

자료구조 및 알고리즘

for(A;B;C)
D;

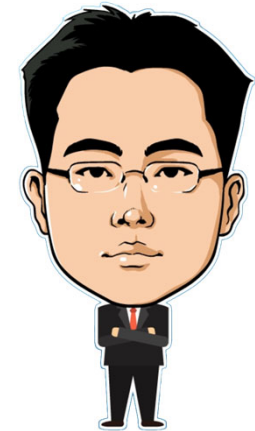


리스트
(List)

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

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

백문이불여일타(百聞而不如一打)

- 연결 리스트

- 단순 연결 리스트

- 원형 연결 리스트

- 이중 연결 리스트

- 연결 리스트 구현



연결 리스트 (1/5)

● 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.

○ 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 붙여서 추가한다.
copy()	◀ 리스트를 복사한다.
reverse()	◀ 리스트의 순서를 역으로 뒤집는다.
sort()	◀ 리스트의 원소들을 정렬한다.

연결 리스트 (2/5)

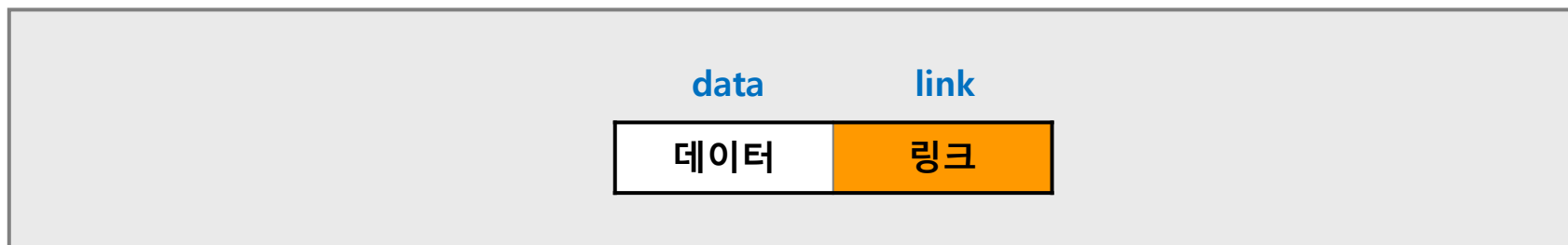
- **연결 리스트**(Linked List)

- 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 (3/5)

● 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.

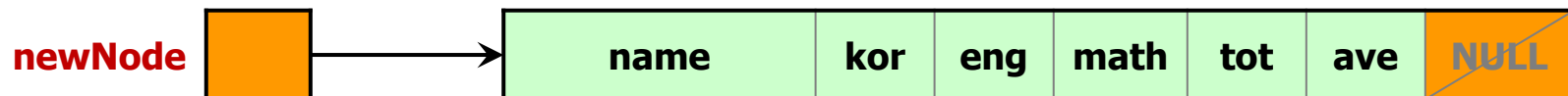
연결 리스트 (4/5)

- 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);  
}
```

THE
C
PROGRAMMING
LANGUAGE



연결 리스트 (5/5)

- **C++ STL: forward_list, list**

- 컨테이너 라이브러리(Containers Library)

- 순차 컨테이너(Sequence Containers)

- `<array>` : 정적 연속 배열(static contiguous array) (since C++11)
- `<vector>` : 동적 연속 배열(dynamic contiguous array)
- `<deque>` : 덱(double-ended queue)
- `<forward_list>` : 단일 연결 리스트(singly-linked list) (since C++11)
- `<list>` : 이중 연결 리스트(doubly-linked list)



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

단순 연결 리스트: 알고리즘

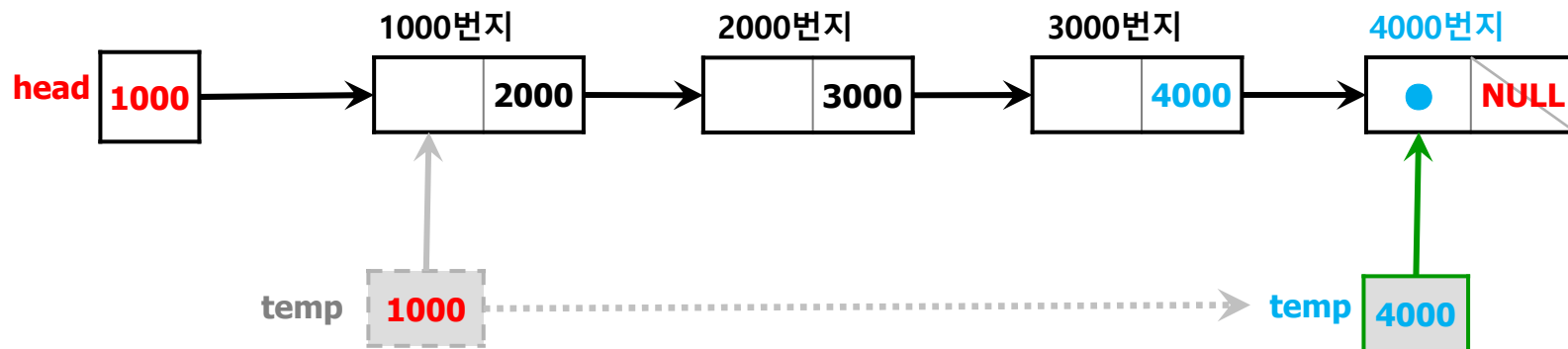


단순 연결 리스트 (1/8)

- 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

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

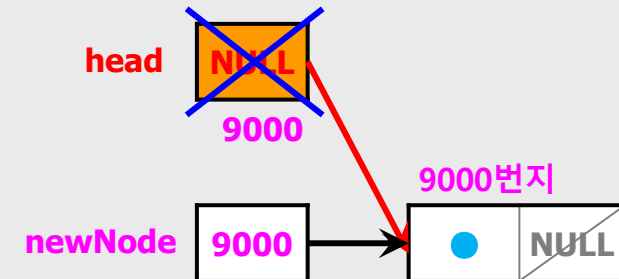
```
    newNode ← makeSNode(data);
```

```
    newNode.link = head;
```

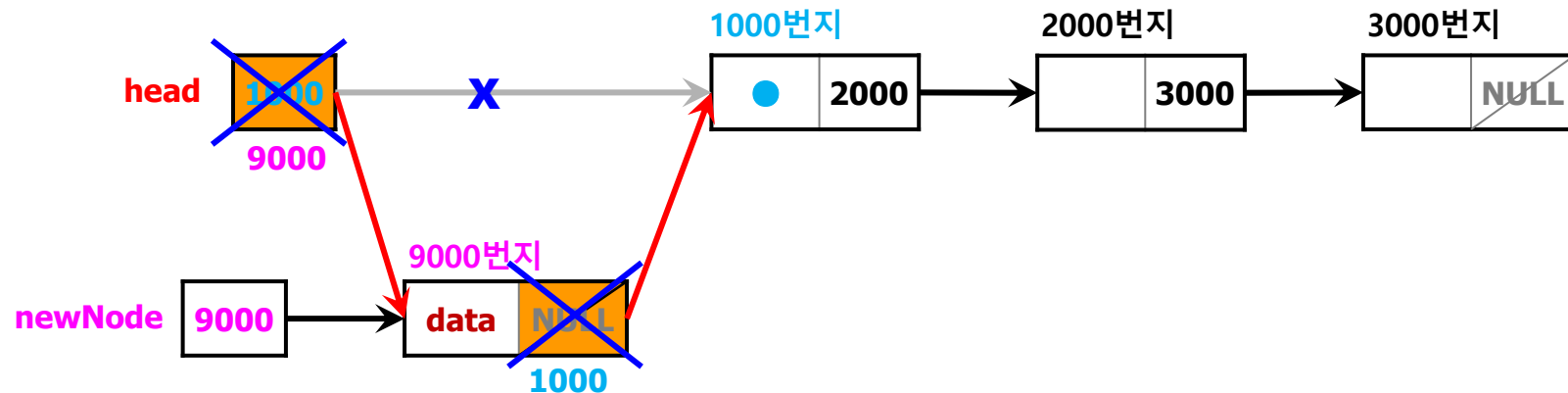
```
    head ← newNode;
```

```
end insertFirstNode()
```

// 빈 리스트일 경우...



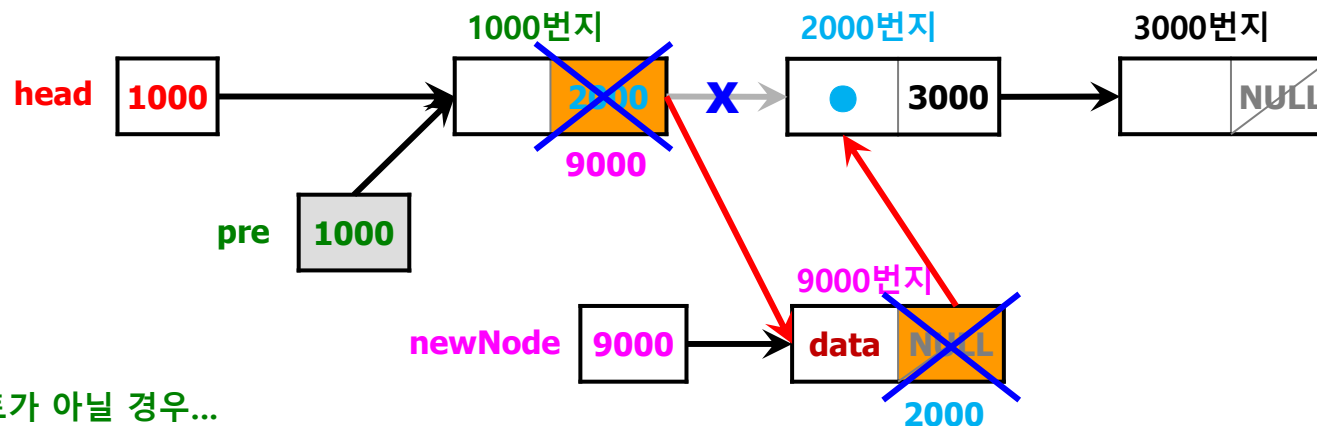
// 빈 리스트가 아닐 경우...



단순 연결 리스트 (3/8)

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

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

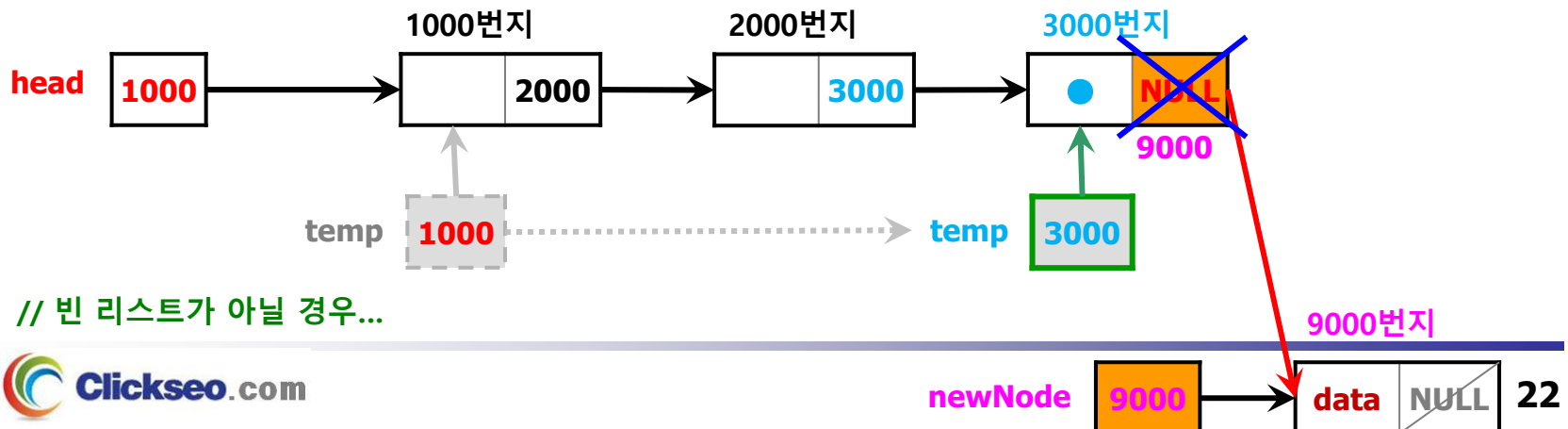


// 빈 리스트가 아닐 경우...

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

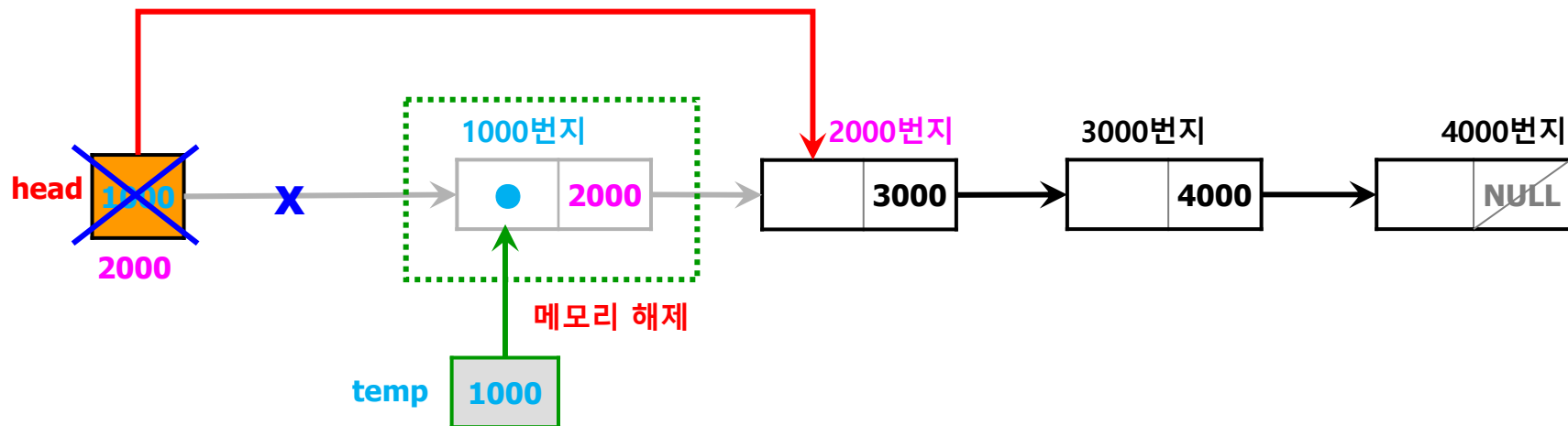
```
deleteSNode(head, data)
  if (head = NULL) then error;
  else {
    temp ← head;
    while (temp != NULL) {
      if (temp.data = data) then {
        if (temp = head) then deleteFirstNode();
        else if (temp = NULL) then deleteLastNode();
        else deleteMiddleNode();
      }
      pre ←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

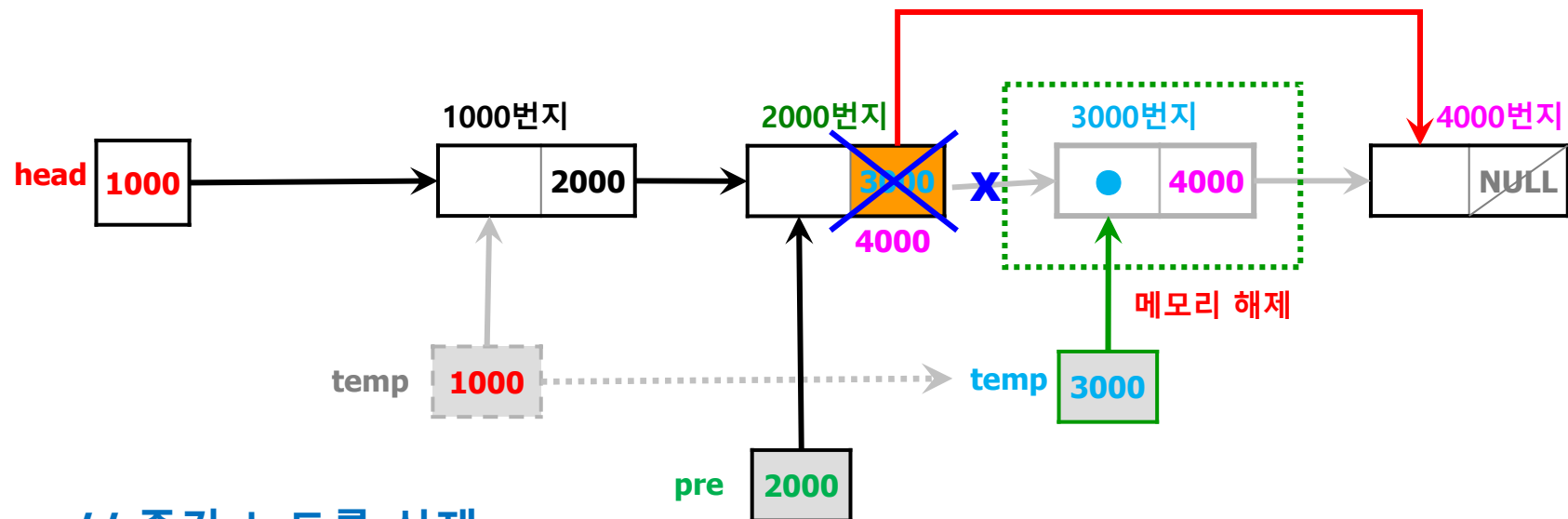
// deleteFirstSNode();

