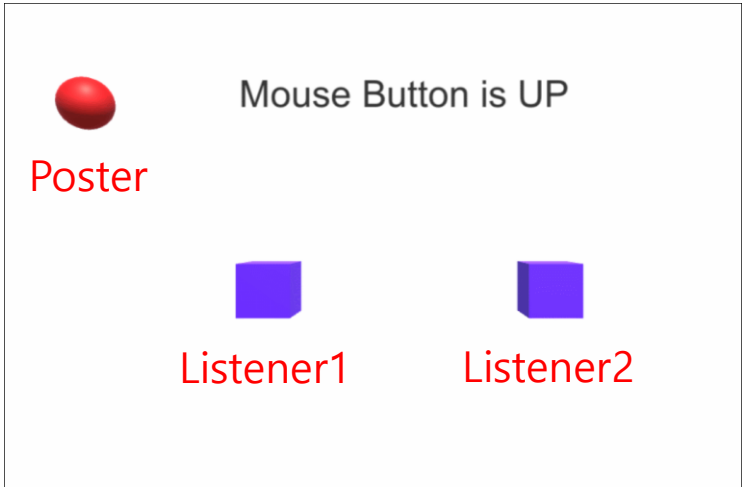
Data Structure

Linked List

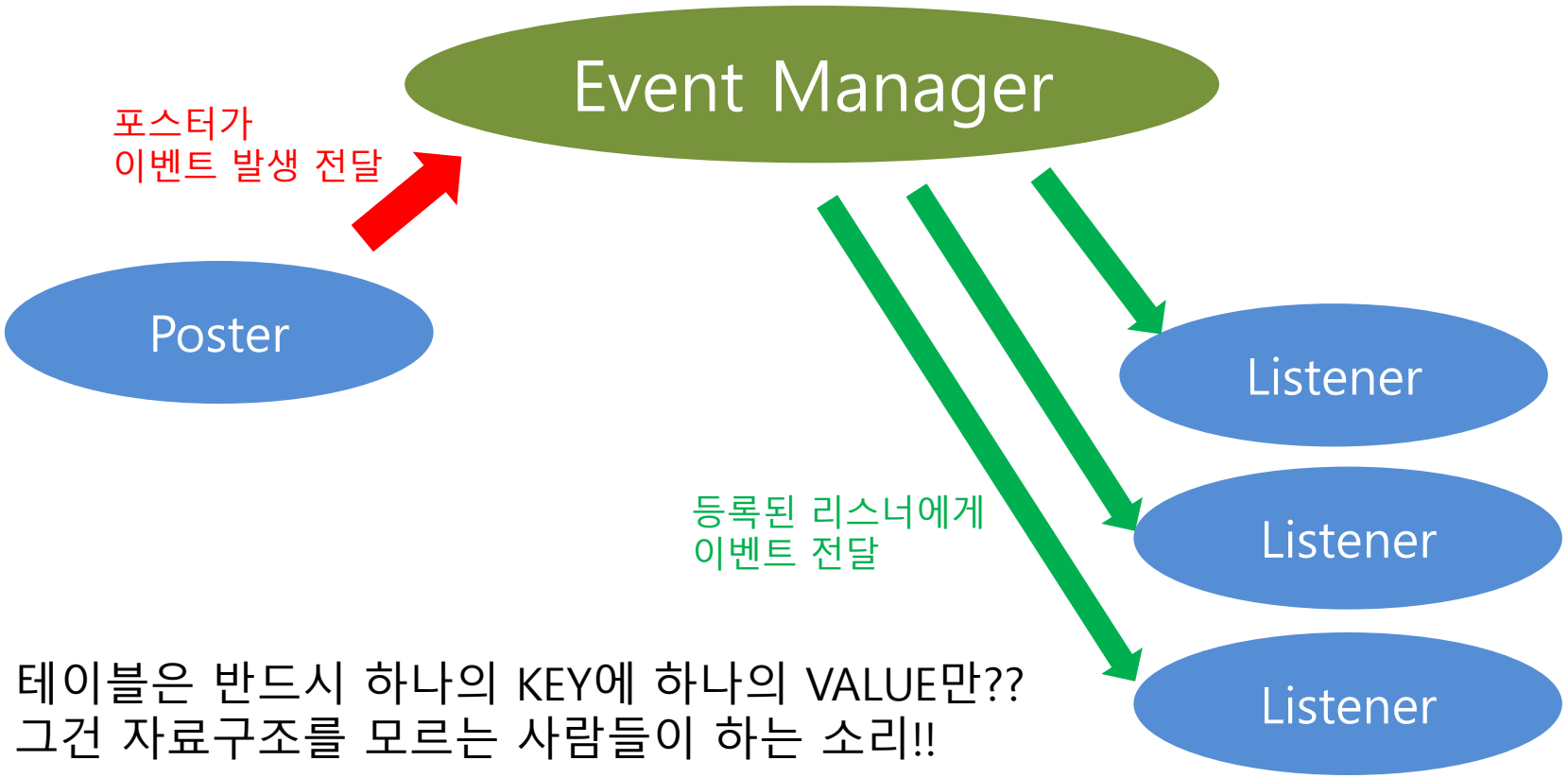
자료구조 왜 배우나요?



