CHIP 2학년 수업

연결리스트 기초

2018년 1학기 2주차

목차

연결리스트 기본 구조

연결리스트 기본 구조

- 리스트(list)란?
  - 데이터의 목록을 다루는 구조가 단순한 자료구조
  - 가장 널리 쓰이는 자료구조
  - 선형 구조를 이룸.

- 연결 리스트(Linked List)
  - 리스트의 기본 구조인 노드(Node)가 선형으로 연결된 구조

연결리스트 기본 구조

- **노드(Node)** : 데이터+ 링크(포인터)로 구성된 기본적인 구조

- **노드 구조 코드(C언어)**

```
typedef int element;
typedef struct Node{
    element data;  // 노드의 데이터
    Node *next;    // 다음 노드를 가리키는 포인터
}Node;
```

연결리스트 기본 구조

- **Header Pointer**
  - 연결 리스트의 **가장 앞 부분을 가리키는** 포인터
  - 이 포인터를 이용해서 앞으로 사용할 연산들이 가능해짐
  - 실질적으로 **리스트 자체를 가리키는 포인터.**

- Java의 예시 :
  - ArrayList 클래스
  - **import** java.util.ArrayList;

- Python의 예시
  - 기본적으로 제공되는 자료형
  - Ex) List = [1,2,3,4,5]

왜 굳이 복잡한 연결리스트를 사용하나요?

차라리 배열쓰면 더 편하지 않을까요?

- **Q1 : 왜 굳이 복잡한 연결리스트를 사용하나요?**
  - A1 : 삽입/삭제 연산이 효율적이다.
    또한 동적인 구조라서 크기에 제한이 없다.

- **Q2 : 차라리 배열쓰면 더 편하지 않을까요?**
  - A2 : 배열이 쓰기는 더 편하다.
    하지만 효율성의 면에서는 리스트가 더 좋다.

예제

Arr

0	1	2	3	4	5	6	7	8
5	6	7	8	9	10	12	14	

- Arr라는 배열이 있다. 아래의 질문에 답을 해보아라.

1. index==4(Arr[4])에 15라는 값을 끼워 넣어보아라.

2. 1번에서 완성된 배열에서 16이라는 값을 더 넣어보아라. 잘 되는가?

예제

Arr

0	1	2	3	4	5	6	7	8
5	6	7	8	15	9	10	12	14

- Arr라는 배열이 있다. 아래의 질문에 답을 해보아라.

1. index==4(Arr[4])에 15라는 값을 끼워 넣어보아라.

```
for(int i = 8; i >=4 ; i--)
    Arr[i] = Arr[i-1];
Arr[4] = 15
```

2. 1번에서 완성된 배열에서 16이라는 값을 더 넣어보아라. 잘 되는가?

예제

Arr

- Arr라는 **리스트**가 있다. 아래의 질문에 답을 해보아라.

1. 다섯번 째 노드에 15라는 값을 끼워 넣어보아라.

```
insert_node(&Arr, 5, 15);
```

2. 1번에서 완성된 리스트에서 16이라는 값을 더 넣어보아라. 잘 되는가?

```
insert_node(&Arr, LAST_NODE, 16);
```

배열과 리스트와의 결론

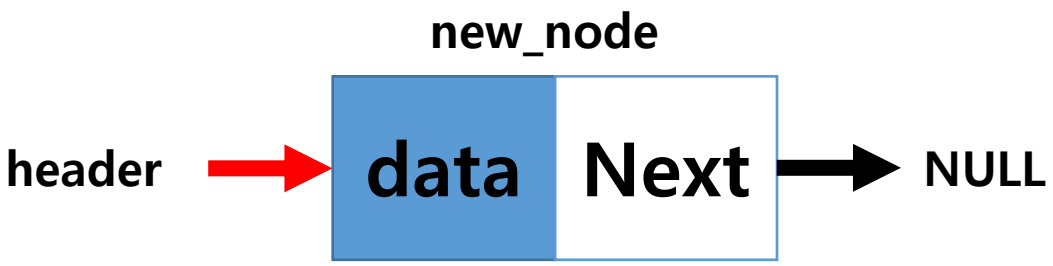
- **배열**
  - 구현이 쉽다.
  - 삽입/삭제 연산이 비효율적이다.
  - 크기가 고정되어 있다.