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

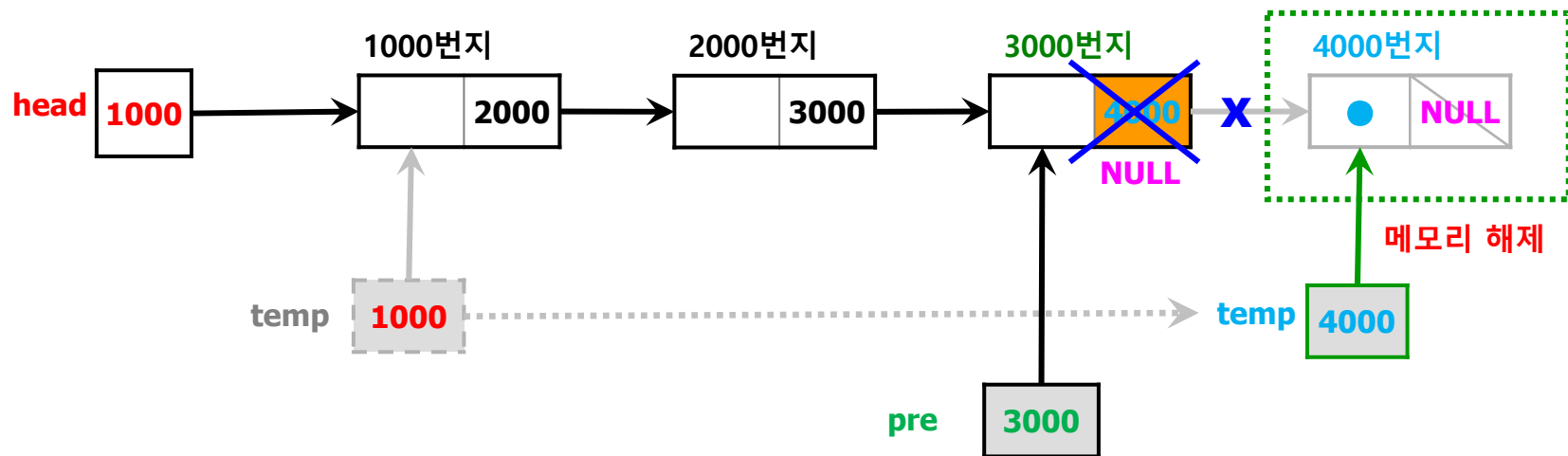
// deleteMiddleSNode();

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();

연결 리스트

단순 연결 리스트

: forward_list 클래스



C++ STL: forward_list 클래스 (1/3)

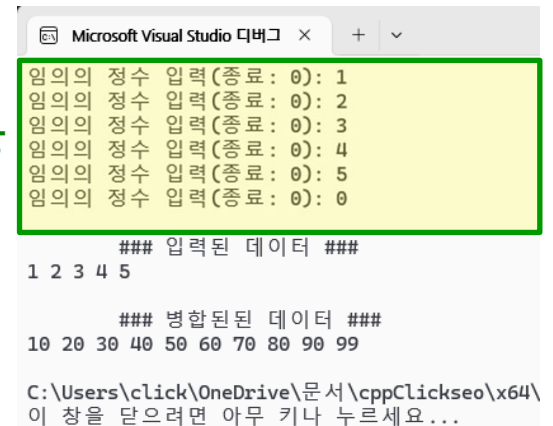
예제 4-1: 단순 연결 리스트 -- C++ STL(forward_list) (1/3) | C++

```
#include <iostream>
#include <forward_list>      // forward_list
using namespace std;

int main(void) {
    int num;
    forward_list<int> tList;
    // forward_list<int> tList = { 10, 20, 30, 40, 50 };
    // tList.assign( { 60, 70, 80, 90, 99 } );    // 새 원소 할당

    while (true) {
        cout << "임의의 정수 입력(종료: 0): ";
        cin >> num;
        if (num == 0)
            break;
    }
```

```
    // 맨 마지막 원소로 추가한다.
    auto it = tList.before_begin();
    for (auto next = tList.begin(); next != tList.end(); ++it, ++next);
    tList.insert_after(it, num);
    // tList.insert_after(tList.end(), num);    // Error!!!
}
```



Microsoft Visual Studio 디버그 콘솔 출력:

```
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

### 병합된 데이터 ###
10 20 30 40 50 60 70 80 90 99

C:\Users\click\OneDrive\문서\cppClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```

C++ STL: forward_list 클래스 (2/3)

예제 4-1: 단순 연결 리스트 -- C++ STL(forward_list) (2/3) | C++

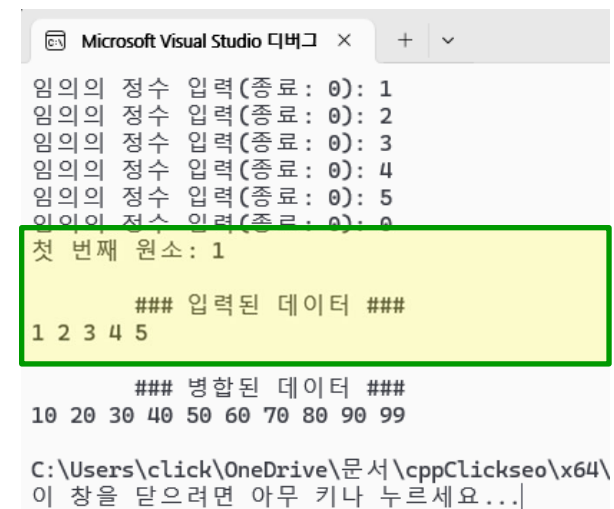
```
cout << "첫 번째 원소: " << tList.front() << endl;
```

```
// 전체 원소 출력
if (tList.empty()) {
    cout << "\n입력된 데이터가 없습니다!!!" << endl;
}
else {
    cout << "\n\t### 입력된 데이터 ###" << endl;
    for (int num: tList)
        cout << num << ' ';
    cout << endl;
}
```

```
// 첫 번째 원소로 추가한다.
// tList.push_front(num);
```

```
// 첫 번째 원소를 삭제한다.
// tList.pop_front();
// tList.erase_after(tList.before_begin());
// tList.erase(tList.begin());
```

// Error: No Function erase().



Microsoft Visual Studio 디버그

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

첫 번째 원소: 1

입력된 데이터 ###
1 2 3 4 5

병합된 데이터 ###
10 20 30 40 50 60 70 80 90 99

C:\Users\click\OneDrive\문서\cppClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...

C++ STL: forward_list 클래스 (3/3)

예제 4-1: 단순 연결 리스트 -- C++ STL(forward_list) (3/3) | C++

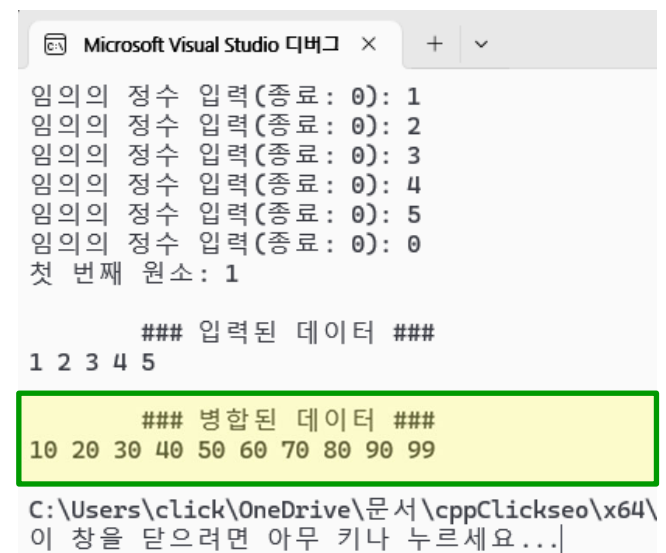
```
// 새로운 forward_list: aList, bList
forward_list<int>      aList = { 50, 20, 10, 40, 30 };
forward_list<int>      bList = { 60, 80, 99, 90, 70 };

aList.sort();          // 리스트를 정렬한다.
bList.sort();
// list.reverse();     // 리스트 원소의 순서를 반전 시킨다.

// merge : 서로 다른 forward_list를 병합한다.
// 단, 두 리스트는 사전에 정렬되어 있어야 한다.
aList.merge(bList);

cout << "\n\t### 병합된 데이터 ###" << endl;
for (int num : aList)
    cout << num << ' ';
cout << '\n';

// 모든 원소를 삭제한다.
tList.clear();
aList.clear();
bList.clear();
return 0;
}
```



Microsoft Visual Studio 디버그 콘솔의 출력 내용입니다. 프로그램은 두 forward_list를 정렬하고 병합한 후, 모든 원소를 삭제하는 과정을 보여줍니다. 병합된 데이터는 10 20 30 40 50 60 70 80 90 99로 출력되었습니다.

```
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
첫 번째 원소: 1

### 입력된 데이터 ###
1 2 3 4 5

### 병합된 데이터 ###
10 20 30 40 50 60 70 80 90 99

C:\Users\click\OneDrive\문서\cppClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```

연결 리스트

단순 연결 리스트: Python, C/C++



단순 연결 리스트: Python, C/C++ (1/3)

● 단순 연결 리스트: Python

```
// 노드: SNode(data, link)
class SNode:
    def __init__(self, data):
        self._data = data
        self._link = None
```

```
    def getData(self): return self._data
    def getLink(self): return self._link
    def setData(self, data): self._data = data
    def setLink(self, link): self._link = link
```

```
# 단순 연결 리스트: SLinkedList
```

```
class SLinkedList:
    def __init__(self):
        self._head = None
        # self._tail = None
        # self._count = 0
```

```
    def __del__(self):
    def isEmpty(self) -> bool:
    def countNode(self) -> int:
    def frontNode(self) -> SNode:
    def rearNode(self) -> SNode:
    def addRear(self, num) -> None:
    def removeFront(self) -> None:
    def printLinkedList(self) -> None:
```

```
# 생성자
# 첫 번째 노드
# 맨 마지막 노드
# 노드의 총 개수
```

```
# 소멸자: 전체 노드 삭제
# 빈 리스트 여부 판단
# 탐색: 노드의 총 개수(count)
# 탐색: 첫 번째 노드
# 탐색: 맨 마지막 노드
# 삽입: 맨 마지막 노드
# 삭제: 첫 번째 노드
# 리스트의 전체 노드 출력
```



단순 연결 리스트: Python, C/C++ (2/3)

● 단순 연결 리스트: C++

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

```
#ifndef __SLinkedList_H__
#define __SLinkedList_H__
```

```
// 단순 연결 리스트: SLinkedList
```

```
class SLinkedList {
public:
    SLinkedList(void);
    ~SLinkedList(void);
    bool isEmpty(void) const;
    int countNode(void) const;
    SNode *frontNode(void) const;
    SNode *rearNode(void) const;
    void addRear(const int &e);
    void removeFront(void);
    void printLinkedList(void) const;
```

```
private:
    SNode *head_; // 첫 번째 노드
    // SNode *tail_; // 맨 마지막 노드
    // int count_; // 노드의 총 개수
};
```

```
#endif
```

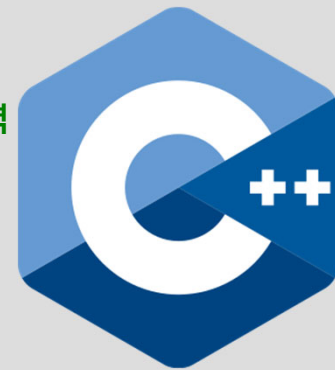
```
// 파일명: ListNode.h
// #pragma once
```

```
#ifndef __SNode_H__
#define __SNode_H__
```

```
// 노드: SNode(data, link)
class SNode {
public:
    SNode(const int& data);
    int getData(void) const;
private:
    int data_;
    SNode *link_;
    friend class SLinkedList;
};
```

```
#endif
```

```
// 생성자
// 소멸자: 전체 노드 삭제
// 빈 리스트 여부 판단
// 탐색: 노드의 총 개수(count_)
// 탐색: 첫 번째 노드(head_)
// 탐색: 맨 마지막 노드(tail_)
// 삽입: 맨 마지막 노드(tail_)
// 삭제: 첫 번째 노드(head_)
// 리스트의 전체 원소(노드) 출력
```



단순 연결 리스트: Python, C/C++ (3/3)

● 단순 연결 리스트: C

```
// 파일명: SLinkedList(head).h
// #pragma once
#include "ListNode.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: 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: ListNode.h
// #pragma once
typedef int element;

#ifndef __SNode_H__
#define __SNode_H__

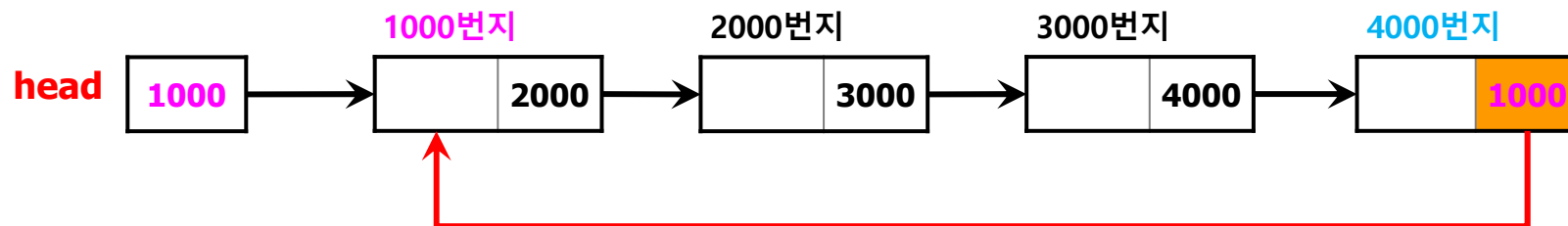
// 노드: SNode(data, link)
typedef struct _SNode {
    element data;
    struct _SNode *link;
} SNode;
#endif

SNode *makeSNode(element data);
```

THE
C
PROGRAMMING
LANGUAGE

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

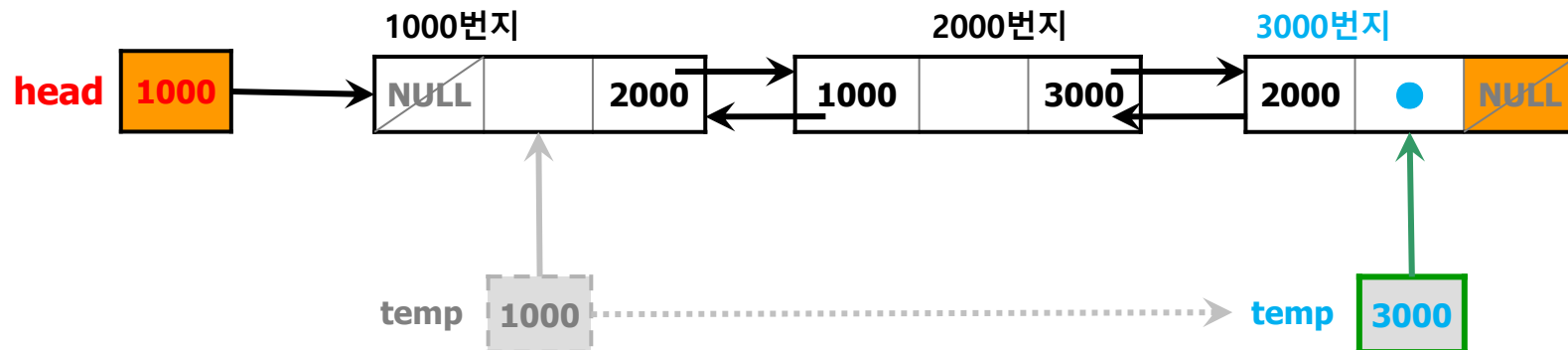


이중 연결 리스트 (1/8)