이벤트가 발생했을 때만 처리하는
이벤트 드리븐 프로그래밍 예제

```
private Dictionary<EventType, List<IListener>> listeners =
    new Dictionary<EventType, List<IListener>>();
```

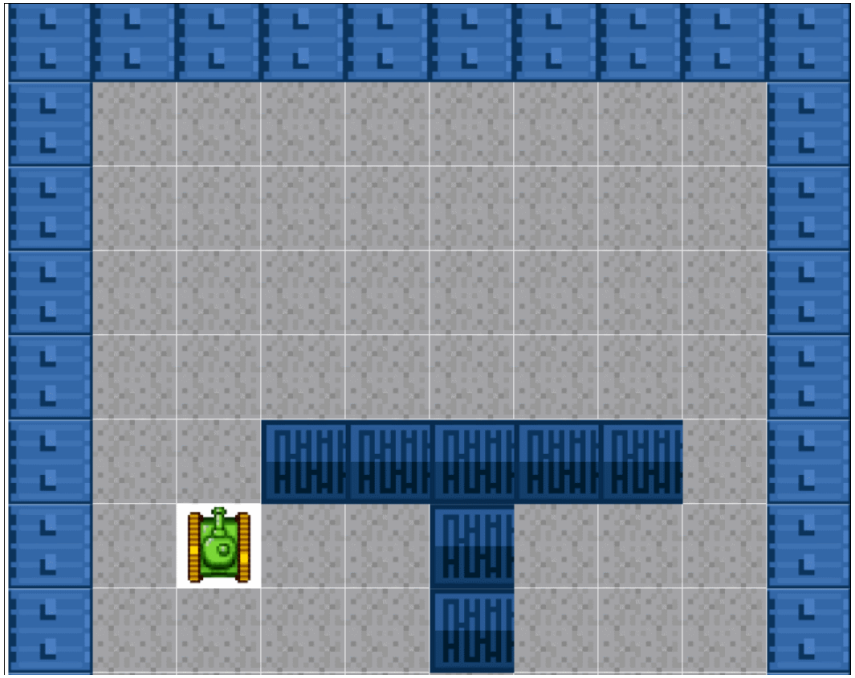
자료구조를 공부한 사람이라면,
Table이란 자료구조의 Closed addressing 기법이 사용되었군요.
라고 말할 겁니다.



테이블은 반드시 하나의 KEY에 하나의 VALUE만??
그건 자료구조를 모르는 사람들이 하는 소리!!

KEY	VALUE
이벤트 종류 1	Listener1 – listener2 – listener3- …….
이벤트 종류 2	Listener10-Listener11 – Listener12- …..
이벤트 종류 3	Listener 15 – Listener 16 – Listener17- …..

자료구조 왜 배우나요?



가장 기초적인 AI의 일종
- A star 알고리즘

구현하는 데 필요한 것
1. Priority Queue
2. Stack(or linked List)
3. 피타고라스 정리(?)
4. 대소비교(?)

자료구조만 알고 있으면
AI를 구현할 수 있다고??

```
class PathFinder
{
public:
    PriorityQueue<PathBlock*> * pQueue;//open node 담는다.
    PathBlock * pPathBlocks[10][10];//F, G, H 정보와 상태를 담는다.
    Stack<Position> * pList;
```

자료구조 왜 배우나요?

타임라인
14:27~15:55

리누스 토발즈가
리눅스에 대해 이야기 하면서
링크드 리스트에 대해
이야기 합니다.

개발자라면 TED에서 개발
관련 영상을 볼 수 있어야 하지
않을까요?

자료구조와 알고리즘

많은 분들이 자료구조와 알고리즘을 혼동해 사용하거나
같은 거라고 생각합니다. 하지만 이 두 가지는 다릅니다.
물론 밀접하게 연관되어 있지요.

1. 자료구조
   : 데이터를 어떤 구조로 저장하고 탐색해야 가장 효율적인가?

2. 알고리즘
   : 문제를 해결하는 방법론

자료구조와 알고리즘

1. 자료구조의 알고리즘
   : 데이터를 저장하고 탐색하는 방법에 대한 고민들

2. 자료구조를 이용한 알고리즘
   : 자료구조를 이용해 어떤 문제를 해결하는 것

자료구조의 구성

1. Insert
   : 데이터를 어떻게 저장할 것인가?

2. Search
: 데이터를 어떻게 탐색할 것인가?

3. Delete
: 필요 없어진 데이터를 어떻게 지울 것인가?

Array 와 Linked List

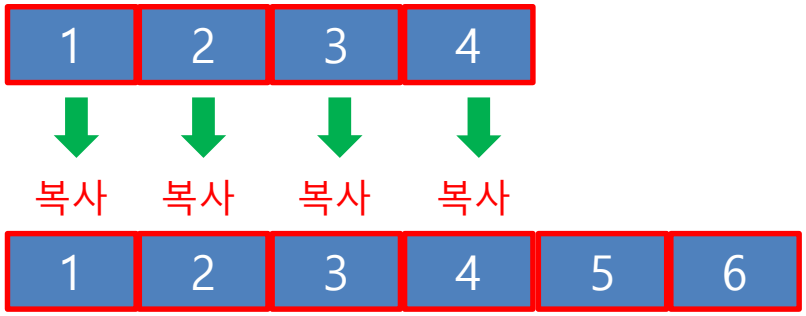
1. 배열의 원리

가변 배열 : 배열의 길이가 변하는 배열
하지만 배열은 원래 길이를 바꿀 수 없다?