- **연결 리스트**
  - 구현이 복잡하다.
  - 삽입/삭제 연산이 효율적이다.
  - 크기 제약이 없다.

연결리스트 기본 연산

연결리스트 생성

- 연결 리스트의 노드를 새로 생성하는 함수



Node *header = create_node();

```
Node *create_node(element data){
    Node *new_node
    new_node = (Node)malloc(sizeof(Node));
    new_node->data = data;
    new_node->next = NULL;
    return new_node;
}
```

연결리스트 삽입

- 연결 리스트에 노드를 삽입하는 함수
- 하지만 삽입을 했을 때, 전제 조건이 있다.

  ✓절대로 Header Pointer나 앞의 포인터와의 **연결이 끊어지면 안된다.**

- 이러한 전제 조건 때문에 삽입 알고리즘은 3가지의 경우로 나눔

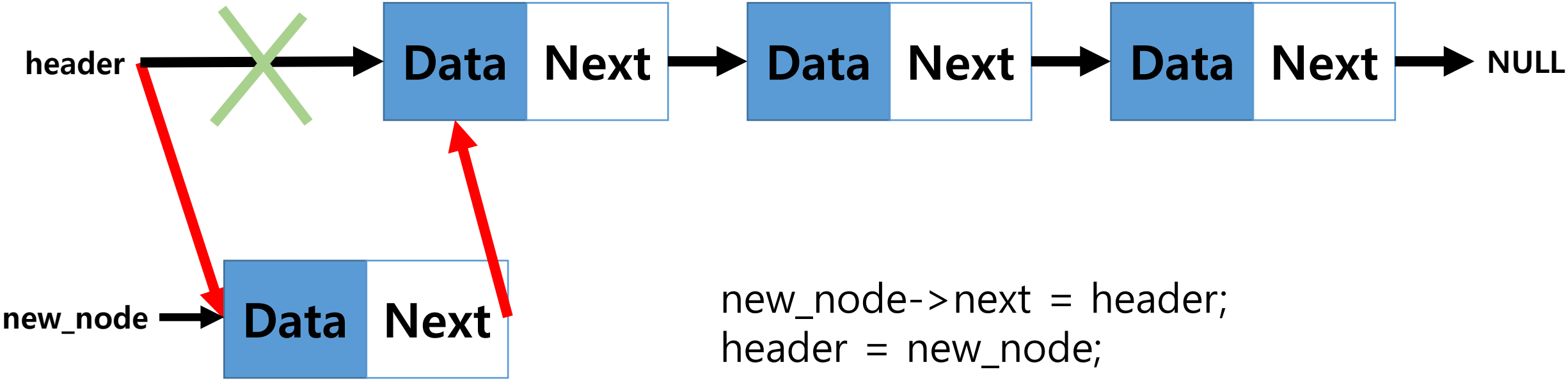
연결리스트 삽입

- **Case 1** : header pointer가 NULL인 경우 (아무것도 없을 때)


```
new_node->next = NULL;
header = new_node;
```