- 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()
```



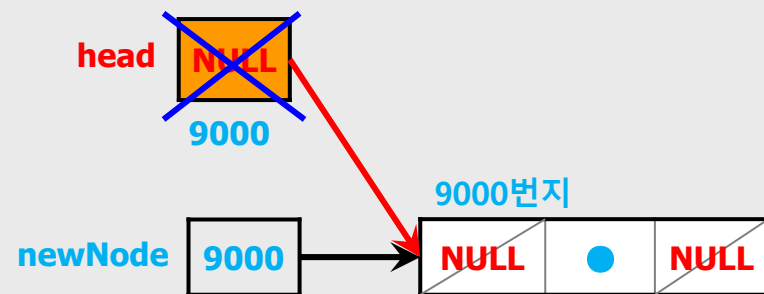
이중 연결 리스트 (2/8)

● 이중 연결 리스트: 삽입 알고리즘

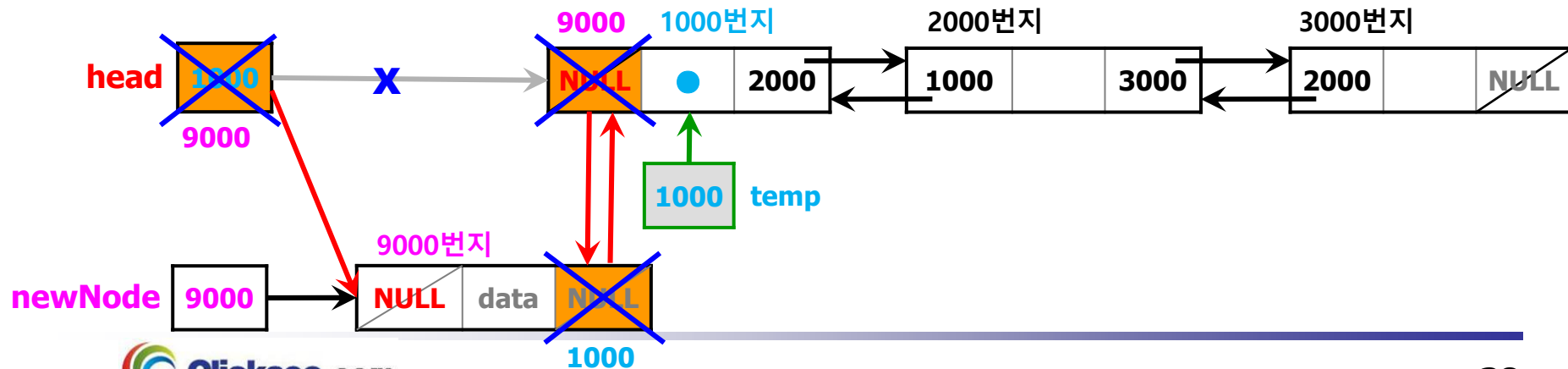
○ 리스트의 첫 번째 노드로 삽입

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

// 빈 리스트일 경우...



// 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;
```

```
}
```

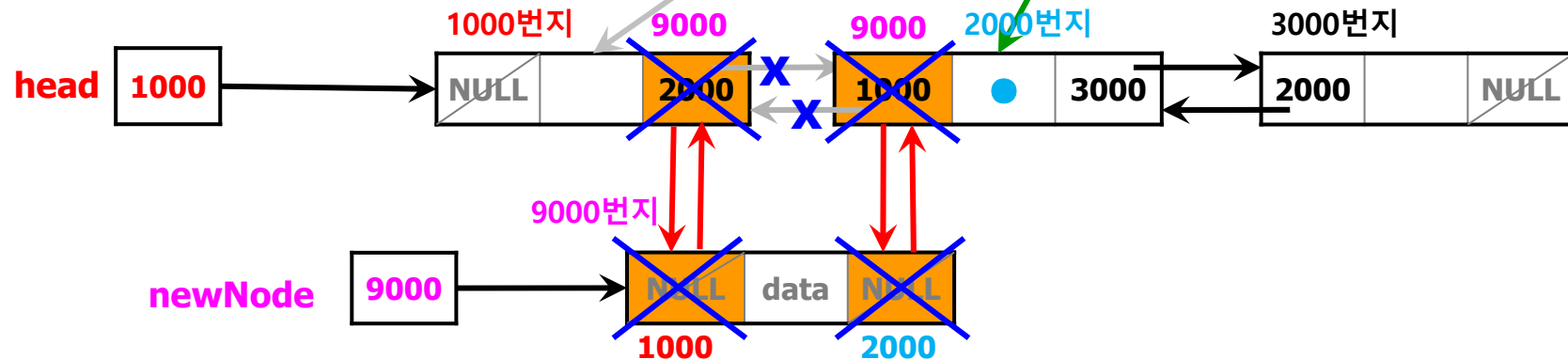
```
end insertMiddleDNode()
```

temp 1000

2000

temp

// 중간 노드로 삽입



이중 연결 리스트 (4/8)

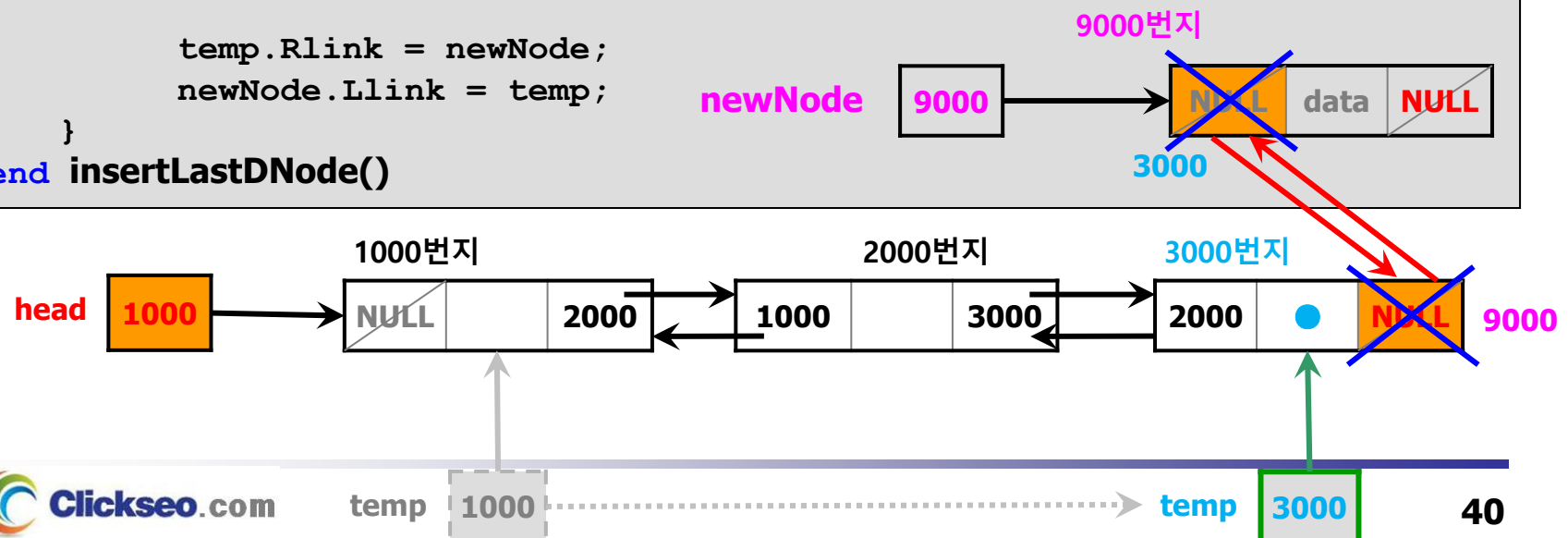
● 이중 연결 리스트: 삽입 알고리즘

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

// 마지막 노드로 삽입



이중 연결 리스트 (5/8)

- 이중 연결 리스트: 삭제 알고리즘

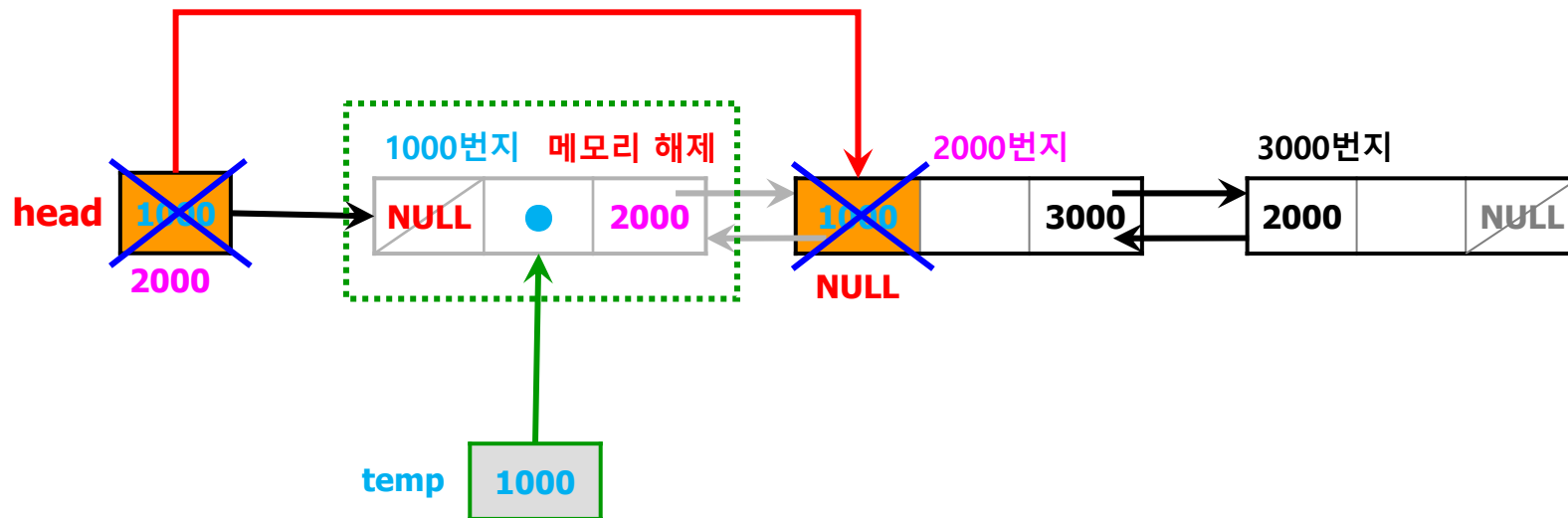
- 리스트에서 조건을 만족하는 노드 삭제 알고리즘

```
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

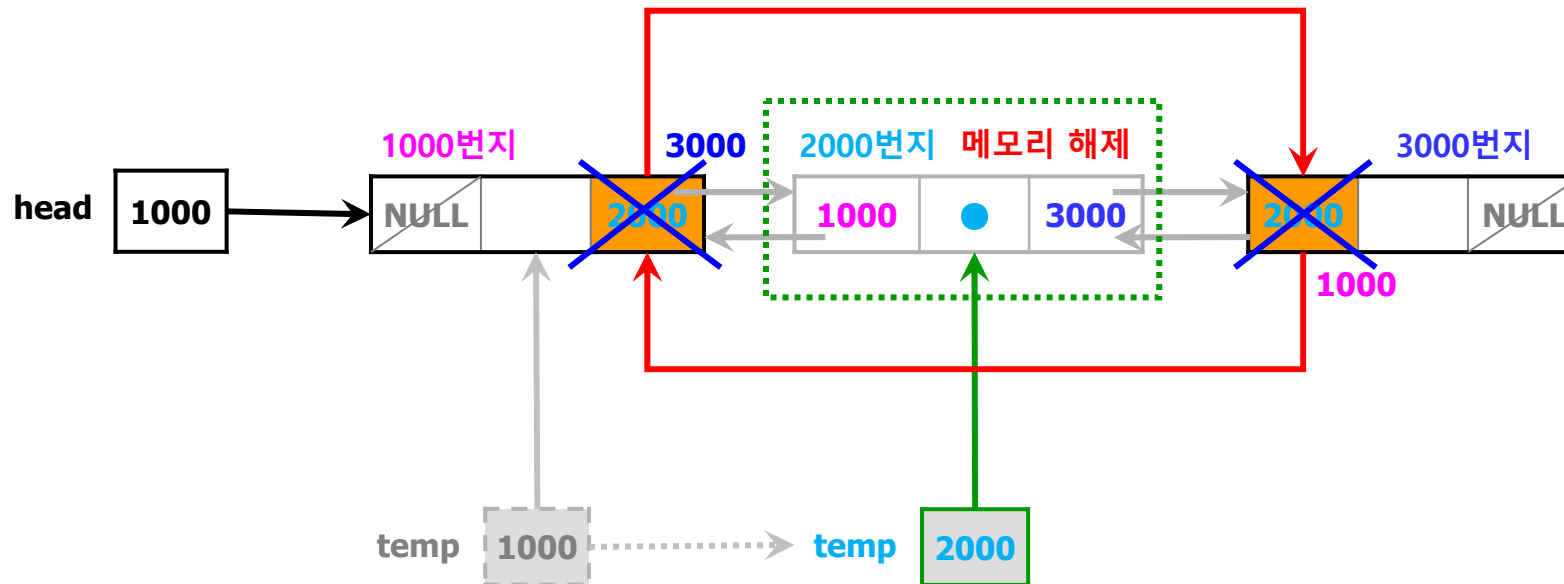


```
// 첫 번째 노드를 삭제  
// deleteFirstDNode();
```

이중 연결 리스트 (7/8)

- 이중 연결 리스트: 삭제 알고리즘

- 리스트의 중간 노드를 삭제



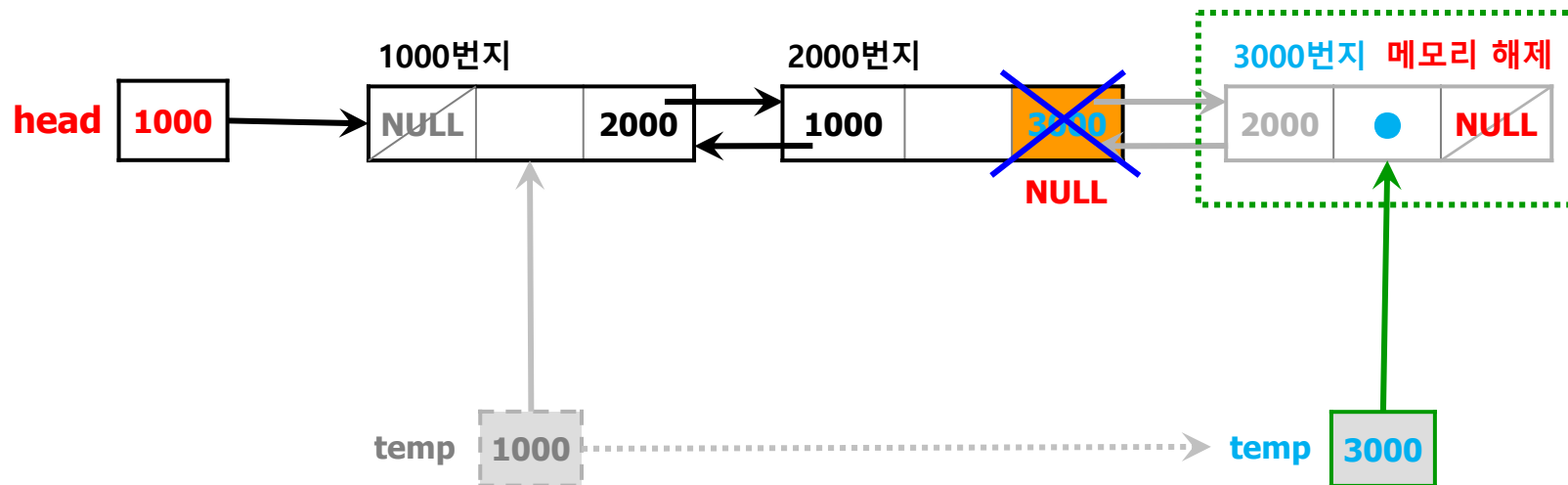
// 중간 노드를 삭제

// deleteMiddleDNode();

이중 연결 리스트 (8/8)

- 이중 연결 리스트: 삭제 알고리즘

- 리스트의 마지막 노드를 삭제



// 마지막 노드를 삭제

// deleteLastDNode();

연결 리스트

이중 연결 리스트
: list 클래스



C++ STL: list 클래스 (1/3)

예제 4-5: 이중 연결 리스트 -- C++ STL(list)

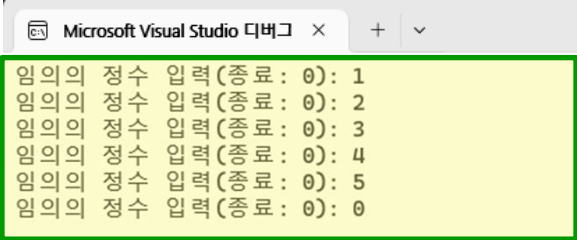
(1/3) | C++

```
#include <iostream>
#include <list>           // list
using namespace std;
int main(void) {
    int          num;
    list<int>     tList;
    // list<int>   tList = { 10, 20, 30, 40, 50 };
    // tList.assign({ 60, 70, 80, 90, 99 });
```

// 새 원소 할당

```
while (true) {
    cout << "임의의 정수 입력(종료: 0): ";
    cin >> num;
    if (num == 0)
        break;
    // 맨 마지막 원소로 추가한다.
    tList.push_back(num);
    // tList.insert(tList.end(), num);

    // 맨 마지막 원소를 삭제한다.
    // tList.pop_back();
    // tList.erase(tList.end());
}
```



Microsoft Visual Studio 디버그

```
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

첫 번째 원소: 1
마지막 원소: 5

### 병합된 데이터 ###
10 20 30 40 50 60 70 80 90 99

C:\Users\click\OneDrive\문서\cppClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```

C++ STL: list 클래스 (2/3)

예제 4-5: 이중 연결 리스트 -- C++ STL(list)