길이가 4인 배열

만약 길이가 6인 배열이 필요하다면??

어마어마한 리소스 낭비!!!!!!

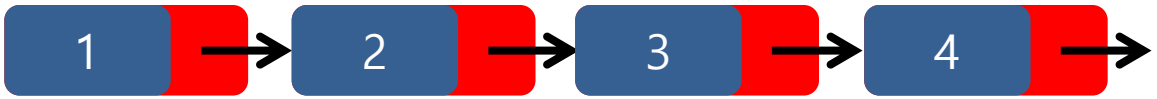


길이가 4인 배열은 그대로 두고
길이가 6인 배열을 따로 할당 후
(길이가 6인 메모리가 있는지 탐색이 이루어져야 함)
데이터의 복사가 이루어집니다.
그 후 길이가 4인 배열은 삭제합니다.

총 3번의 작업이 필요함!! + 메모리 탐색

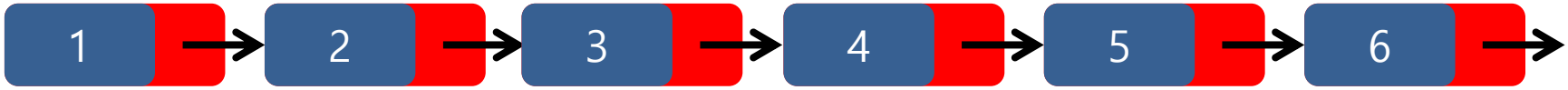
Array 와 Linked List

2. Linked List의 원리



길이가 4인 링크드 리스트

만약 길이가 6인 저장 공간이 필요하다면??



그냥 끝에다가 2 개의 노드만 붙이면 끝!

탐색, 할당, 복사, 삭제 등의
리소스 낭비가 전혀 없다!!!!!

Array 와 Linked List

Q. 그럼 무조건 Linked List가 좋은 건가요?

A. 아닙니다. 상황에 따라 적절한 자료구조를 사용해야 합니다.

Q. 그렇다면 배열이 좋을 때는 언제이지요?

A. 배열을 만들면 유관 데이터를 메모리에 순차적으로 쭉 나열할 수 있습니다. 이렇게 되면 Cache hit의 가능성이 커져서 성능에 큰 도움이 됩니다.

Dummy LinkedList

이번 시간에 다뤄볼 자료구조는
Linked List 중에서도 Dummy Linked List 입니다.

1. 노드란?

저장하고 싶은 데이터

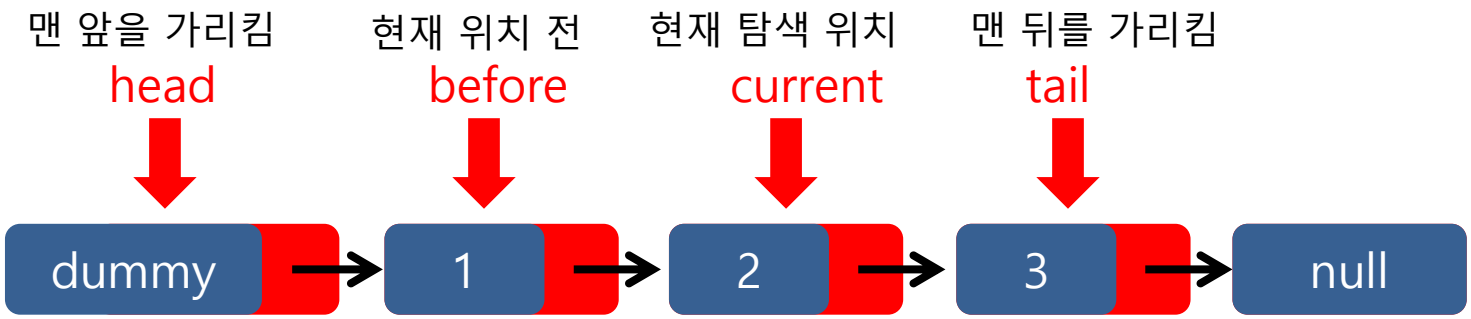
다음 노드를 가리키는 참조

Dummy LinkedList

2. 더미란?

더미란 실제 데이터를 담고 있는 노드가 아니라
구현의 편의를 위해 맨 앞에 두는 무의미한 노드입니다.

Dummy LinkedList의 구현



실제로 링크드 리스트 클래스 내에는 노드를 가리키는

1. 4개의 참조(C에서는 포인터)와
2. 데이터의 개수를 담고 있는 데이터 개수

만 존재함.

