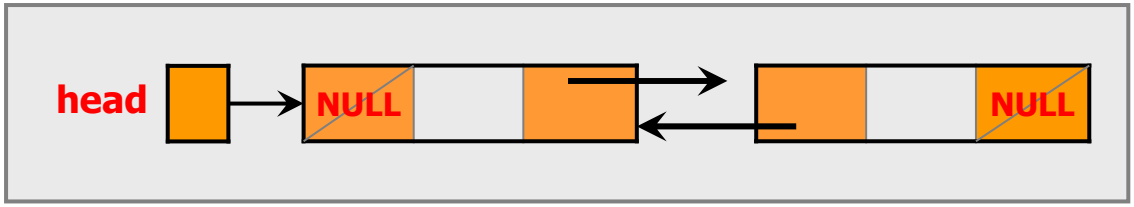


연결 리스트 구현

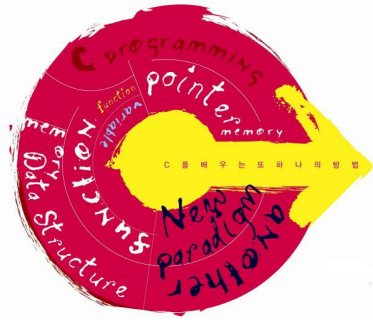
- 선형 리스트

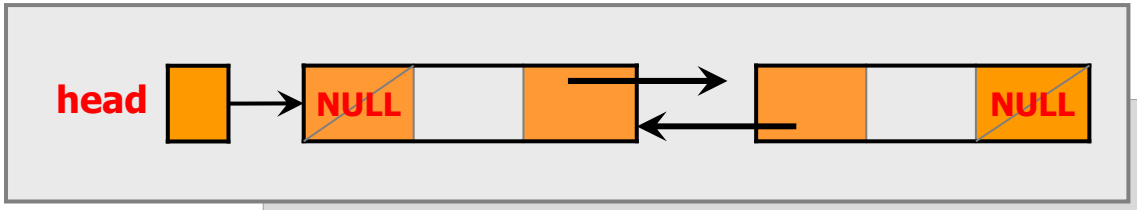
백문이불여일타(百聞而不如一打)

- 연결 리스트



- **연결 리스트 구현**
  - 단순 연결 리스트
  - 원형 연결 리스트
  - **이중 연결 리스트**





연결 리스트

이중 연결 리스트: C

THE C PROGRAMMING LANGUAGE

이중 연결 리스트: C (1/12)

● 이중 연결 리스트: C

```
// 파일명: DLinkedList(head).h
// #pragma once
#include "LinkedNode.h"   // DNode, makeDNode

// 구조체: DLinkedList
#ifndef __DLinkedList_H__
#define __DLinkedList_H__

// 이중 연결 리스트: DLinkedList
typedef struct _DLinkedList {
      DNode          *head;      // 첫 번째 노드
      // DNode       *tail;      // 맨 마지막 노드
      // int         count;      // 노드의 총 개수
} DLinkedList;

#endif

// 이중 연결 리스트 구현: 리스트 생성 및 조작 함수
DLinkedList  *dListCreate(void);
DLinkedList  *dListDestroy(DLinkedList *dList);
_Bool        dListEmpty(DLinkedList *dList);
int          countDNode(DLinkedList *dList);
DNode        *frontDNode(DLinkedList *dList);
DNode        *rearDNode(DLinkedList *dList);
void         dListAddRear(DLinkedList *dList, DNode *newNode);
void         dListRemoveFront(DLinkedList *dList);
void         printDLinkedList(DLinkedList *dList);
void         printRevDLinkedList(DLinkedList *dList);
```

```
// 파일명: LinkedNode.h
// #pragma once
typedef      int            element;

#ifndef __DNode_H__
#define __DNode_H__

// 노드: DNode(data, Llink, Rlink)
typedef struct _DNode {
      element          data;
      struct _DNode    *Llink;
      struct _DNode    *Rlink;
} DNode;

#endif

DNode* makeDNode(element data);
```

이중 연결 리스트: C (2/12)