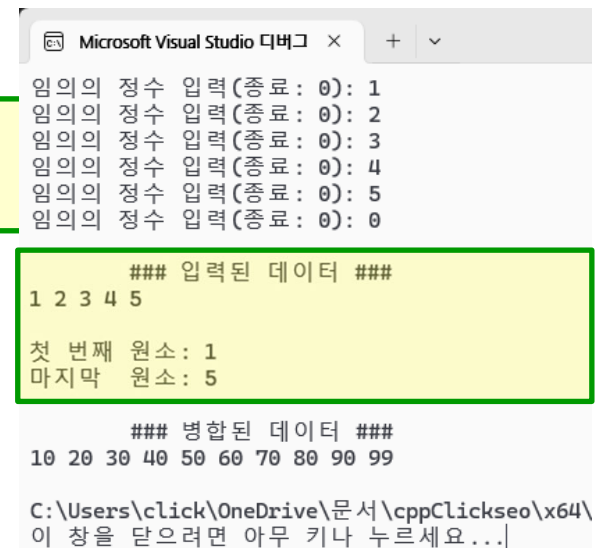
(2/3) | C++

```
// 전체 원소 출력
if (tList.empty()) {
    cout << "\n입력된 데이터가 없습니다!!!" << endl;
}
else {
    cout << "\n\t### 입력된 데이터 ###" << endl;
    for (int num : tList)
        cout << num << ' ';
    cout << endl;
}
cout << endl;
```

```
cout << "첫 번째 원소: " << tList.front() << endl;
cout << "마지막 원소: " << tList.back() << endl;
```

```
// 첫 번째 원소로 추가한다.
// tList.push_front(num);
// tList.insert(tList.begin(), num);
```

```
// 첫 번째 원소를 삭제한다.
// tList.pop_front();
// tList.erase(tList.begin());
```



Microsoft Visual Studio 디버그

```
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

첫 번째 원소: 1
마지막 원소: 5

### 병합된 데이터 ###
10 20 30 40 50 60 70 80 90 99

C:\Users\click\OneDrive\문서\cppClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```

C++ STL: list 클래스 (3/3)

예제 4-5: 이중 연결 리스트 -- C++ STL(list)

(3/3) | C++

```
// 새로운 list: aList, bList
list<int>      aList = { 50, 20, 10, 40, 30 };
list<int>      bList = { 60, 80, 99, 90, 70 };

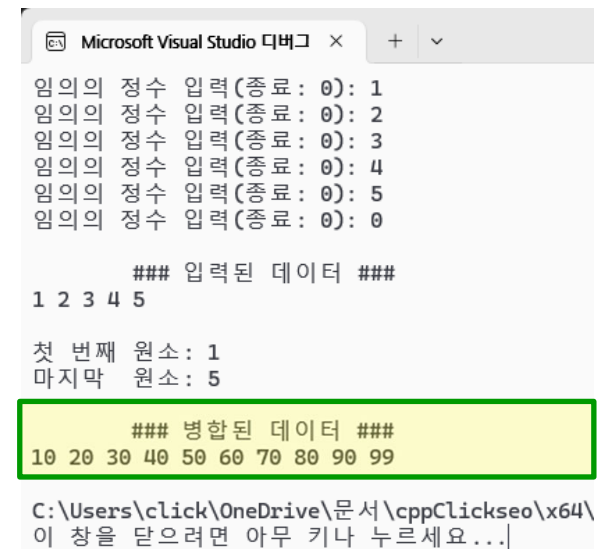
aList.sort();           // 리스트를 정렬한다.
bList.sort();
// aList.reverse();      // 리스트 원소의 순서를 반전 시킨다.

// merge: 서로 다른 list를 병합한다.
// 단, 두 리스트는 사전에 정렬되어 있어야 한다.
aList.merge(bList);

cout << "\n\t### 병합된 데이터 ###" << endl;
for (int num : aList)
    cout << num << ' ';
cout << '\n';

// 모든 원소를 삭제한다.
tList.clear();
aList.clear();
bList.clear();

return 0;
}
```



```
Microsoft Visual Studio 디버그 × + v
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

첫 번째 원소: 1
마지막 원소: 5

      ### 병합된 데이터 ###
10 20 30 40 50 60 70 80 90 99

C:\Users\click\OneDrive\문서\cppClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```


연결 리스트

이중 연결 리스트: Python, C/C++



이중 연결 리스트: Python, C/C++ (1/3)

● 이중 연결 리스트: Python

```
# 노드: Dnode(data, link, Rlink)
```

```
class DNode:
```

```
    def __init__(self, data):
```

```
        self._data = data
```

```
        self._Llink = None
```

```
        self._Rlink = None
```

```
    def getData(self): return self._data
```

```
    def getLink(self): return self._Llink
```

```
    def getRlink(self): return self._Rlink
```

```
    def setData(self, data): self._data = data
```

```
    def setLink(self, Llink): self._Llink = Llink
```

```
    def setRlink(self, Rlink): self._Rlink = Rlink
```

```
# 이중 연결 리스트: DLinkedList
```

```
class DLinkedList:
```

```
    def __init__(self):
```

```
        self._head = None
```

```
        # self._tail = None
```

```
        # self._count = 0
```

```
    def __del__(self):
```

```
    def isEmpty(self) -> bool:
```

```
    def countNode(self) -> int:
```

```
    def frontNode(self) -> DNode:
```

```
    def rearNode(self) -> DNode:
```

```
    def addRear(self, num) -> None:
```

```
    def removeFront(self) -> None:
```

```
    def printLinkedList(self) -> None:
```

```
    def printRevLinkedList(self) -> None:
```

```
        # 생성자
```

```
        # 첫 번째 노드
```

```
        # 맨 마지막 노드
```

```
        # 노드의 총 개수
```

```
        # 소멸자: 전체 노드 삭제
```

```
        # 빈 리스트 여부 판단
```

```
        # 탐색: 노드의 총 개수
```

```
        # 탐색: 첫 번째 노드(head)
```

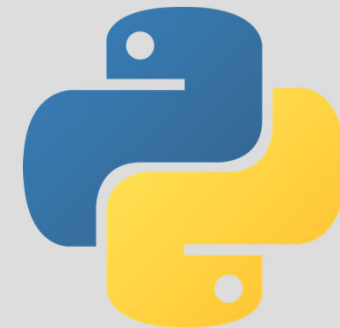
```
        # 탐색: 맨 마지막 노드
```

```
        # 삽입: 맨 마지막 노드
```

```
        # 삭제: 첫 번째 노드(head)
```

```
        # 출력(순방향): 리스트의 전체 원소(노드) 출력
```

```
        # 출력(역방향): 리스트의 전체 원소(노드) 출력
```



이중 연결 리스트: Python, C/C++ (2/3)

● 이중 연결 리스트: C++

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

```
#ifndef _DLinkedList_H_
#define _DLinkedList_H_
```

```
// 이중 연결 리스트: DLinkedList(head)
```

```
class DLinkedList {
public:
```

```
    DLinkedList(void);
```

```
    ~DLinkedList(void);
```

```
    bool isEmpty(void) const;
```

```
    int countNode(void) const;
```

```
    DNode *frontNode(void) const;
```

```
    DNode *rearNode(void) const;
```

```
    void addRear(const int &e);
```

```
    void removeFront(void);
```

```
    void printLinkedList(void) const;
```

```
    void printRevLinkedList(void) const;
```

```
private:
```

```
    DNode *head_; // 첫 번째 노드
```

```
    // DNode *tail_; // 맨 마지막 노드
```

```
    // int count_; // 노드의 총 개수
```

```
};
```

```
#endif
```

```
// 파일명: ListNode.h
// #pragma once
```

```
#ifndef _DNode_H_
#define _DNode_H_
```

```
// 노드: DNode(data, Link, Rlink)
```

```
class DNode {
public:
```

```
    DNode(const int &data);
```

```
    int getData(void) const;
```

```
private:
```

```
    int data_;
```

```
    DNode *Link_;
```

```
    DNode *Rlink_;
```

```
friend class DLinkedList;
```

```
};
```

```
#endif
```

```
// 생성자
```

```
// 소멸자: 전체 노드 삭제
```

```
// 빈 리스트 여부 판단
```

```
// 탐색: 노드의 총 개수(count_)
```

```
// 탐색: 첫 번째 노드(head_)
```

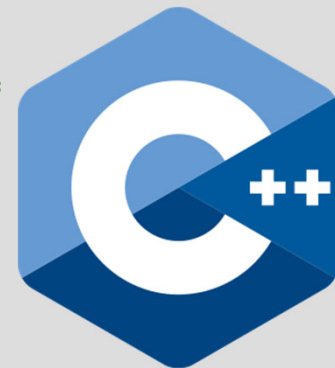
```
// 탐색: 맨 마지막 노드(tail_)
```

```
// 삽입: 맨 마지막 노드(tail_)
```

```
// 삭제: 첫 번째 노드(head_)
```

```
// 리스트의 전체 원소(노드) 출력: 순방향
```

```
// 리스트의 전체 원소(노드) 출력: 역방향
```



이중 연결 리스트: Python, C/C++ (3/3)

● 이중 연결 리스트: C

```
// 파일명: DLinkedList(head).h
// #pragma once
#include "LinkedList.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: LinkedList.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);
```

THE
C
PROGRAMMING
LANGUAGE

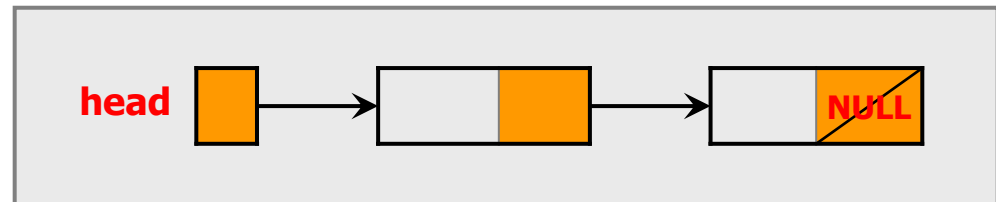
연결 리스트 구현



- 선형 리스트

백문이불여일타(百聞而不如一打)

- 연결 리스트



- 연결 리스트 구현

- 단순 연결 리스트

- 원형 연결 리스트

- 이중 연결 리스트



head



연결 리스트 구현

단순 연결 리스트: Python



단순 연결 리스트: Python (1/9)

● 단순 연결 리스트: Python

```
// 노드: SNode(data, link)
class SNode:
    def __init__(self, data):
        self._data = data
        self._link = None
```

```
    def getData(self): return self._data
    def getLink(self): return self._link
    def setData(self, data): self._data = data
    def setLink(self, link): self._link = link
```

```
# 단순 연결 리스트: SLinkedList
```

```
class SLinkedList:
    def __init__(self):
        self._head = None
        # self._tail = None
        # self._count = 0
```

```
    def __del__(self):
    def isEmpty(self) -> bool:
    def countNode(self) -> int:
    def frontNode(self) -> SNode:
    def rearNode(self) -> SNode:
    def addRear(self, num) -> None:
    def removeFront(self) -> None:
    def printLinkedList(self) -> None:
```

```
# 생성자
# 첫 번째 노드
# 맨 마지막 노드
# 노드의 총 개수
# 소멸자: 전체 노드 삭제
# 빈 리스트 여부 판단
# 탐색: 노드의 총 개수(count)
# 탐색: 첫 번째 노드
# 탐색: 맨 마지막 노드
# 삽입: 맨 마지막 노드
# 삭제: 첫 번째 노드
# 리스트의 전체 노드 출력
```



단순 연결 리스트: Python (2/9)

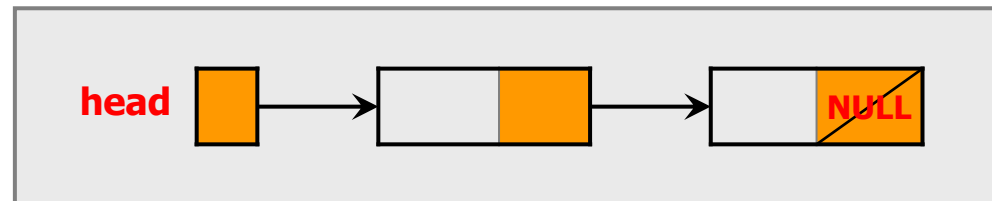
예제 4-4: 단순 연결 리스트

SLinkedList(head).py

(1/8)

```
# 노드: SNode(data, link)
class SNode:
    def __init__(self, data):
        self.__data = data
        self.__link = None
```

```
def getData(self):          return self.__data
def getLink(self):          return self.__link
def setData(self, data):    self.__data = data
def setLink(self, link):    self.__link = link
```



단순 연결 리스트: Python (3/9)

예제 4-4: 단순 연결 리스트

SLinkedList(head).py

(2/8)

```
# 단순 연결 리스트: SLinkedList(head)
```

```
class SLinkedList:
```

```
    def __init__(self):
```

```
        self.__head = None
```

```
        # self.__tail = None
```

```
        # self.__count = 0
```

```
    # 생성자
```

```
    # 첫 번째 노드
```

```
    # 맨 마지막 노드
```

```
    # 노드의 총 개수
```

```
# 소멸자: 전체 노드 삭제
```

```
def __del__(self):
```

```
    # while not self.isEmpty():
```

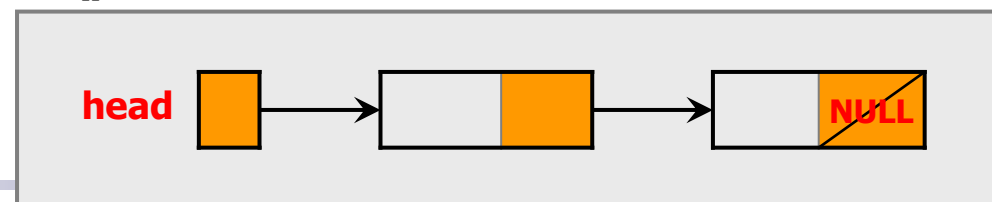
```
    #     self.removeFront()
```

```
while not self.isEmpty():
```

```
    old = self.__head
```

```
    self.__head = old.getLink()
```

```
del old
```



단순 연결 리스트: Python (4/9)

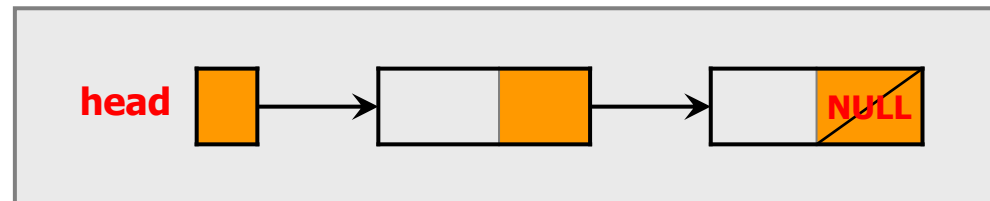
예제 4-4: 단순 연결 리스트

SLinkedList(head).py

(3/8)

```
# 빈 리스트 여부 판단(head)
def isEmpty(self) -> bool:
    return self.__head == None
```

```
# 탐색: 노드의 총 개수
def countNode(self) -> int:
    count = 0
    rNode = self.__head
    while rNode:
        count += 1
        rNode = rNode.getLink()
    return count
```



단순 연결 리스트: Python (5/9)

예제 4-4: 단순 연결 리스트

SLinkedList(head).py

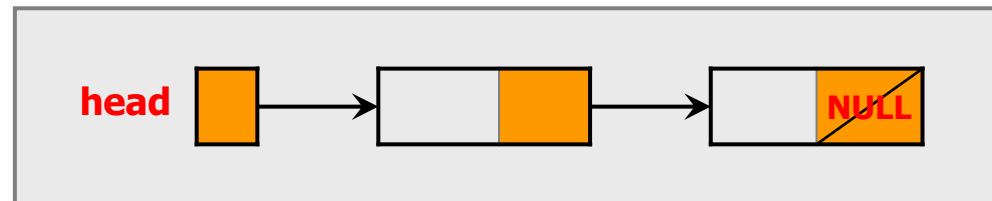
(4/8)

탐색: 첫 번째 노드(head)

```
def frontNode(self) -> SNode:  
    return self.__head
```

탐색: 맨 마지막 노드

```
def rearNode(self) -> SNode:  
    if self.isEmpty():  
        return None  
    rNode = self.__head  
    while rNode.getLink():  
        rNode = rNode.getLink()  
    return rNode
```



단순 연결 리스트: Python (6/9)

예제 4-4: 단순 연결 리스트

SLinkedList(head).py

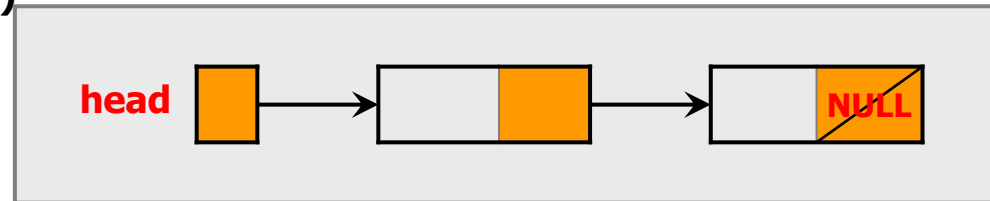
(5/8)

```
# 삽입: 맨 마지막 노드
def addRear(self, num): -> None:
    newNode = SNode(num)
    if self.isEmpty():
        self.__head = newNode
    else:
        rNode = self.rearNode()
        rNode.setLink(newNode)
```

```
# 삭제: 첫 번째 노드(head)
def removeFront(self) -> None:
    if self.isEmpty():
        return
```

```
    old = self.__head
    self.__head = old.getLink()
```

```
    del old
```