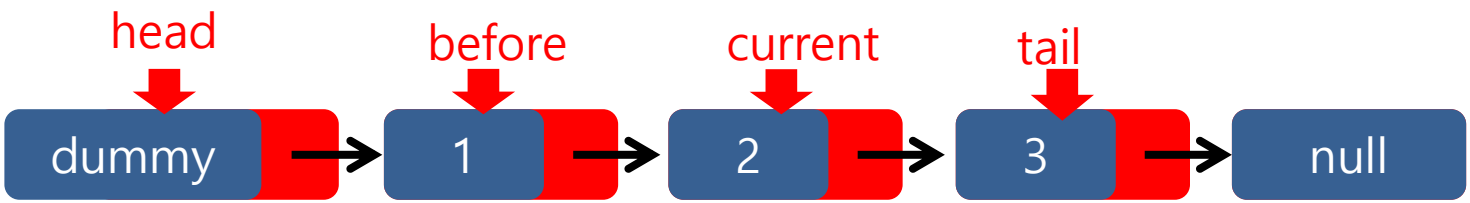
Dummy LinkedList의 구현

노드를 구현한 클래스

```
class Node
{
public:
    T data;
    Node * next;

    Node() { next = 0; }
    Node(T d) : data(d) { next = 0; }
};
```

Dummy LinkedList의 구현



1. 4개의 참조(C에서는 포인터)
2. 데이터의 개수를 담고있는 데이터 개수

멤버 변수

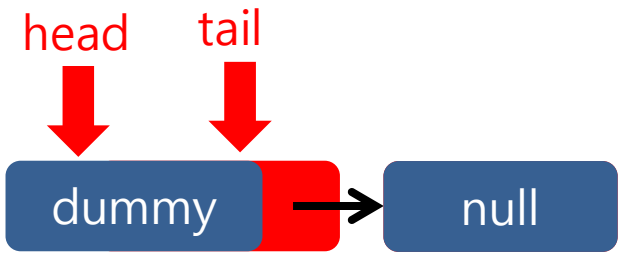
```
private:
    Node * head;
    Node * tail;
    Node * current;
    Node * before;
    int numOfData;
```

Dummy LinkedList의 구현

멤버함수 구현

1. 생성자 or 초기화

```
LinkedList()
    : current(0), before(0), numOfData(0)
{
    Node * dummy = new Node();
    head = dummy;
    tail = dummy;
}
```



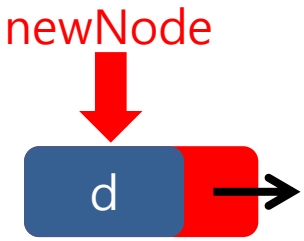
Dummy LinkedList의 구현

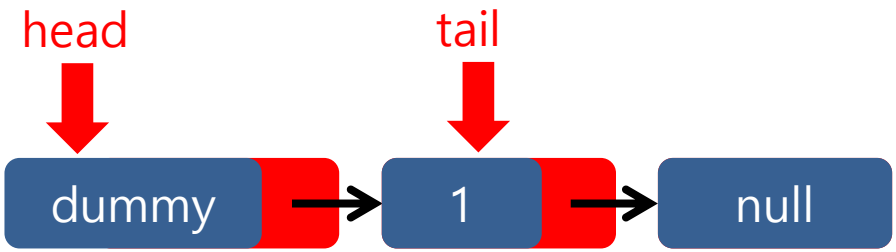
멤버함수 구현

2. Add() : data의 Insert 1

```
void LinkedList<T>::Add(T d)
{
    Node * newNode = new Node(d);

    tail->next = newNode;
    tail = newNode;
    numOfData++;
}
```





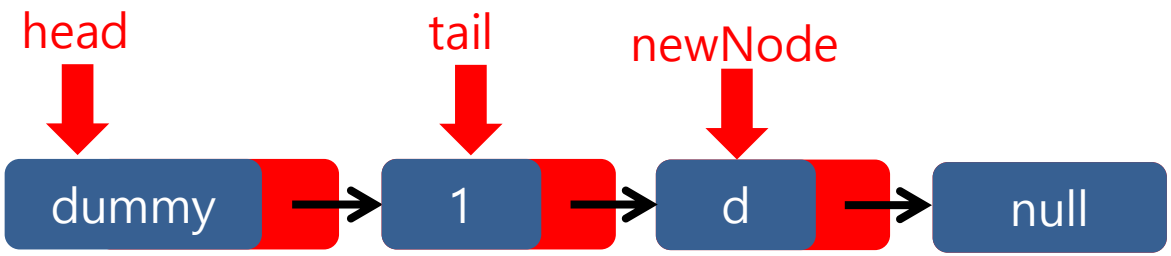
Dummy LinkedList의 구현

멤버함수 구현

3. Add() : data의 Insert 2

```
void LinkedList<T>::Add(T d)
{
    Node * newNode = new Node(d);

    tail->next = newNode;
    tail = newNode;
    numOfData++;
}
```