예제 4-6: 이중 연결 리스트 DLinkedList(demo).c (1/2)

```c
#include <stdio.h>
#include <stdlib.h>                     // exit, malloc, free
#include <stdbool.h>                    // bool, true, false
#include "DLinkedList(head).h"          // DLinkedList >> head
// #include "DLinkedList(tail).h"       // DLinkedList >> head, tail, count
// #include "LinkedNode.h"              // DNode, makeDNode

int main(void)
{
        int             num;

        // head = NULL, tail = NULL, count = 0;
        DLinkedList     *dList = dListCreate();
        while (true) {
                printf("임의의 정수 입력(종료: 0): ");
                scanf_s("%d", &num);
                // scanf("%d", &num);
                if (num == 0)
                        break;

                DNode   *newNode = makeDNode(num);  // 새로운 노드 생성
                dListAddRear(dList, newNode);       // 맨 마지막 노드로 삽입
        }
```

```
Microsoft Visual Studio 디버그 콘솔
임의의 정수 입력(종료: 0) : 1
임의의 정수 입력(종료: 0) : 2
임의의 정수 입력(종료: 0) : 3
임의의 정수 입력(종료: 0) : 4
임의의 정수 입력(종료: 0) : 5
임의의 정수 입력(종료: 0) : 0

### 입력된 데이터(순방향) ###
 1 ->> 2 ->> 3 ->> 4 ->> 5 ->> NULL

### 입력된 데이터(역방향) ###
 5 ->> 4 ->> 3 ->> 2 ->> 1 ->> NULL

노드의 총 개수: 5
첫 번째 노드의 데이터: 1
마지막 노드의 데이터 : 5
```

이중 연결 리스트: C (3/12)

예제 4-6: 이중 연결 리스트 DLinkedList(demo).c (2/2)

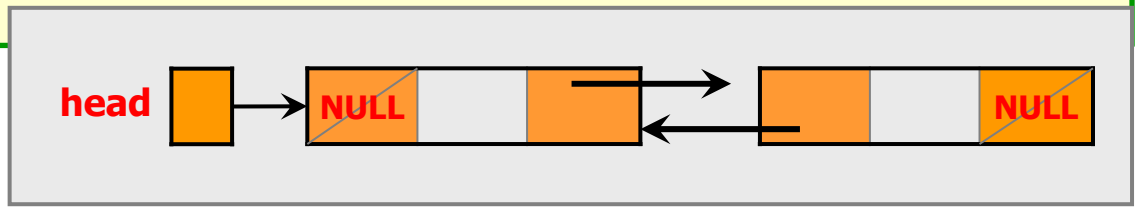
```
        // 리스트 전체 노드의 데이터 출력
        printDLinkedList(dList);          // 순방향 출력
        printRevDLinkedList(dList);       // 역방향 출력

        // 노드의 총 개수
        printf("노드의 총 개수: %d\n", countSNode(dList));

        // 빈 리스트 여부 판단
        if (sListEmpty(dList)) {
                printf("입력된 데이터가 없습니다!!!\n");
        }
        else {
                // 첫 번째 노드와 마지막 노드의 데이터
                printf("첫 번째 노드의 데이터: %d\n", frontDNode(dList)->data);
                printf("마지막 노드의 데이터 : %d\n", rearDNode(dList)->data);
        }

        // dList = dListDestroy(dList);
        return 0;
}
```

```
Microsoft Visual Studio 디버그 콘솔
임의의 정수 입력(종료: 0) : 1
임의의 정수 입력(종료: 0) : 2
임의의 정수 입력(종료: 0) : 3
임의의 정수 입력(종료: 0) : 4
임의의 정수 입력(종료: 0) : 5
임의의 정수 입력(종료: 0) : 0

### 입력된 데이터(순방향) ###
 1 ->>  2 ->>  3 ->>  4 ->>  5 ->> NULL

### 입력된 데이터(역방향) ###
 5 ->>  4 ->>  3 ->>  2 ->>  1 ->> NULL

노드의 총 개수: 5
첫 번째 노드의 데이터: 1
마지막 노드의 데이터 : 5
```

이중 연결 리스트: C (4/12)

예제 4-6: 이중 연결 리스트 LinkedNode.h

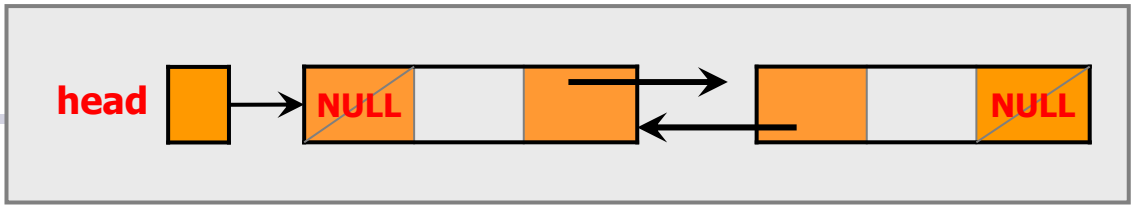
```
// #pragma once
typedef int    element;

// 이중 연결 리스트 구현(C)
// 노드: DNode(data, Llink, Rlink)
#ifndef __DNode_H__
#define __DNode_H__

typedef struct _DNode {
        element         data;
        struct _DNode   *Llink;
        struct _DNode   *Rlink;
} DNode;

#endif

DNode  *makeDNode(element data);
```