단순 연결 리스트: Python (7/9)

예제 4-4: 단순 연결 리스트

SLinkedList(head).py

(6/8)

리스트의 전체 노드 출력

```
def printLinkedList(self): -> None:
```

```
    if self.isEmpty():
```

```
        print('입력된 데이터가 없습니다!!!')
```

```
        return
```

```
    print('\n### 입력된 데이터 ###')
```

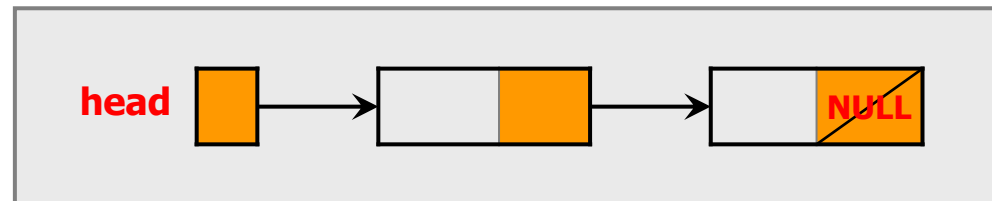
```
    tNode = self.__head
```

```
    while temp :
```

```
        print(f'{tNode.getData()} ->> ', end=' ')
```

```
        tNode = temp.getLink()
```

```
    print(' NULL\n')
```



단순 연결 리스트: Python (8/9)

예제 4-4: 단순 연결 리스트

SLinkedList(head).py

(7/8)

```
if __name__ == '__main__':  
    # head = None  
    sList = SLinkedList()  
  
    while (True):  
        num = int(input('임의의 정수 입력 (종료: 0): '))  
        if num == 0:  
            break  
  
        # 맨 마지막 노드로 삽입  
        sList.addRear(num)  
  
        # 전체 원소 출력  
        # sList.printLinkedList()
```

IDLE Shell 3.11.2

File Edit Shell Debug Options Window Help

Python 3.11.2 (tags/v3.11.2:878ead1, Feb 7 2023)
Type "help", "copyright", "credits" or "license"

>>>

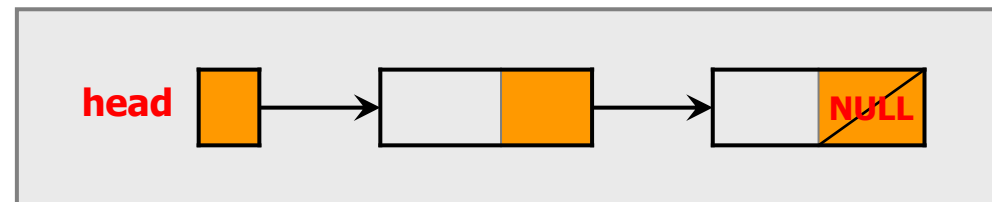
```
= RESTART: C:\Users\Clickseo\OneDrive\Clickseo\W  
임의의 정수 입력 (종료: 0): 1  
임의의 정수 입력 (종료: 0): 2  
임의의 정수 입력 (종료: 0): 3  
임의의 정수 입력 (종료: 0): 4  
임의의 정수 입력 (종료: 0): 5  
임의의 정수 입력 (종료: 0): 0
```

입력된 데이터

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

```
노드의 총 개수: 5  
첫번째 노드의 데이터: 1  
마지막 노드의 데이터: 5
```

>>>



단순 연결 리스트: Python (9/9)

예제 4-4: 단순 연결 리스트

SLinkedList(head).py

(8/8)

리스트 전체 노드의 데이터 출력

```
sList.printLinkedList()
```

노드의 총 개수

```
print(f'노드의 총 개수: {sList.countNode()}')
```

빈 리스트 여부 판단

```
if sList.isEmpty():
```

```
    print('입력된 데이터가 없습니다!!!')
```

```
else:
```

첫번째 노드와 마지막 노드의 데이터

```
print(f'첫번째 노드의 데이터: {sList.frontNode().getData()}')
```

```
print(f'마지막 노드의 데이터: {sList.rearNode().getData()}')
```

```
print()
```

```
# del sList
```

```
# sList.__del__()
```

IDLE Shell 3.11.2

File Edit Shell Debug Options Window Help

Python 3.11.2 (tags/v3.11.2:878ead1, Feb. 7 2021)
Type "help", "copyright", "credits" or "license"

>>>

= RESTART: C:\Users\Clickseo\OneDrive\Clickseo\

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

첫번째 노드의 데이터: 1

마지막 노드의 데이터: 5

>>>

head





연결 리스트 구현

단순 연결 리스트: C++



단순 연결 리스트: C++ (1/11)

● 단순 연결 리스트: C++

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

#ifndef __SLinkedList_H__
#define __SLinkedList_H__

// 단순 연결 리스트: SLinkedList
class SLinkedList {
public:
    SLinkedList(void);
    ~SLinkedList(void);
    bool isEmpty(void) const;
    int countNode(void) const;
    SNode *frontNode(void) const;
    SNode *rearNode(void) const;
    void addRear(const int &e);
    void removeFront(void);
    void printLinkedList(void) const;
private:
    SNode *head_; // 첫 번째 노드
    // SNode *tail_; // 맨 마지막 노드
    // int count_; // 노드의 총 개수
};

#endif
```

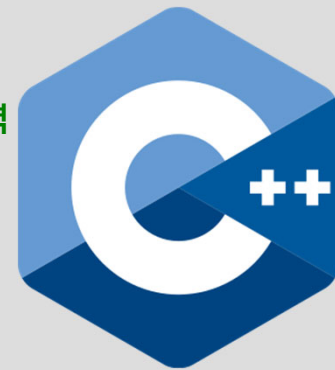
```
// 파일명: LinkedList.h
// #pragma once

#ifndef __SNode_H__
#define __SNode_H__

// 노드: SNode(data, link)
class SNode {
public:
    SNode(const int& data);
    int getData(void) const;
private:
    int data_;
    SNode *link_;
    friend class SLinkedList;
};

#endif
```

```
// 생성자
// 소멸자: 전체 노드 삭제
// 빈 리스트 여부 판단
// 탐색: 노드의 총 개수(count_)
// 탐색: 첫 번째 노드(head_)
// 탐색: 맨 마지막 노드(tail_)
// 삽입: 맨 마지막 노드(tail_)
// 삭제: 첫 번째 노드(head_)
// 리스트의 전체 원소(노드) 출력
```



단순 연결 리스트: C++ (2/11)

예제 4-3: 단순 연결 리스트

SLinkedList(demo).cpp

(1/2)

```
#include <iostream>
#include "SLinkedList(head).h"          // SLinkedList >> head
// #include "SLinkedList(tail).h"      // SLinkedList >> head, count, tail
// #include "LinkedList.h"             // SNode
using namespace std;
```

```
int main(void)
{
    int          num;
    SLinkedList  sList = SLinkedList();    // head_ = nullptr;

    while (true) {
        cout << "임의의 정수 입력 (종료: 0): ";
        cin >> num;
        if (num == 0)
            break;
        // 맨 마지막 노드로 삽입
        sList.addRear(num);
    }
```

```
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++ (3/11)

예제 4-3: 단순 연결 리스트

SLinkedList(demo).cpp

(2/2)

```
// 리스트 전체 노드의 데이터 출력
sList.printLinkedList();

// 노드의 총 개수
cout << "노드의 총 개수: " << sList.countNode() << endl;

// 빈 리스트 여부 판단
if (sList.isEmpty()) {
    cout << "입력된 데이터가 없습니다!!!\n" << endl;
}
else {
    // 첫번째 노드와 마지막 노드의 데이터
    cout << "첫번째 노드의 데이터: " << sList.frontNode()->getData() << endl;
    cout << "마지막 노드의 데이터: " << sList.rearNode()->getData() << endl;
}
cout << endl;

// sList.~SLinkedList();
return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
입력된 데이터: 1
입력된 데이터: 2
입력된 데이터: 3
입력된 데이터: 4
입력된 데이터: 5
입력된 데이터: 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++ (4/11)

예제 4-3: 단순 연결 리스트

ListNode.h

```
// 단순 연결 리스트
```

```
// #pragma once
```

```
#ifndef __SNode_H__
```

```
#define __SNode_H__
```

```
// 노드: SNode(data, link)
```

```
class SNode {
```

```
public:
```

```
    SNode(const int &data);
```

```
    int    getData(void) const;
```

```
private:
```

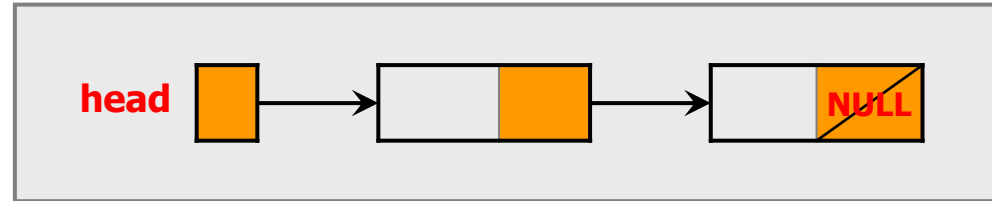
```
    int    data_;
```

```
    SNode  *link_;
```

```
    friend class    SLinkedList;
```

```
};
```

```
#endif
```



단순 연결 리스트: C++ (5/11)

예제 4-3: 단순 연결 리스트

LinkedList.cpp

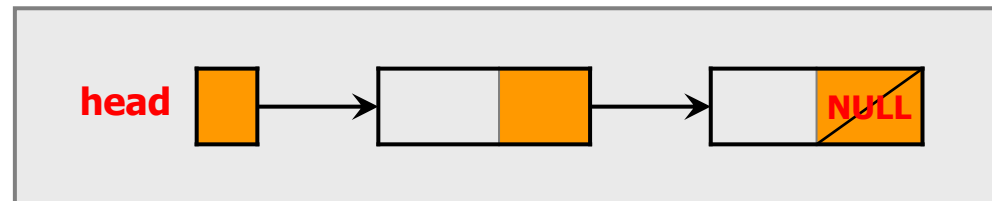
```
#include <iostream>

#include "LinkedList.h" // SNode

using namespace std;

// 생성자: 새로운 노드(SNode: data, link) 생성
SNode::SNode(const int &data):
    data_(data), link_(nullptr) {}

// 노드의 데이터 반환
int SNode::getData(void) const {
    return data_;
}
```



단순 연결 리스트: C++ (6/11)

예제 4-3: 단순 연결 리스트

SLinkedList(head).h

```
// #pragma once
#include "LinkedList.h"      // SNode

#ifndef __SLinkedList_H__
#define __SLinkedList_H__

// 단순 연결 리스트: SLinkedList(head, tail, count)
class SLinkedList {
public:
    SLinkedList(void);           // 생성자
    ~SLinkedList(void);          // 소멸자: 전체 노드 삭제
    bool isEmpty(void) const;    // 빈 리스트 여부 판단
    int countNode(void) const;   // 탐색: 노드의 총 개수 (count_)
    SNode *frontNode(void) const; // 탐색: 첫 번째 노드 (head_)
    SNode *rearNode(void) const;  // 탐색: 맨 마지막 노드 (tail_)
    void addRear(const int &e);    // 삽입: 맨 마지막 노드 (tail_)
    void removeFront(void);        // 삭제: 첫 번째 노드 (head_)
    void printLinkedList(void) const; // 리스트의 전체 원소 (노드) 출력

private:
    SNode *head_;               // 첫 번째 노드
    // SNode *tail_;           // 맨 마지막 노드
    // int count_;              // 노드의 총 개수
};

#endif
```

단순 연결 리스트: C++ (7/11)

예제 4-3: 단순 연결 리스트

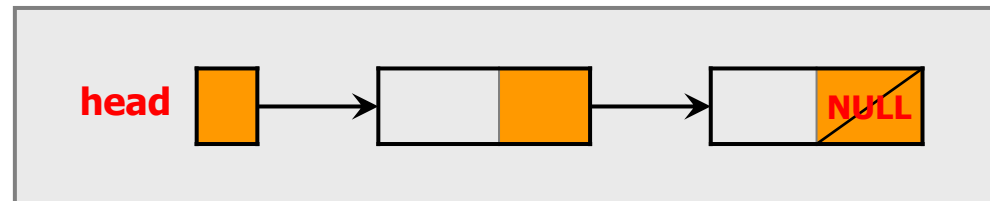
SLinkedList(head).cpp

(1/5)

```
#include <iostream>
#include "SLinkedList(head).h"           // SLinkedList >> head
// #include "SLinkedList(tail).h"       // SLinkedList >> head, count, tail
// #include "LinkedList.h"              // SNode
using namespace std;

// 생성자: 빈 리스트 생성(head)
SLinkedList::SLinkedList(void)
    : head_(nullptr) { }

// 소멸자: 전체 노드 삭제
SLinkedList::~~SLinkedList(void) {
    // while (!isEmpty())
    //     removeFront();
    SNode *tNode = head_;
    while (tNode) {
        head_ = tNode->link_;
        delete tNode;
        tNode = head_;
    }
}
```



단순 연결 리스트: C++ (8/11)

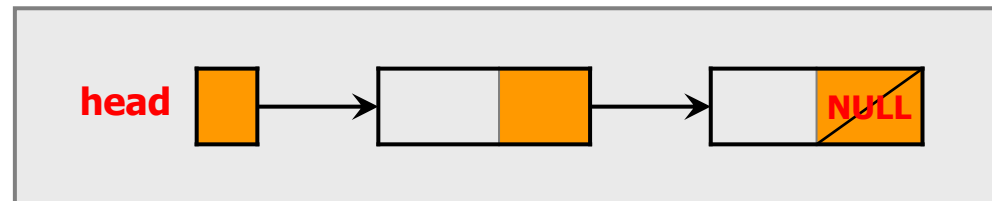
예제 4-3: 단순 연결 리스트

SLinkedList(head).cpp

(2/5)

```
// 빈 리스트 여부 판단(head)
bool SLinkedList::isEmpty(void) const {
    return head_ == nullptr;
}

// 탐색: 노드의 총 개수
int SLinkedList::countNode(void) const {
    int count = 0;
    SNode *rNode = head_;
    while (rNode) {
        ++count;
        rNode = rNode->link_;
    }
    return count;
}
```



단순 연결 리스트: C++ (9/11)

예제 4-3: 단순 연결 리스트

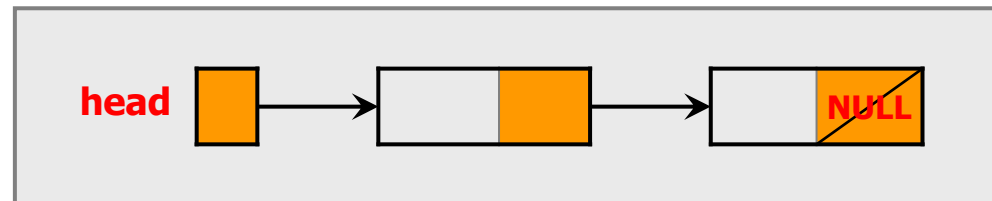
SLinkedList(head).cpp

(3/5)

```
// 탐색: 첫 번째 노드(head_)
SNode *SLinkedList::frontNode(void) const {
    return head_;
}

// 탐색: 맨 마지막 노드
SNode *SLinkedList::rearNode(void) const {
    if (isEmpty())
        return nullptr;

    SNode *rNode = head_;
    while (rNode->link_)
        rNode = rNode->link_;
    return rNode;
}
```



단순 연결 리스트: C++ (10/11)

예제 4-3: 단순 연결 리스트

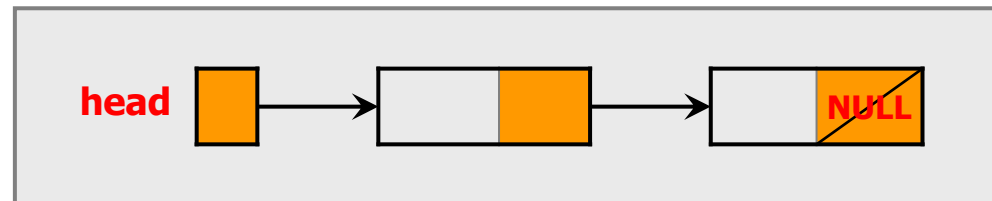
SLinkedList(head).cpp

(4/5)

```
// 삽입: 맨 마지막 노드
void SLinkedList::addRear(const int &e) {
    SNode *newNode = new SNode(e);
    if (isEmpty()) {
        head_ = newNode;
    }
    else {
        SNode *rNode = rearNode();
        rNode->link_ = newNode;
    }
}

// 삭제: 첫 번째 노드(head)
void SLinkedList::removeFront(void) {
    if (isEmpty())
        return;

    SNode *old = head_;
    head_ = old->link_;
    delete old;
}
```



단순 연결 리스트: C++ (11/11)

예제 4-3: 단순 연결 리스트

SLinkedList(head).cpp

(5/5)

// 출력: 리스트의 전체 노드의 데이터