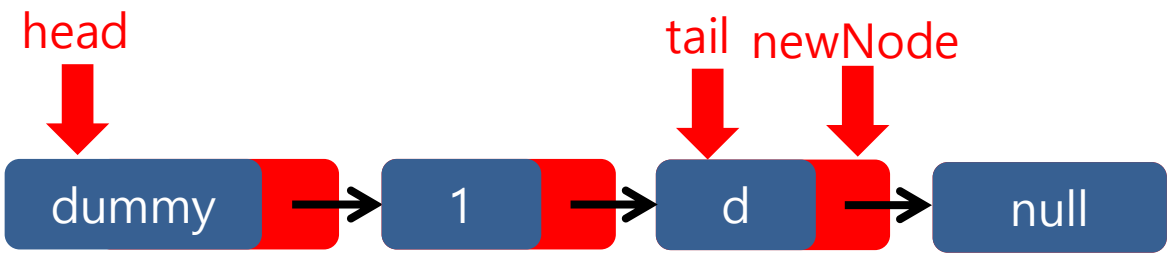
Dummy LinkedList의 구현

멤버함수 구현

4. Add() : data의 Insert 3

```
void LinkedList<T>::Add(T d)
{
    Node * newNode = new Node(d);

    tail->next = newNode;
    tail = newNode;
    numOfData++;
}
```



Dummy LinkedList의 구현

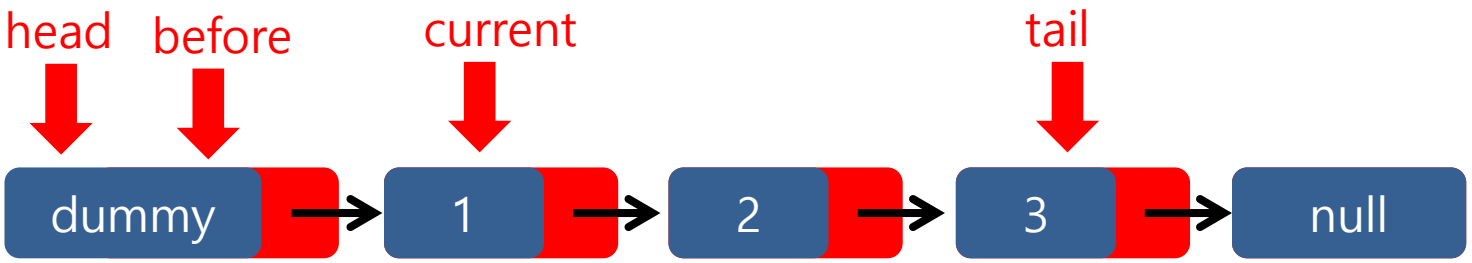
멤버함수 구현

5. First() : data의 search 1-1

```
bool LinkedList<T>::First(T * d)
{
    before = head;
    current = head->next;

    if (current != 0)
    {
        *d = current->data;
        return true;
    }

    return false;
}
```



Dummy LinkedList의 구현

멤버함수 구현

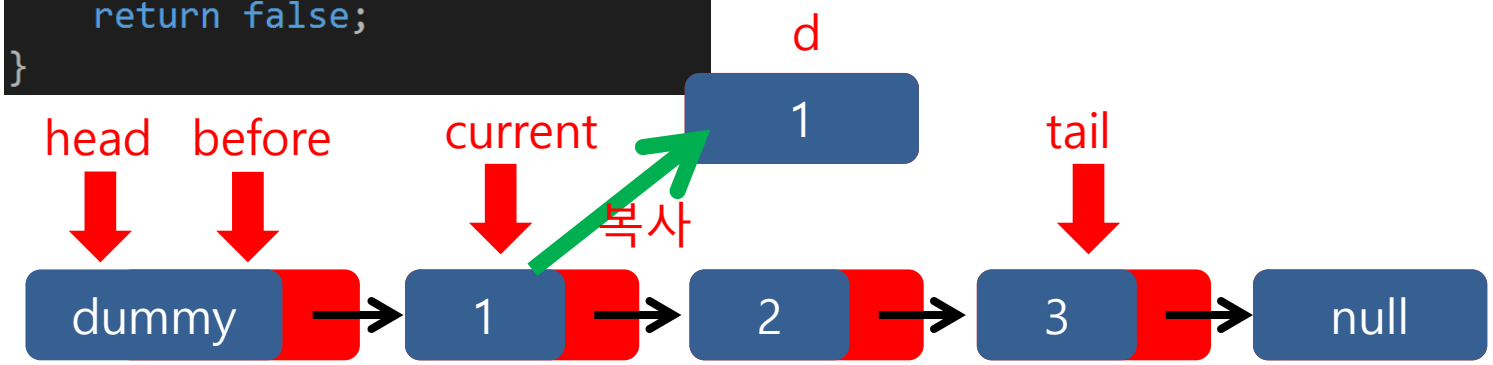
6. First() : data의 search 1-2

```
bool LinkedList<T>::First(T * d)
{
    before = head;
    current = head->next;

    if (current != 0)
    {
        *d = current->data;
        return true;
    }

    return false;
}
```

매개변수로 대입한
변수 d에 데이터를 담는다.