이중 연결 리스트: C (5/12)

예제 4-6: 이중 연결 리스트 LinkedNode.c

```
#include <stdio.h>
#include <stdlib.h>          // malloc, free
#include "LinkedNode.h"      // DNode

// 이중 연결 리스트 구현(C)
// 새로운 노드(DNode: data, Llink, Rlink) 생성
DNode  *makeDNode(element num) {
        DNode  *newNode = (DNode *)malloc(sizeof(DNode));
        if (newNode == NULL) {
                printf("노드 생성 실패!!! \n");
                exit(1);
        }
        newNode->data = num;
        newNode->Llink = NULL;
        newNode->Rlink = NULL;
        return newNode;
}
```



이중 연결 리스트: C (6/12)

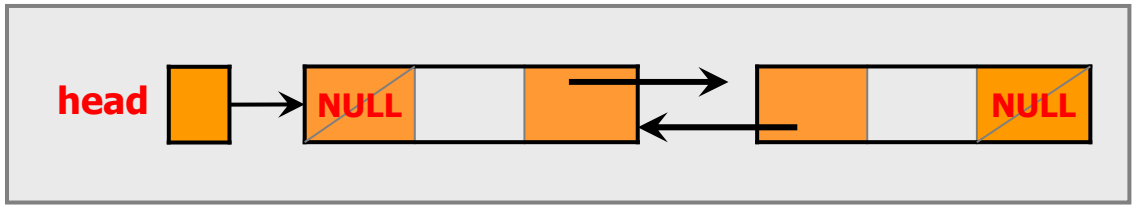
예제 4-6: 이중 연결 리스트 DLinkedList(head).h

```
// #pragma once
#include "LinkedNode.h"                 // DNode, makeDNode

// 구조체: DLinkedList
#ifndef __DLinkedList_H__
#define __DLinkedList_H__

typedef struct _DLinkedList {
        DNode           *head;          // 첫 번째 노드
        // DNode        *tail;          // 맨 마지막 노드
        // int          count;          // 노드의 총 개수
} DLinkedList;

#endif

// 이중 연결 리스트 구현(C): 리스트 생성 및 조작 함수
DLinkedList    *dListCreate(void);
DLinkedList    *dListDestroy(DLinkedList *dList);
_Bool   dListEmpty(DLinkedList *dList);
int     countDNode(DLinkedList *dList);
DNode   *frontDNode(DLinkedList *dList);
DNode   *rearDNode(DLinkedList *dList);
void    dListAddRear(DLinkedList *dList, DNode *newNode);
void    dListRemoveFront(DLinkedList *dList);
void    printDLinkedList(DLinkedList *dList);
void    printRevDLinkedList(DLinkedList *dList);
```

이중 연결 리스트: C (7/12)

예제 4-6: 이중 연결 리스트 DLinkedList(head).c (1/6)

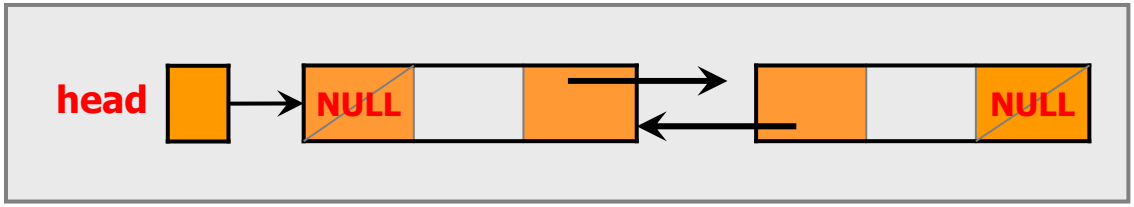
```
#include <stdio.h>
#include <stdlib.h>                    // exit, malloc, free
#include <stdbool.h>                   // bool, true, false
#include "DLinkedList(head).h"         // DLinkedList >> head
// #include "DLinkedList(tail).h"      // DLinkedList >> head, tail, count
// #include "LinkedNode.h"             // DNode, makeDNode

// 빈 리스트 생성
DLinkedList     *dListCreate(void) {
        DLinkedList     *dList = (DLinkedList *)malloc(sizeof(DLinkedList));
        if (dList == NULL) {
                printf("메모리 할당 실패!!! \n");
                exit(1);
        }
        dList->head = NULL;
        return dList;
}
```