```
void SLinkedList::printLinkedList(void) const {  
    if (isEmpty()) {  
        cout << "\n입력된 데이터가 없습니다!!!" << endl;  
        return;  
    }  
  
    cout << "\n\t### 입력된 데이터 ###" << endl;  
    SNode *tNode = head_;  
    while (tNode) {  
        cout.width(3);  
        cout << tNode->data_ << " ->";  
        tNode = tNode->link_;  
    }  
    cout << " NULL\n" << endl;  
}
```



head



연결 리스트 구현

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 "ListNode.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: ListNode.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);
```

THE
C
PROGRAMMING
LANGUAGE

단순 연결 리스트: C (2/12)

예제 4-2: 단순 연결 리스트

SLinkedList(demo).c (1/2)

```
#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 "LinkedList.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", countNode(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", frontNode(sList)->data );
    printf("마지막 노드의 데이터 : %d\n", rearNode(sList)->data );
}

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

Microsoft Visual Studio 디버그 콘솔

```
입력된 데이터: 1
입력된 데이터: 2
입력된 데이터: 3
입력된 데이터: 4
입력된 데이터: 5
입력된 데이터: 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

LinkedList.h

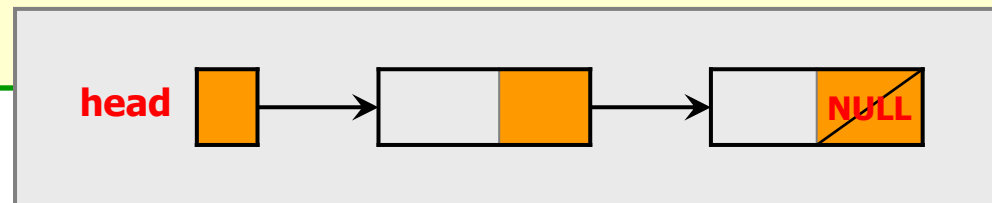
```
// #pragma once
typedef int    element;

// 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);
```



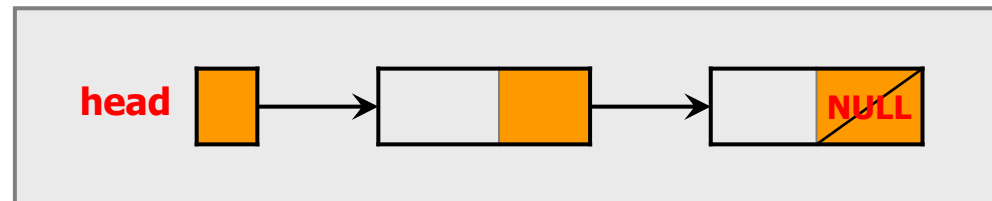
단순 연결 리스트: C (5/12)

예제 4-2: 단순 연결 리스트

LinkedList.c

```
#include <stdio.h>
#include <stdlib.h>           // malloc, free
#include "LinkedList.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 "LinkedList.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);
```

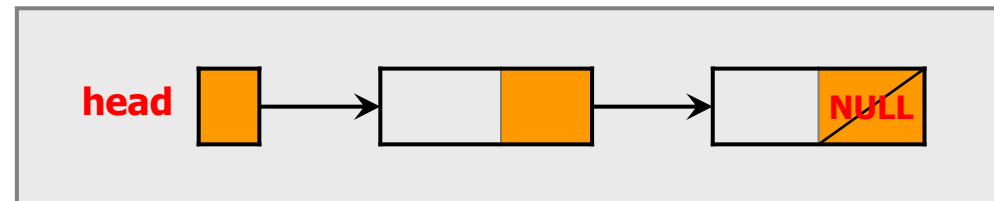
단순 연결 리스트: C (7/12)

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 "LinkedList.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;
}
```

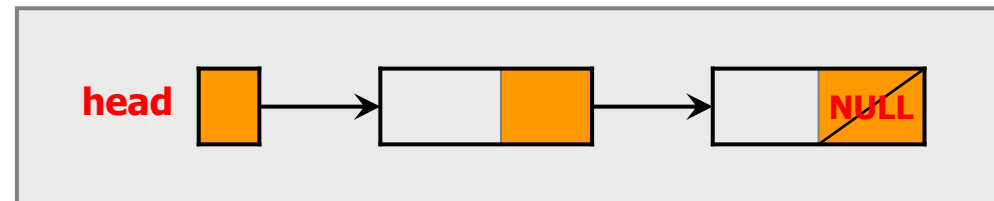


단순 연결 리스트: C (8/12)

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;
}
```



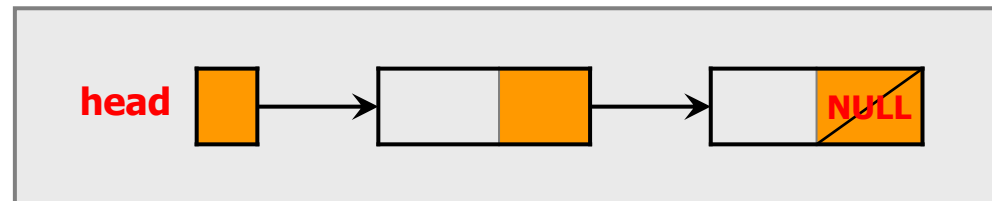
단순 연결 리스트: C (9/12)

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)

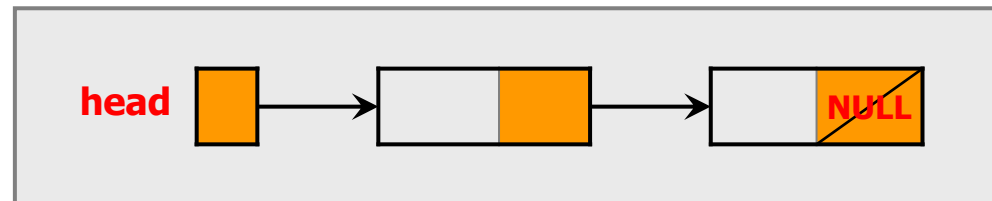
예제 4-2: 단순 연결 리스트

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

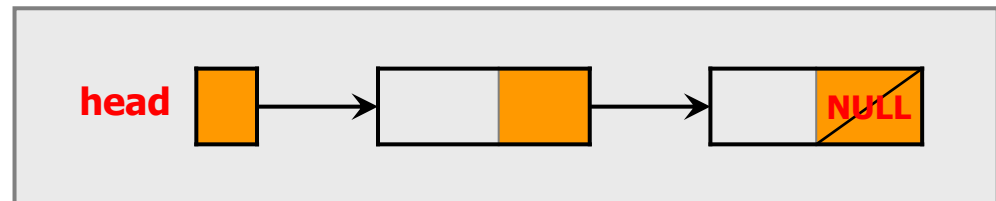
SLinkedList(head).c (5/6)

// 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);  
}
```



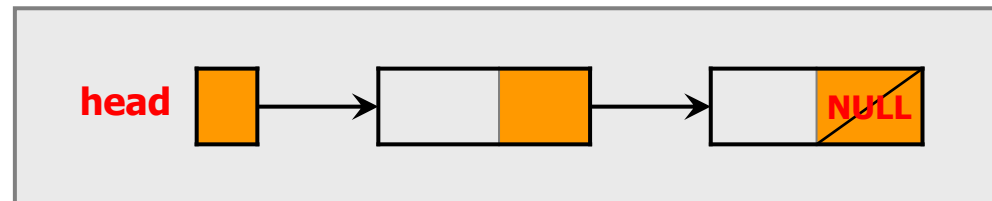
단순 연결 리스트: C (12/12)

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("%03d ->>", tNode->data);
        tNode = tNode->link;
    }
    printf(" NULL\n");
}
```



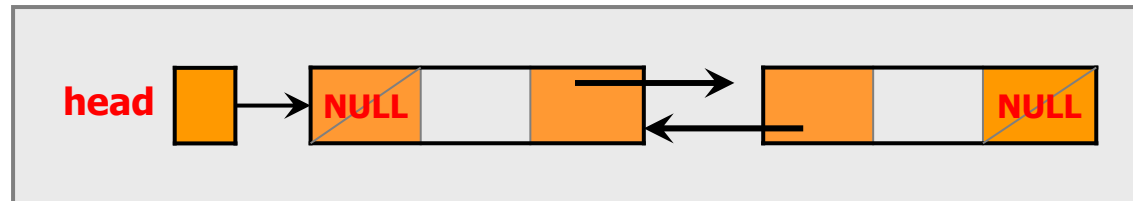
연결 리스트 구현



- 선형 리스트

백문이불여일타(百聞而不如一打)

- 연결 리스트



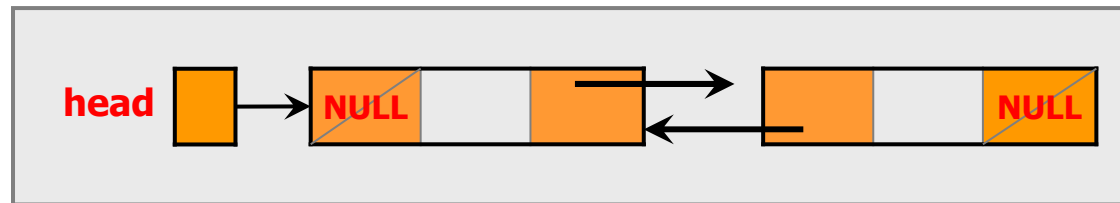
- 연결 리스트 구현

- 단순 연결 리스트

- 원형 연결 리스트

- 이중 연결 리스트





연결 리스트 구현

이중 연결 리스트: Python



이중 연결 리스트: Python (1/10)

● 이중 연결 리스트: Python

```
# 노드: Dnode(data, link, Rlink)
```

```
class DNode:
```

```
    def __init__(self, data):
```

```
        self._data = data
```

```
        self._Llink = None
```

```
        self._Rlink = None
```

```
    def getData(self): return self._data
```

```
    def getLink(self): return self._Llink
```

```
    def getRlink(self): return self._Rlink
```

```
    def setData(self, data): self._data = data
```

```
    def setLink(self, Llink): self._Llink = Llink
```

```
    def setRlink(self, Rlink): self._Rlink = Rlink
```

```
# 이중 연결 리스트: DLinkedList
```

```
class DLinkedList:
```

```
    def __init__(self):
```

```
        self._head = None
```

```
        # self._tail = None
```

```
        # self._count = 0
```

```
        # 생성자
```

```
        # 첫 번째 노드
```

```
        # 맨 마지막 노드
```

```
        # 노드의 총 개수
```

```
    def __del__(self):
```

```
    def isEmpty(self) -> bool:
```

```
    def countNode(self) -> int:
```

```
    def frontNode(self) -> DNode:
```

```
    def rearNode(self) -> DNode:
```

```
    def addRear(self, num) -> None:
```

```
    def removeFront(self) -> None:
```

```
    def printLinkedList(self) -> None:
```

```
    def printRevLinkedList(self) -> None:
```

```
        # 소멸자: 전체 노드 삭제
```

```
        # 빈 리스트 여부 판단
```

```
        # 탐색: 노드의 총 개수
```

```
        # 탐색: 첫 번째 노드(head)
```

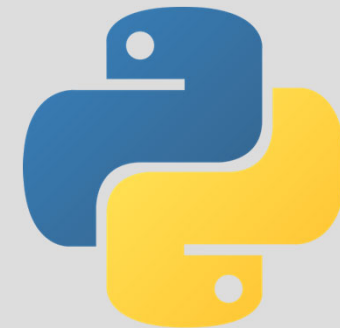
```
        # 탐색: 맨 마지막 노드
```

```
        # 삽입: 맨 마지막 노드
```

```
        # 삭제: 첫 번째 노드(head)
```

```
        # 출력(순방향): 리스트의 전체 원소(노드) 출력
```

```
        # 출력(역방향): 리스트의 전체 원소(노드) 출력
```



이중 연결 리스트: Python (2/10)

예제 4-8: 이중 연결 리스트

DLinkedList(head).py

(1/9)

```
# 노드: DNode(data, Llink, Rlink)
class DNode:
    def __init__(self, data):
        self.__data = data
        self.__Llink = None
        self.__Rlink = None

    def getData(self):
        return self.__data
    def getLlink(self):
        return self.__Llink
    def getRlink(self):
        return self.__Rlink
    def setData(self, data):
        self.__data = data
    def setLlink(self, Llink):
        self.__Llink = Llink
    def setRlink(self, Rlink):
        self.__Rlink = Rlink

# getattr(DNode, '__data')
# getattr(DNode, '__Llink')
# getattr(DNode, '__Rlink')
# setattr(DNode, '__data', 'data')
# setattr(DNode, '__Llink', 'Llink')
# setattr(DNode, '__Rlink', 'Rlink')
```

이중 연결 리스트: Python (3/10)

예제 4-8: 이중 연결 리스트

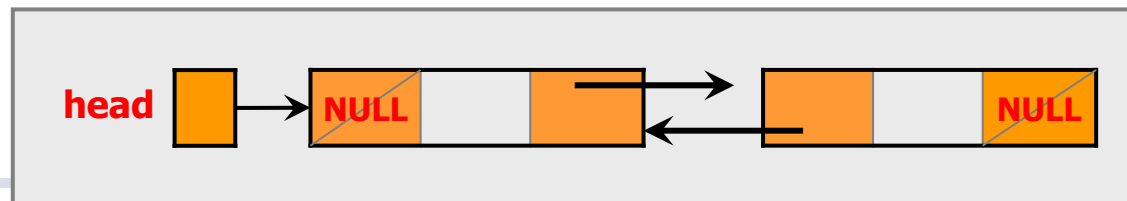
DLinkedList(head).py

(2/9)

```
# 이중 연결 리스트: DLinkedList(head)
class DLinkedList:
    def __init__(self):          # 생성자
        self.__head = None      # 첫 번째 노드
        # self.__tail = None    # 맨 마지막 노드
        # self.__count = 0      # 노드의 총 개수
```

소멸자: 전체 노드 삭제

```
def __del__(self):
    # while not self.isEmpty():
    #     self.removeFront()
    while not self.isEmpty():
        old = self.__head
        self.__head = old.getRlink()
        del old
```



이중 연결 리스트: Python (4/10)

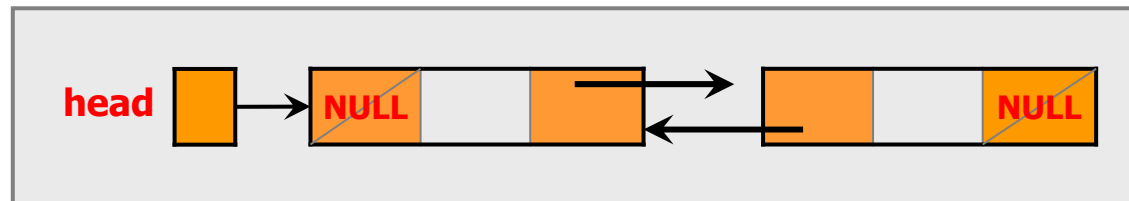
예제 4-8: 이중 연결 리스트

DLinkedList(head).py

(3/9)

```
# 빈 리스트 여부 판단(head)
def isEmpty(self) -> bool:
    return self.__head == None

# 탐색: 노드의 총 개수
def countNode(self) -> int:
    count = 0
    rNode = self.__head
    while rNode:
        count += 1
        rNode = rNode.getRlink()
    return count
```



이중 연결 리스트: Python (5/10)

예제 4-8: 이중 연결 리스트

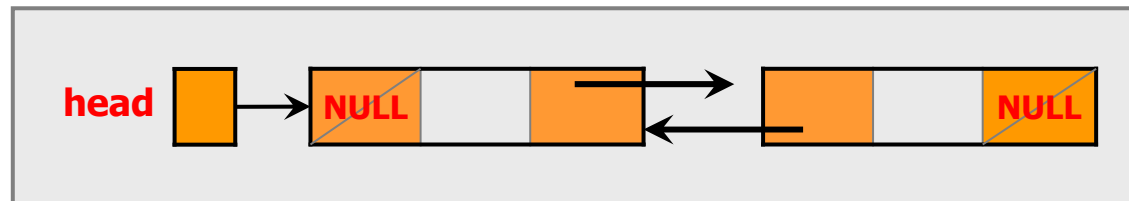
DLinkedList(head).py

(4/9)

```
# 탐색: 첫 번째 노드(head)
def frontNode(self) -> DNode:
    return self.__head

# 탐색: 맨 마지막 노드
def rearNode(self) -> DNode:
    if self.isEmpty():
        return None

    rNode = self.__head
    while rNode.getRlink():
        rNode = rNode.getRlink()
    return rNode
```



이중 연결 리스트: Python (6/10)

예제 4-8: 이중 연결 리스트

DLinkedList(head).py

(5/9)

```
# 삽입: 맨 마지막 노드
def addRear(self, num) -> None:
    newNode = DNode(num)
    if self.isEmpty():
        self.__head = newNode
    else:
        rNode = self.rearNode()
        rNode.setRlink(newNode)
        newNode.setLlink(rNode)

# 삭제: 첫 번째 노드(head)
def removeFront(self) -> None:
    if self.isEmpty():
        return

    old = self.__head
    self.__head = old.getRlink()
    if self.__head:
        self.__head.setLlink(None)

    del old
```

head



이중 연결 리스트: Python (7/10)