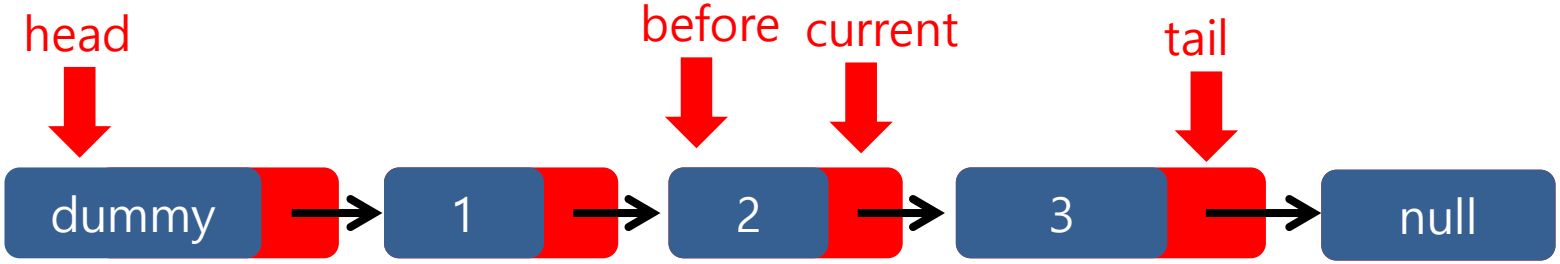
Dummy LinkedList의 구현

멤버함수 구현

7. Next() : data의 search 2-1

```
bool LinkedList<T>::Next(T * d)
{
    if (current->next == 0)
        return false;

    before = current;
    current = current->next;
    *d = current->data;
    return true;
}
```



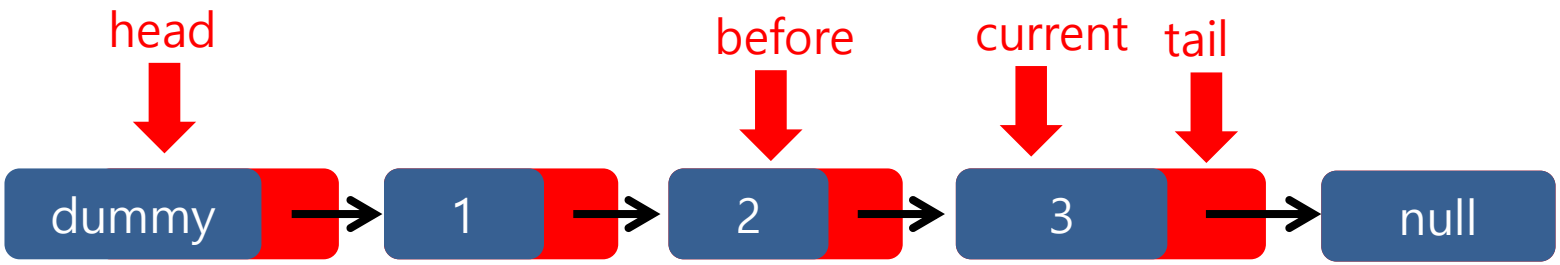
Dummy LinkedList의 구현

멤버함수 구현

8. Next() : data의 search 2-1

```
bool LinkedList<T>::Next(T * d)
{
    if (current->next == 0)
        return false;

    before = current;
    current = current->next;
    *d = current->data;
    return true;
}
```



Dummy LinkedList의 구현

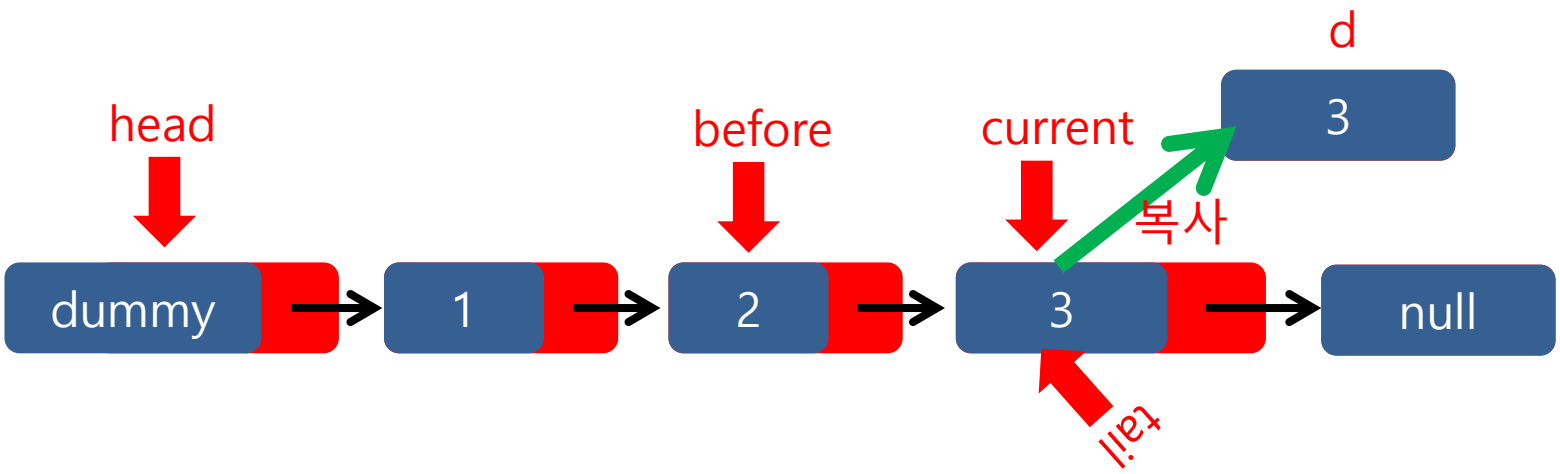
멤버함수 구현

9. Next() : data의 search 2-1

```
bool LinkedList<T>::Next(T * d)
{
    if (current->next == 0)
        return false;

    before = current;
    current = current->next;
    *d = current->data;
    return true;
}
```

매개변수로 대입한
변수 d에 데이터를 담는다.