연결리스트 삽입

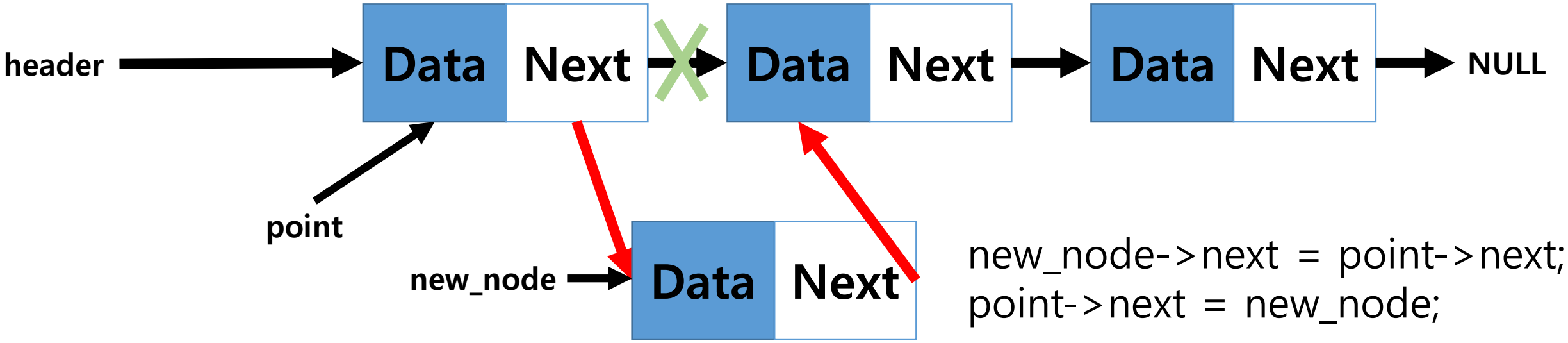
- **Case 2** : 리스트의 가장 앞에 삽입하는 경우




```
new_node->next = header;
header = new_node;
```

연결리스트 삽입

- **Case 3** : 리스트의 중간에 삽입하는 경우



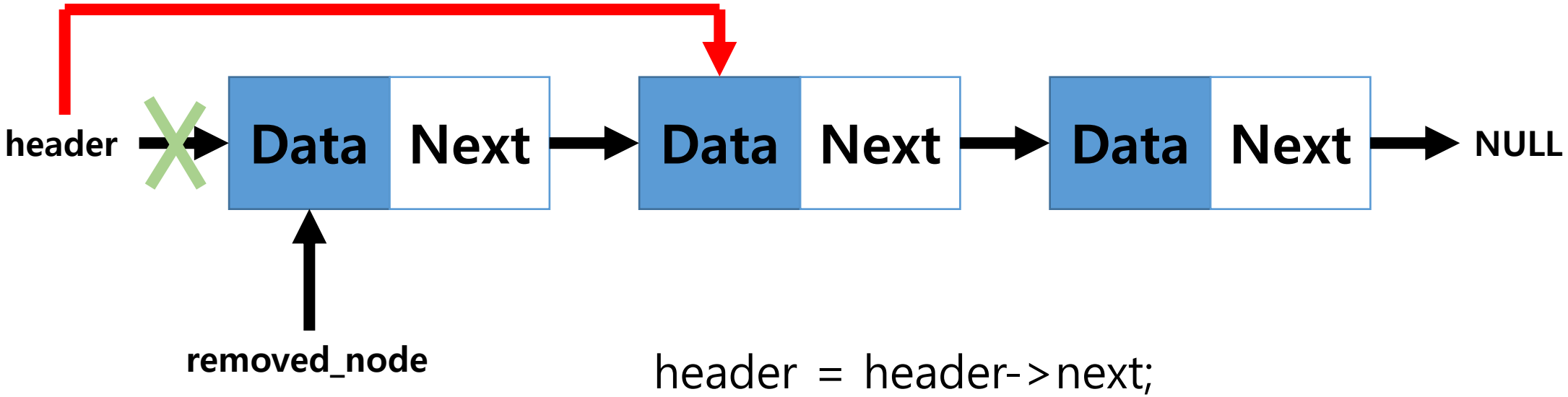

```
new_node->next = point->next;
point->next = new_node;
```