예제 4-8: 이중 연결 리스트

DLinkedList(head).py

(6/9)

출력(순방향): 리스트 전체 원소(노드)의 데이터 출력

```
def printLinkedList(self) -> None:
```

```
    if self.isEmpty():
```

```
        print('입력된 데이터가 없습니다!!!')
```

```
        return
```

```
    print(f'\n### 입력된 데이터 (순방향) ###')
```

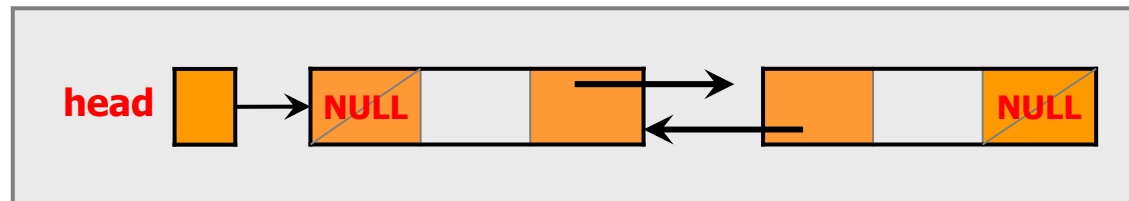
```
    tNode = self.__head
```

```
    while tNode:
```

```
        print(f'{tNode.getData()} ->>', end=' ')
```

```
        tNode = tNode.getRlink()
```

```
    print(' None\n')
```



이중 연결 리스트: Python (8/10)

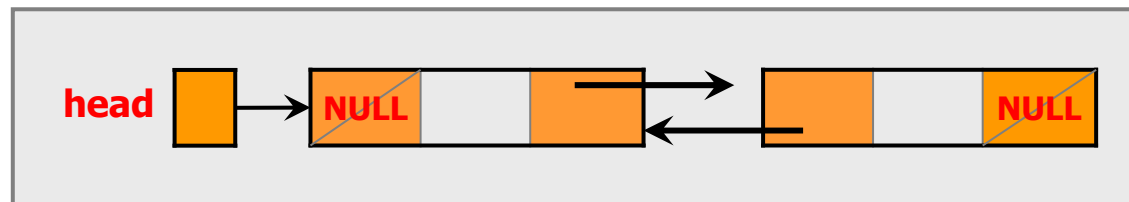
예제 4-8: 이중 연결 리스트

DLinkedList(head).py

(7/9)

```
# 출력(역방향): 리스트의 전체 원소(노드)의 데이터 출력
def printRevLinkedList(self) -> None:
    if self.isEmpty():
        print('입력된 데이터가 없습니다!!!')
        return

    print(f'\n### 입력된 데이터(역방향) ###')
    rNode = self.rearNode()
    while rNode:
        print(f'{rNode.getData()} ->>', end=' ')
        rNode = rNode.getLink()
    print(' None\n')
```



이중 연결 리스트: Python (9/10)

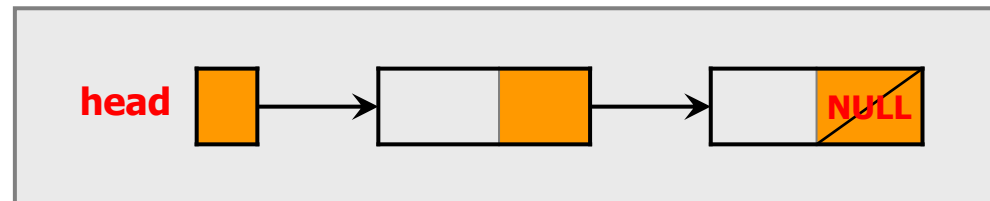
예제 4-4: 단순 연결 리스트

DLinkedList(head).py

(8/9)

```
if __name__ == '__main__':  
    # head = None  
    dList = DLinkedList()  
  
    while (True):  
        num = int(input('임의의 정수 입력 (종료: 0): '))  
        if num == 0:  
            break  
  
        # 맨 마지막 노드로 삽입  
        dList.addRear(num)  
  
    # 전체 원소 출력  
    # dList.printLinkedList()  
    # dList.printRevLinkedList()
```

```
IDLE Shell 3.11.2  
File Edit Shell Debug Options Window Help  
Python 3.11.2 (tags/v3.11.2:878ead1, Feb 7 2022)  
Type "help", "copyright", "credits" or "license"  
>>>  
= RESTART: C:\Users\Clickseo\OneDrive\Clickseo\W  
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 ->> None  
  
### 입력된 데이터 (역방향) ###  
5 ->> 4 ->> 3 ->> 2 ->> 1 ->> None  
  
노드의 총 개수: 5  
첫번째 노드의 데이터: 1  
마지막 노드의 데이터: 5  
>>>
```



이중 연결 리스트: Python (10/10)

예제 4-4: 단순 연결 리스트

DLinkedList(head).py

(9/9)

```
# 전체 원소 출력
dList.printLinkedList()      # 순방향 출력
dList.printRevLinkedList()    # 역방향 출력

# 노드의 총 개수
print(f'노드의 총 개수: {dList.countNode()}')

# 빈 리스트 여부 판단
if dList.isEmpty():
    print('입력된 데이터가 없습니다!!!')
else:
    # 첫번째 노드와 마지막 노드의 데이터
    print(f'첫번째 노드의 데이터: {dList.frontNode().getData()}')
    print(f'마지막 노드의 데이터: {dList.rearNode().getData()}')
print()

# del sList
# sList.__del__()
```

IDLE Shell 3.11.2

File Edit Shell Debug Options Window Help

Python 3.11.2 (tags/v3.11.2:878ead1, Feb 7 2022)
Type "help", "copyright", "credits" or "license"

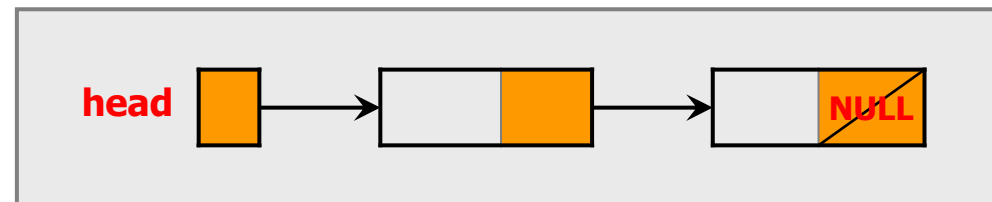
```
>>>
= RESTART: C:\Users\Clickseo\OneDrive\Clickseo\
입력의 정수 입력 (종료: 0): 1
입력의 정수 입력 (종료: 0): 2
입력의 정수 입력 (종료: 0): 3
입력의 정수 입력 (종료: 0): 4
입력의 정수 입력 (종료: 0): 5
입력의 정수 입력 (종료: 0): 0

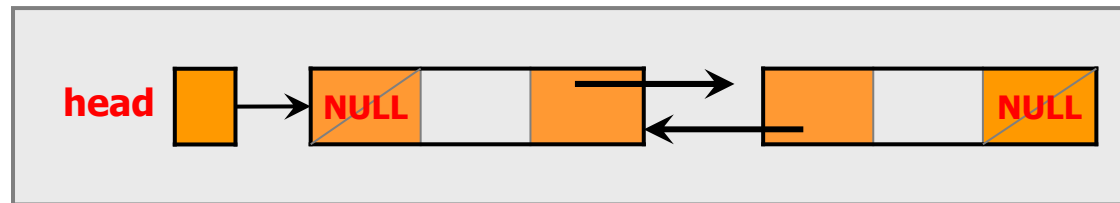
### 입력된 데이터(순방향) ###
1 -> 2 -> 3 -> 4 -> 5 -> None

### 입력된 데이터(역방향) ###
5 -> 4 -> 3 -> 2 -> 1 -> None

노드의 총 개수: 5
첫번째 노드의 데이터: 1
마지막 노드의 데이터: 5

>>>
```





연결 리스트 구현

이중 연결 리스트: C++



이중 연결 리스트: C++ (1/13)

● 이중 연결 리스트: C++

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

```
#ifndef _DLinkedList_H_
#define _DLinkedList_H_
```

```
// 이중 연결 리스트: DLinkedList(head)
```

```
class DLinkedList {
public:
```

```
    DLinkedList(void);
```

```
    ~DLinkedList(void);
```

```
    bool isEmpty(void) const;
```

```
    int countNode(void) const;
```

```
    DNode *frontNode(void) const;
```

```
    DNode *rearNode(void) const;
```

```
    void addRear(const int &e);
```

```
    void removeFront(void);
```

```
    void printLinkedList(void) const;
```

```
    void printRevLinkedList(void) const;
```

```
private:
```

```
    DNode *head_; // 첫 번째 노드
```

```
    // DNode *tail_; // 맨 마지막 노드
```

```
    // int count_; // 노드의 총 개수
```

```
};
```

```
#endif
```

```
// 파일명: ListNode.h
// #pragma once
```

```
#ifndef _DNode_H_
#define _DNode_H_
```

```
// 노드: DNode(data, Link, Rlink)
```

```
class DNode {
public:
```

```
    DNode(const int &data);
```

```
    int getData(void) const;
```

```
private:
```

```
    int data_;
```

```
    DNode *Link_;
```

```
    DNode *Rlink_;
```

```
friend class DLinkedList;
```

```
};
```

```
#endif
```

```
// 생성자
```

```
// 소멸자: 전체 노드 삭제
```

```
// 빈 리스트 여부 판단
```

```
// 탐색: 노드의 총 개수(count_)
```

```
// 탐색: 첫 번째 노드(head_)
```

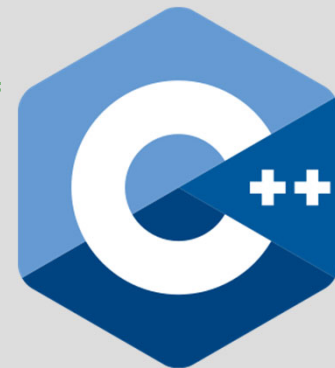
```
// 탐색: 맨 마지막 노드(tail_)
```

```
// 삽입: 맨 마지막 노드(tail_)
```

```
// 삭제: 첫 번째 노드(head_)
```

```
// 리스트의 전체 원소(노드) 출력: 순방향
```

```
// 리스트의 전체 원소(노드) 출력: 역방향
```



이중 연결 리스트: C++ (2/13)

예제 4-7: 이중 연결 리스트

DLinkedList(demo).cpp

(1/2)

```
#include <iostream>
#include "DLinkedList(head).h" // DLinkedList >> head
// #include "DLinkedList.h" // DLinkedList >> head, count, tail
// #include "LinkedList.h" // DNode
using namespace std;
```

```
int main(void)
{
    int num;
    DLinkedList dList = DLinkedList(); // head_ = nullptr;
    while (true) {
        cout << "임의의 정수 입력(종료: 0): ";
        cin >> num;
        if (num == 0)
            break;

        // 맨 마지막 노드로 삽입
        dList.addRear(num);
    }
```

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++ (3/13)

예제 4-7: 이중 연결 리스트

DLinkedList(demo).cpp

(2/2)

```
// 리스트 전체 노드의 데이터 출력
dList.printLinkedList();           // 순방향 출력
dList.printRevLinkedList();        // 역방향 출력

// 노드의 총 개수
cout << "노드의 총 개수: " << dList.countNode() << endl;

// 빈 리스트 여부 판단
if (dList.isEmpty()) {
    cout << "입력된 데이터가 없습니다!!!\n" << endl;
}
else {
    // 첫 번째 노드와 마지막 노드의 데이터
    cout << "첫 번째 노드의 데이터: " << dList.frontNode()->getData() << endl;
    cout << "마지막 노드의 데이터 : " << dList.rearNode()->getData() << endl;
}
cout << endl;

// dList.~DLinkedList();
return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
이의의정수인원(중요: 0): 1
이의의정수인원(중요: 0): 2
이의의정수인원(중요: 0): 3
이의의정수인원(중요: 0): 4
이의의정수인원(중요: 0): 5
이의의정수인원(중요: 0): 0
```

```
### 입력된 데이터: 순방향 ###
1 ->> 2 ->> 3 ->> 4 ->> 5 ->> NULL
```

```
### 입력된 데이터: 역방향 ###
5 ->> 4 ->> 3 ->> 2 ->> 1 ->> NULL
```

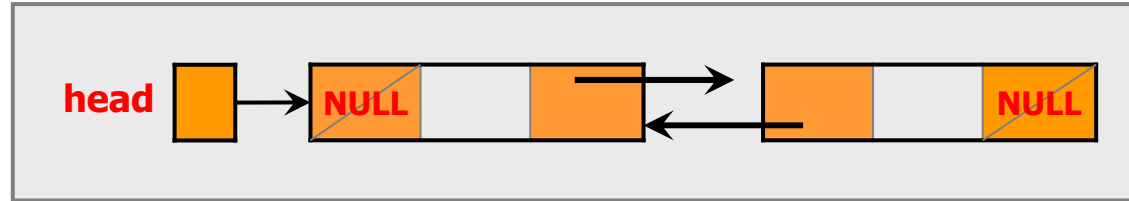
```
노드의 총 개수: 5
첫 번째 노드의 데이터: 1
마지막 노드의 데이터: 5
```


이중 연결 리스트: C++ (4/13)

예제 4-7: 이중 연결 리스트

LinkedList.h

```
// 이중 연결 리스트
// #pragma once
#ifndef __DNode_H__
#define __DNode_H__
```



```
// 노드: DNode(data, Llink, Rlink)
```

```
class DNode {
```

```
public:
```

```
    DNode(const int &data);
```

```
    int    getData(void) const;
```

```
private:
```

```
    int          data_;
```

```
    DNode        *Llink_;
```

```
    DNode        *Rlink_;
```

```
    friend class DLinkedList;
```

```
};
```

이중 연결 리스트: C++ (5/13)

예제 4-7: 이중 연결 리스트

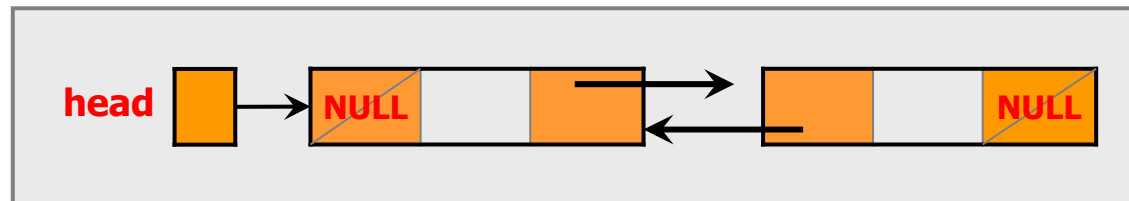
LinkedList.cpp

```
#include <iostream>

#include "LinkedList.h"      // DNode
using namespace std;

// DNode 생성자: 새로운 노드(DNode: data, link) 생성
DNode::DNode(const int &data):
    data_(data), Llink_(nullptr), Rlink_(nullptr) {}

// 노드의 데이터 반환
int    DNode::getData(void) const {
    return data_;
}
```



이중 연결 리스트: C++ (6/13)

예제 4-7: 이중 연결 리스트

DLinkedList(head).h

```
// #pragma once
#include "LinkedList.h"      // DNode

#ifndef __DLinkedList_H__
#define __DLinkedList_H__

// 이중 연결 리스트: DLinkedList(head)
class DLinkedList {
public:
    DLinkedList(void);      // 생성자
    ~DLinkedList(void);     // 소멸자: 전체 노드 삭제
    bool isEmpty(void) const; // 빈 리스트 여부 판단
    int countNode(void) const; // 탐색: 노드의 총 개수 (count_)
    DNode *frontNode(void) const; // 탐색: 첫 번째 노드 (head_)
    DNode *rearNode(void) const; // 탐색: 맨 마지막 노드 (tail_)
    void addRear(const int &e); // 삽입: 맨 마지막 노드 (tail_)
    void removeFront(void); // 삭제: 첫 번째 노드 (head_)
    void printLinkedList(void) const; // 출력: 리스트 전체 노드의 데이터 (순방향)
    void printRevLinkedList(void) const; // 출력: 리스트 전체 노드의 데이터 (역방향)

private:
    DNode *head_; // 첫 번째 노드
    DNode *tail_; // 맨 마지막 노드
    int count_; // 노드의 총 개수
};

#endif
```

이중 연결 리스트: C++ (7/13)

예제 4-7: 이중 연결 리스트

DLinkedList(head).cpp

(1/7)

```
#include <iostream>
#include "SLinkedList(head).h"           // SLinkedList >> head
// #include "SLinkedList(tail).h"       // SLinkedList >> head, count, tail
// #include "LinkedList.h"              // SNode
using namespace std;

// 생성자: 빈 리스트 생성(head)
DLinkedList::DLinkedList():
    head_(nullptr) { }

// 소멸자: 전체 노드 삭제
DLinkedList::~~DLinkedList() {
    // while (!isEmpty())
    //     removeFront();
    DNode *tNode = head_;
    while (tNode) {
        head_ = tNode->Rlink_;
        delete tNode;
        tNode = head_;
    }
}
```

head



이중 연결 리스트: C++ (8/13)