Dummy LinkedList의 구현

멤버함수 구현

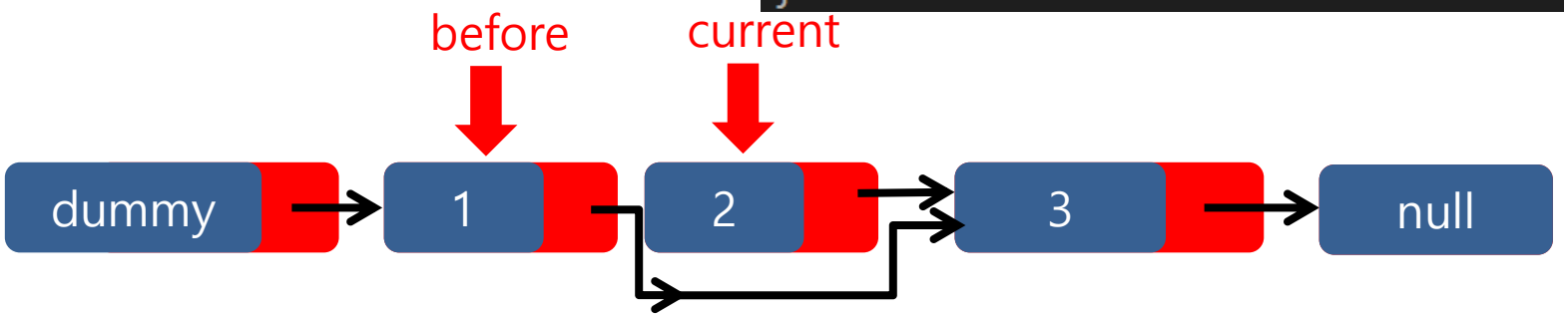
10. Delete() : data의 삭제 1

```
T LinkedList<T>::Delete()
{
    Node * delNode = current;
    T retData = delNode->data;

    if (delNode == tail)
        tail = head;


    before->next = current->next;
    current = before;
    numOfData--;
    delete delNode;

    return retData;
}
```



Dummy LinkedList의 구현

멤버함수 구현

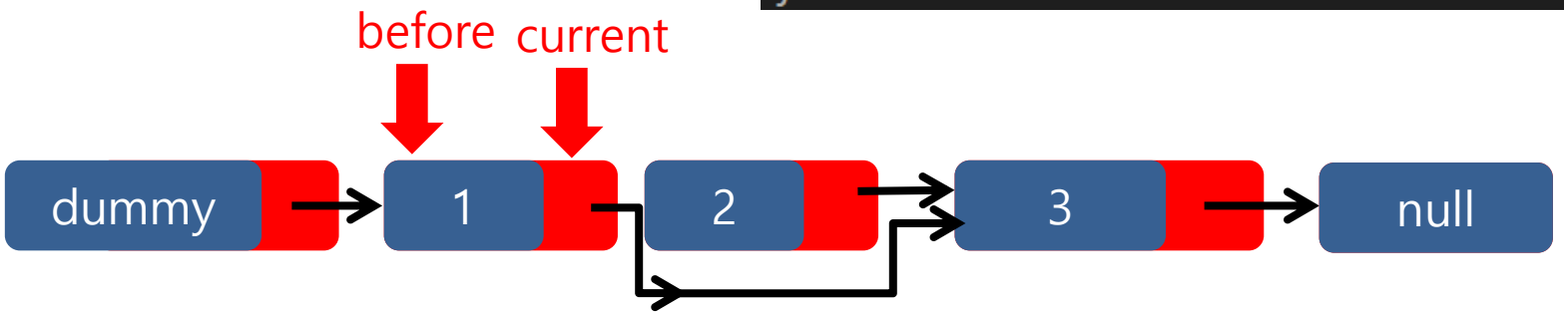
11. Delete() : data의 삭제 2

```
T LinkedList<T>::Delete()
{
    Node * delNode = current;
    T retData = delNode->data;

    if (delNode == tail)
        tail = head;


    before->next = current->next;
    current = before;
    numOfData--;
    delete delNode;

    return retData;
}
```