이중 연결 리스트: C (8/12)

예제 4-6: 이중 연결 리스트 DLinkedList(head).c (2/6)

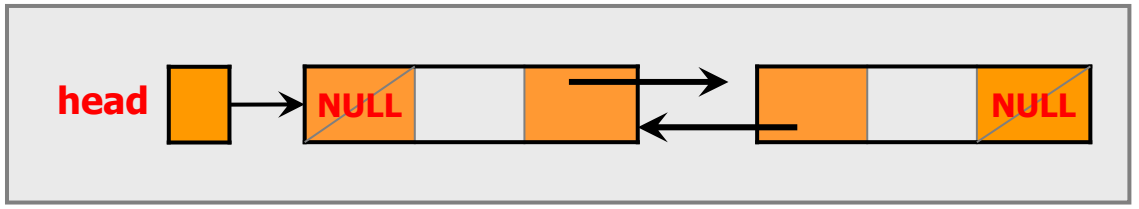
```
// 리스트 삭제: 리스트의 전체 노드 삭제
DLinkedList     *dListDestroy(DLinkedList *dList) {
        // while (!dListEmpty(sList))
        //      dListRemoveFront(sList);
        // free(dList);
        DNode  *tNode, *old;
        tNode = dList->head;
        while (tNode) {
                old = tNode;
                tNode = tNode->Llink;
                free(old);
        }
        free(dList);
        return NULL;
}
```



이중 연결 리스트: C (9/12)

예제 4-6: 이중 연결 리스트 DLinkedList(head).c (3/6)

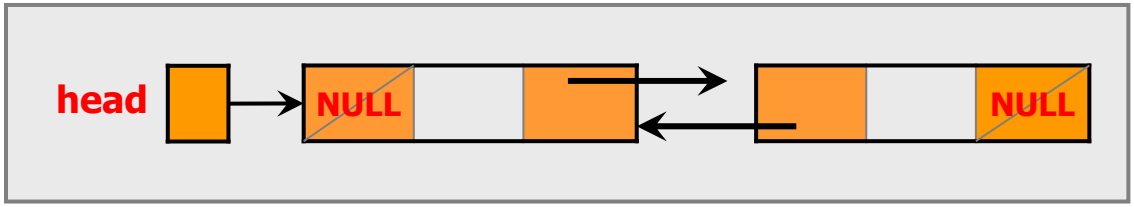
```
// 빈 리스트 여부 판단
_Bool    dListEmpty(DLinkedList *dList) {
         return dList->head == NULL;
}

// 탐색: 노드의 총 개수
int      countDNode(DLinkedList *dList) {
         int      count = 0;
         DNode    *rNode = dList->head;
         while (rNode) {
                  ++count;
                  rNode = rNode->Rlink;
         }
         return count;
}
```



이중 연결 리스트: C (10/12)

예제 4-6: 이중 연결 리스트 DLinkedList(head).c (4/6)

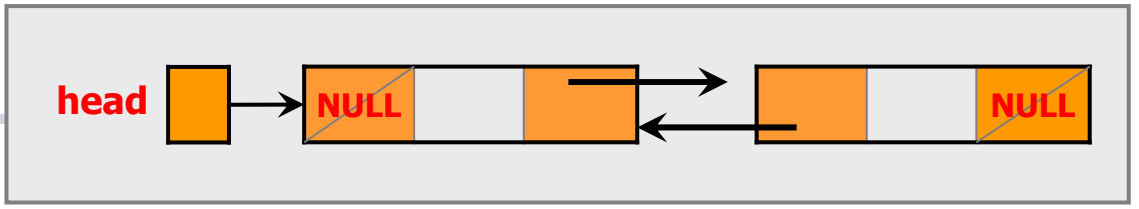
```
// 탐색: 리스트의 첫 번째 노드(head)
DNode  *frontDNode(DLinkedList *dList) {
        return dList->head;
}

// 탐색: 리스트의 맨 마지막 노드
DNode  *rearDNode(DLinkedList *dList) {
        if (dListEmpty(dList))
                return NULL;

        DNode  *rNode = dList->head;
        while (rNode->Rlink)
                rNode = rNode->Rlink;
        return rNode;
}
```