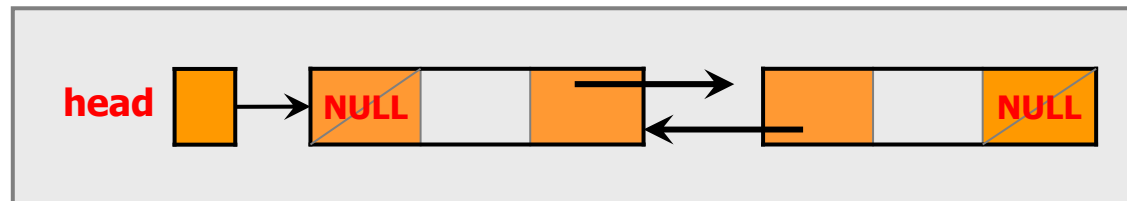
예제 4-7: 이중 연결 리스트

DLinkedList(head).cpp

(2/7)

```
// 빈 리스트 여부 판단(head)
bool DLinkedList::isEmpty() const {
    return head_ == nullptr;
}

// 탐색: 노드의 총 개수
int DLinkedList::countNode() const {
    int count = 0;
    DNode *rNode = head_;
    while (rNode) {
        ++count;
        rNode = rNode->Rlink_;
    }
    return count;
}
```



이중 연결 리스트: C++ (9/13)

예제 4-7: 이중 연결 리스트

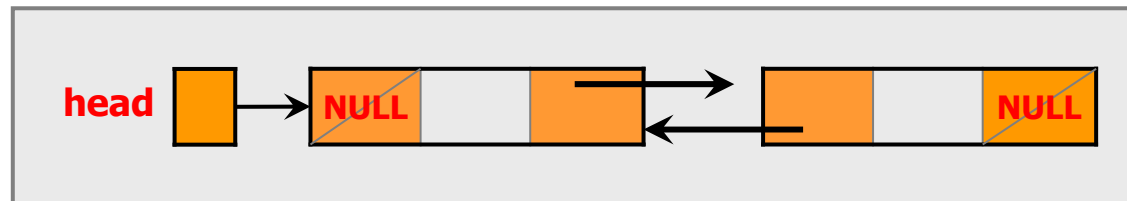
DLinkedList(head).cpp

(3/7)

```
// 탐색: 첫 번째 노드(head)
DNode *DLinkedList::frontNode() const {
    return head_;
}

// 탐색: 맨 마지막 노드
DNode *DLinkedList::rearNode() const {
    if (isEmpty())
        return nullptr;

    DNode *rNode = head_;
    while (rNode->Rlink_)
        rNode = rNode->Rlink_;
    return rNode;
}
```



이중 연결 리스트: C++ (10/13)

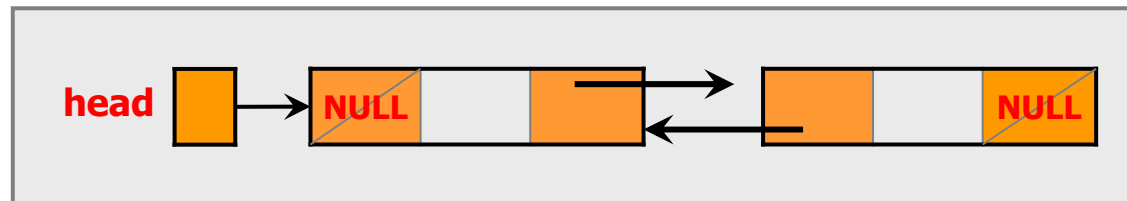
예제 4-7: 이중 연결 리스트

DLinkedList(head).cpp

(4/7)

// 삽입: 맨 마지막 노드

```
void DLinkedList::addRear(const int &e) {  
    DNode *newNode = new DNode(e);  
    if (isEmpty()) {  
        head_ = newNode;  
    }  
    else {  
        DNode *rNode = rearNode();  
        rNode->Rlink_ = newNode;  
        newNode->Llink_ = rNode;  
    }  
}
```



이중 연결 리스트: C++ (11/13)

예제 4-7: 이중 연결 리스트

DLinkedList(head).cpp

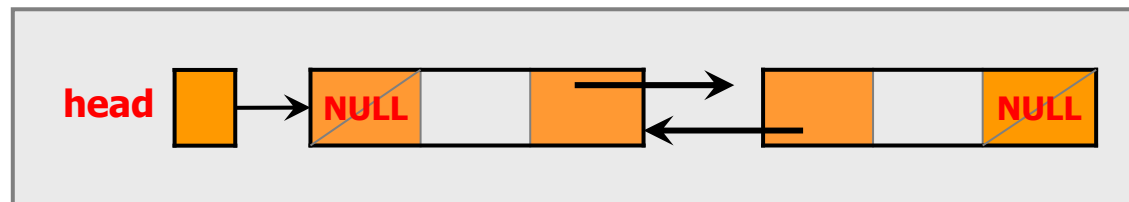
(5/7)

```
// 삭제: 첫 번째 노드(head)
void DLinkedList::removeFront() {
    if (isEmpty())
        return;

    DNode *old = head_;
    head_ = old->Rlink_;

    if (head_) // head_ != nullptr
        head_->Llink_ = nullptr;

    delete old;
}
```



이중 연결 리스트: C++ (12/13)

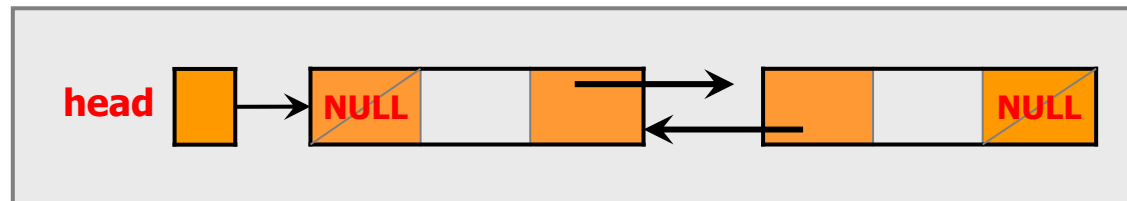
예제 4-7: 이중 연결 리스트

DLinkedList(head).cpp

(6/7)

```
// 출력: 리스트 전체 노드의 데이터(순방향)
void DLinkedList::printLinkedList() const {
    if (isEmpty()) {
        cout << "\n입력된 데이터가 없습니다!!!" << endl;
        return;
    }

    cout << "\n\t### 입력된 데이터 ###" << endl;
    DNode *tNode = head_;
    while (tNode) {
        cout.width(3);
        cout << tNode->data_ << " ->>";
        tNode = tNode->Rlink_;
    }
    cout << " NULL\n" << endl;
}
```



이중 연결 리스트: C++ (13/13)

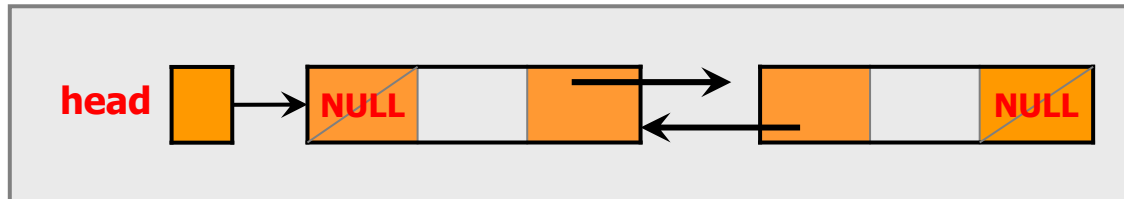
예제 4-7: 이중 연결 리스트

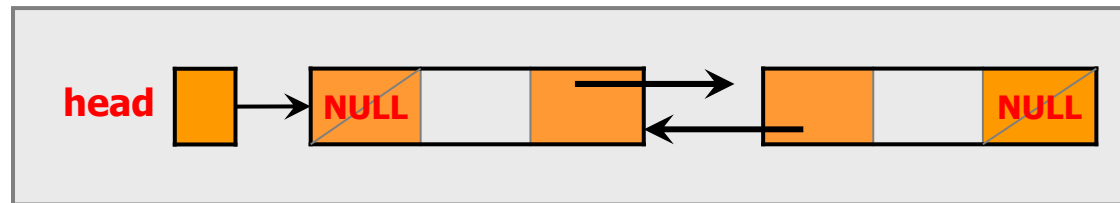
DLinkedList(head).cpp

(7/7)

// 출력: 리스트 전체 노드의 데이터(역방향)

```
void DLinkedList::printRevLinkedList() const {  
    if (isEmpty()) {  
        cout << "\n입력된 데이터가 없습니다!!!" << endl;  
        return;  
    }  
  
    cout << "\n\t### 입력된 데이터 ###" << endl;  
    DNode *rNode = rearNode();  
    while (rNode) {  
        cout.width(3);  
        cout << rNode->data_ << " ->>";  
        rNode = rNode->Link_;  
    }  
    cout << " NULL\n";  
}
```





연결 리스트 구현

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 "LinkedList.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: LinkedList.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);
```

THE
C
PROGRAMMING
LANGUAGE

이중 연결 리스트: C (2/12)

예제 4-6: 이중 연결 리스트

DLinkedList(demo).c (1/2)

```
#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 "LinkedList.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 디버깅 콘솔

입력	정수	입력	종료
1	0	1	0
2	0	2	0
3	0	3	0
4	0	4	0
5	0	5	0
6	0	6	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

LinkedList.h

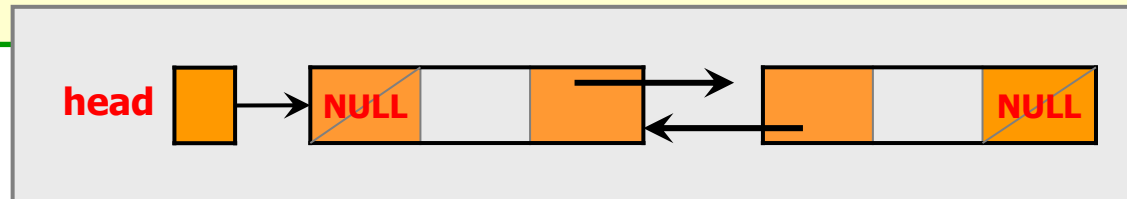
```
// #pragma once
typedef int    element;

// 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);
```



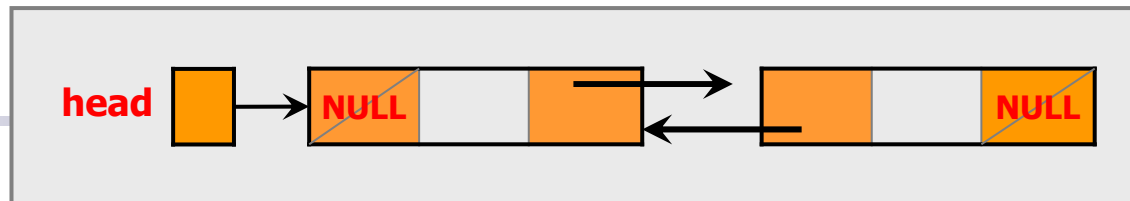
이중 연결 리스트: C (5/12)

예제 4-6: 이중 연결 리스트

LinkedList.c

```
#include <stdio.h>
#include <stdlib.h>          // malloc, free
#include "LinkedList.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 "LinkedList.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)

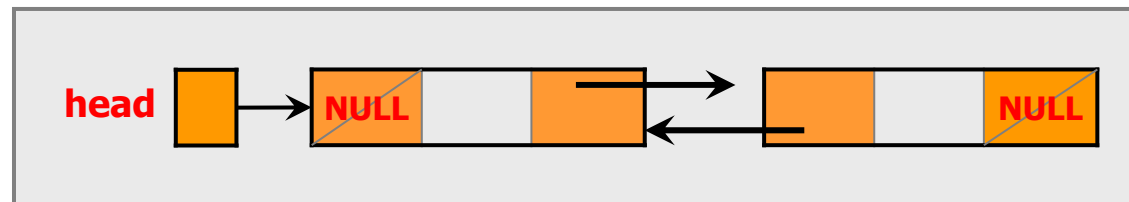
예제 4-6: 이중 연결 리스트

DLinkedList(head).c (1/6)

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "DLinkedList(head).h"
// #include "DLinkedList(tail).h"
// #include "LinkedList.h"

// exit, malloc, free
// bool, true, false
// DLinkedList >> head
// DLinkedList >> head, tail, count
// 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;
}
```

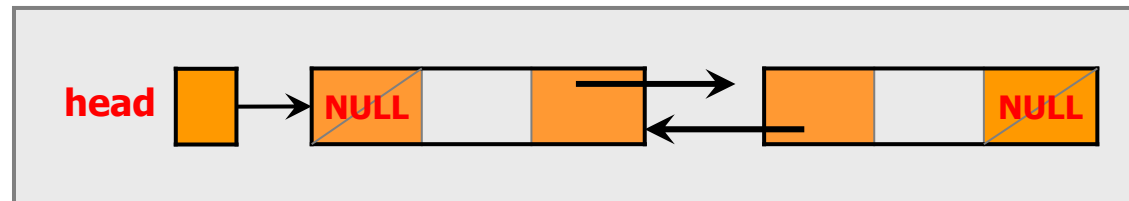


이중 연결 리스트: C (8/12)

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;
}
```



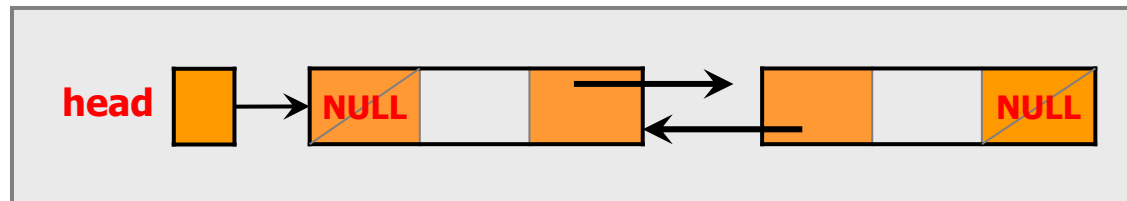
이중 연결 리스트: C (9/12)

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)

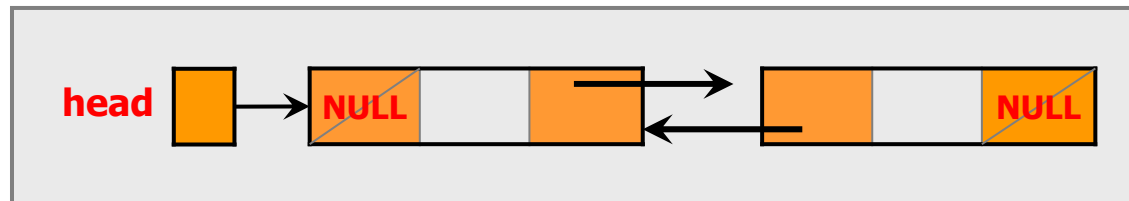
예제 4-6: 이중 연결 리스트

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);
}
```

head



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");
}

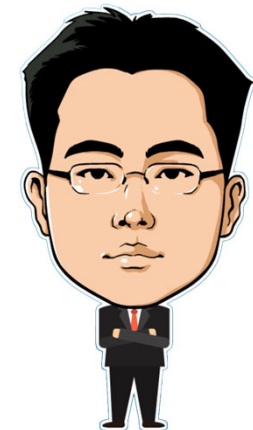
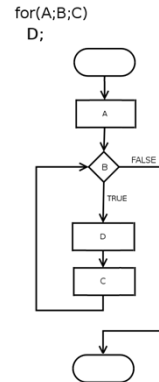
// 출력: 리스트 전체 노드의 데이터(역방향)
void revPrintDLinkedList(DLinkedList *dList) {
    if (dListEmpty(dList)) {
        printf("입력된 데이터가 없습니다!!!\n");
        return;
    }
    printf("\n ### 입력된 데이터(역방향) ### \n\n");
    DNode *rNode = rearDNode(dList);
    while (rNode) {
        printf("%3d ->>", rNode->data);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

head



참고문헌

- [1] "이것이 자료구조+알고리즘이다: with C 언어", 박상현, 한빛미디어, 2022.
- [2] "C++로 구현하는 자료구조와 알고리즘(2판)", Michael T. Goodrich, 김유성 외 2인 번역, 한빛아카데미, 2020.
- [3] "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 문병로, 한빛아카데미, 2022.
- [4] 문병로, "IT CookBook, 쉽게 배우는 알고리즘: 관계 중심의 사고법"(3판, 개정판, 한빛아카데미, 2024.
- [5] "코딩 테스트를 위한 자료 구조와 알고리즘 with C++", John Carey 외 2인, 황선규 역, 길벗, 2020.
- [6] "이것이 취업을 위한 코딩 테스트다 with 파이썬", 나동빈, 한빛미디어, 2020.
- [7] "SW Expert Academy", SAMSUNG, 2025 of viewing the site, <https://swexpertacademy.com/>.
- [8] "BAEKJOON", (BOJ) BaekJoon Online Judge, 2025 of viewing the site, <https://www.acmicpc.net/>.
- [9] "programmers", grepp, 2025 of viewing the site, <https://programmers.co.kr/>.
- [10] "goormlevel", goorm, 2025 of viewing the siteh, <https://level.goorm.io/>



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.