연결리스트 삽입

```
void insert_node(Node **header, Node *p, Node *new_node){
    if (*header == NULL){    // Case 1. 리스트에 노드가 아무것도 없을 때
        new_node->next = NULL;
        *header = new_node;
    }
    else if (p == NULL){      // Case 2. 리스트 가장 앞에 삽입을 할 때
        new_node->next = *header;
        *header = new_node;
    }
    else {                    // Case 3. 리스트 중간에 삽입을 할 때
        new_node->next = p->next;
        p->next = new_node;
   }
}
```

연결리스트 삭제

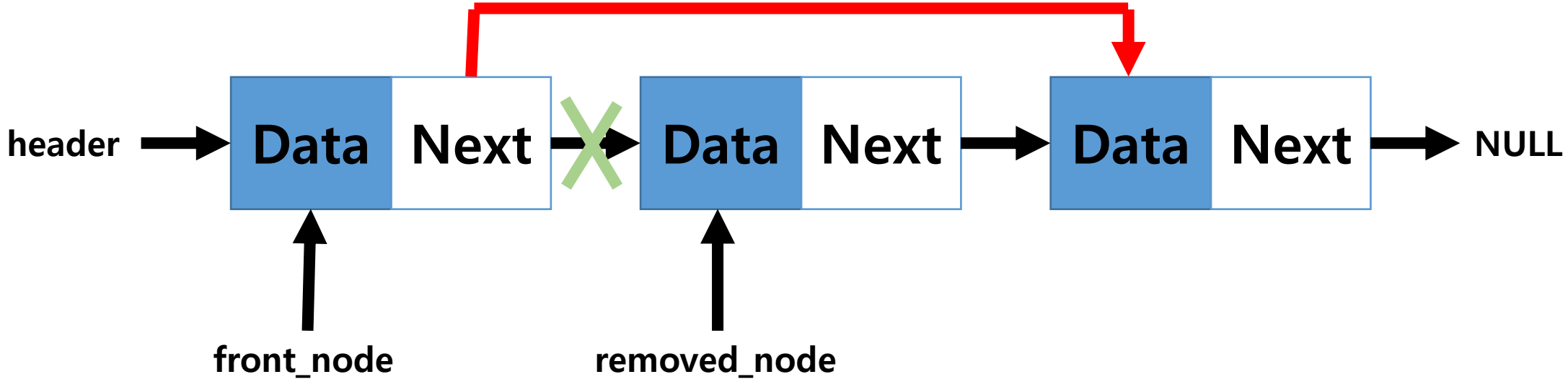
- **Case 1 :** 가장 앞 노드를 삭제




```
header = header->next;
```

연결리스트 삭제

- **Case 2 :** 지정한 노드를 삭제



front_node->next = removed_node->next;

연결리스트 삭제

```
void remove_node(Node **header, Node *front_node, Node *removed_node){
    if (front_node == NULL)    // Case 1. 가장 앞 노드를 삭제
        *header = (*header)->next;
    else                       // Case 2. 지정한 노드를 삭제
        front_node->next = removed_node->next;
    free(removed_node);
}
```

연결리스트 탐색 및 출력

```
Node *search_node(Node *header, int data){
    Node *point = head;
    while ( point != NULL ) {
        if ( p->data == data )
            return point;
        point = point->next;
    }
    return point;
}
```

```
void print_node(Node *header){
    Node *point = head;
    while ( point != NULL ) {
        printf("%d->", p->data);
        point = point->next;
    }
    printf("₩n");
}
```

실습

실습

1. int형으로 0부터 15까지 값을 가지는 연결 리스트를 생성하라.

2. 위의 연결 리스트에서 짝수 값을 가지는 노드를 삭제하라.

3. 위(2번까지 완성했던 것)의 연결 리스트를 출력하라.

선택문제 : 위의 연결리스트에서 짝수와 홀수를 나타내는 연결 리스트로 각각 나눠서 출력해보아라.