이중 연결 리스트: C (11/12)

예제 4-6: 이중 연결 리스트 DLinkedList(head).c (5/6)

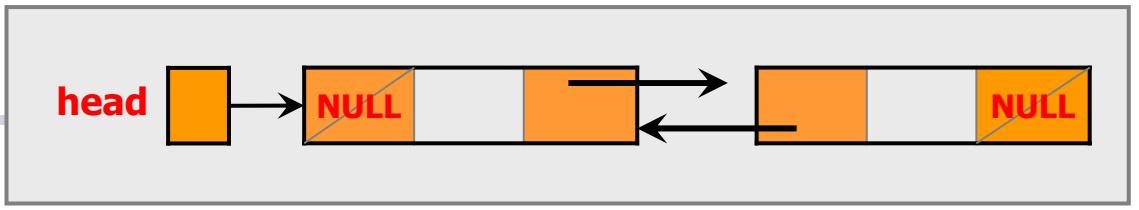
```
// 삽입: 리스트의 맨 마지막 노드로...
void    dListAddRear(DLinkedList *dList, DNode *newNode) {
        if (dListEmpty(dList))
                dList->head = newNode;
        else {
                DNode  *rNode = rearDNode(dList);
                rNode->Rlink = newNode;
                newNode->Llink = rNode;
        }
}

// 삭제: 리스트에서 첫 번째 노드를...
void    dListRemoveFront(DLinkedList *dList) {
        if (dListEmpty(dList))
                return;

        DNode  *old = dList->head;
        dList->head = old->Rlink;
        if (dList->head != NULL)
                dList->head->Llink = NULL;

        free(old);
}
```



이중 연결 리스트: C (12/12)

예제 4-6: 이중 연결 리스트 DLinkedList(head).c (6/6)

```
// 출력: 리스트 전체 노드의 데이터(순방향)
void    printDLinkedList(DLinkedList *dList) {
        if (dListEmpty(dList)) {
                printf("입력된 데이터가 없습니다!!!\n");
                return;
        }
        printf("\n ### 입력된 데이터(순방향) ### \n\n");
        DNode* tNode = dList->head;
        while (tNode) {
                printf("%3d ->>", tNode->data);
                tNode = tNode->Rlink;
        }
        printf(" NULL\n\n");
}

// 출력: 리스트 전체 노드의 데이터(역방향)
void    revPrintDLinkedList(DLinkedList *dList) {
        if (dListEmpty(dList)) {
                printf("입력된 데이터가 없습니다!!!\n");
                return;
        }
        printf("\n ### 입력된 데이터(역방향) ### \n\n");
        DNode  *rNode = rearDNode(dList);
        while (rNode) {
                printf("%3d ->>", rNode->data);
                rNode = rNode->Llink;
        }
        printf(" NULL\n\n");
}
```



참고문헌

[1] 서현우, "혼자 공부하는 C 언어: 1:1 과외 하듯 배우는 프로그래밍 자습서", 한빛미디어, 2023.

[2] Paul Deitel, Harvey Deitel, "C How to Program", Global Edition, 8/E, Pearson, 2016.

[3] Kamran Amini, 박지윤 번역, "전문가를 위한 C : 동시성, OOP부터 최신 C, 고급 기능까지!", 한빛미디어, 2022.

[4] 서두옥, "(열혈강의) 또 하나의 C : 프로그래밍은 셀프입니다", 프리렉, 2012.

[5] Behrouz A. Forouzan, Richard F. Gilberg, 김진 외 7인 공역, "구조적 프로그래밍 기법을 위한 C", 도서출판 인터비젼, 2004.

[6] Brian W. Kernighan, Dennis M. Ritchie, 김석환 외 2인 공역, "The C Programming Language", 2/E, 대영사, 2004.

[7] "이것이 자료구조+알고리즘이다: with C 언어", 박상현, 한빛아카데미, 2022.

[8] 문병로, "IT CookBook, 쉽게 배우는 알고리즘: 관계 중심의 사고법"(개정판), 개정판, 한빛아카데미, 2018.

[9] "C reference", cppreference.com, 2024 of viewing the site, https://en.cppreference.com/w/c.