Dummy LinkedList의 구현

멤버함수 구현

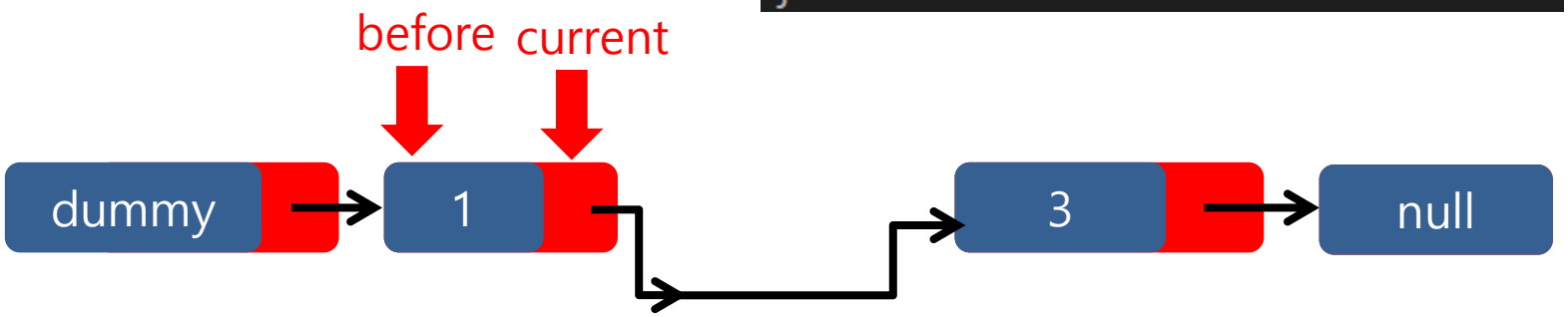
12. Delete() : data의 삭제 3

```
T LinkedList<T>::Delete()
{
    Node * delNode = current;
    T retData = delNode->data;

    if (delNode == tail)
        tail = head;


    before->next = current->next;
    current = before;
    numOfData--;
    delete delNode;

    return retData;
}
```



Dummy LinkedList의 구현

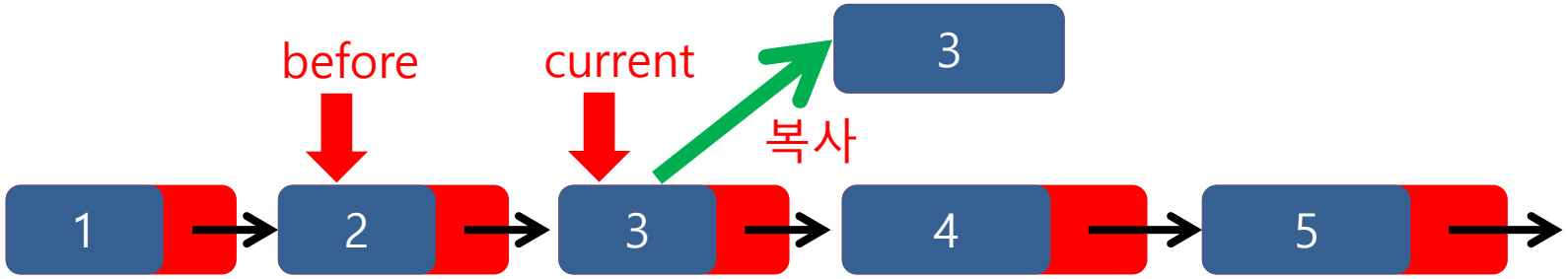
멤버함수 구현

13. delete 함수에서 current를 before로 옮기는 이유

```
bool LinkedList<T>::Next(T * d)
{
    if (current->next == 0)
        return false;

    before = current;
    current = current->next;
    *d = current->data;
    return true;
}
```

Next() 함수가 true를 반환하며
끝난다는 것은
Current가 마지막에 가리킨 노드의
Data를 이미 반환했다는 것

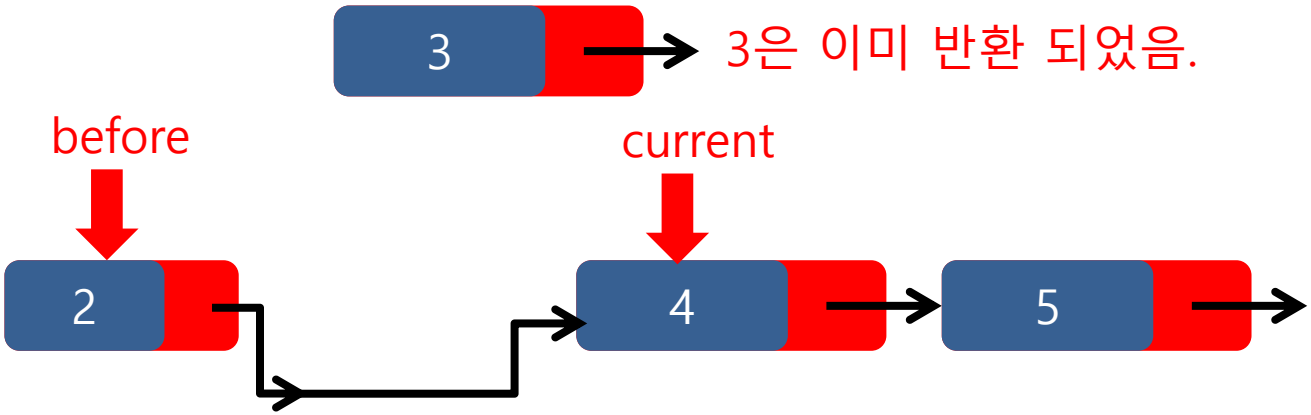


Dummy LinkedList의 구현

멤버함수 구현

13. delete 함수에서 current를 before로 옮기는 이유

current가 3이 저장된 데이터를 삭제 후 뒤 노드로 가버린다면



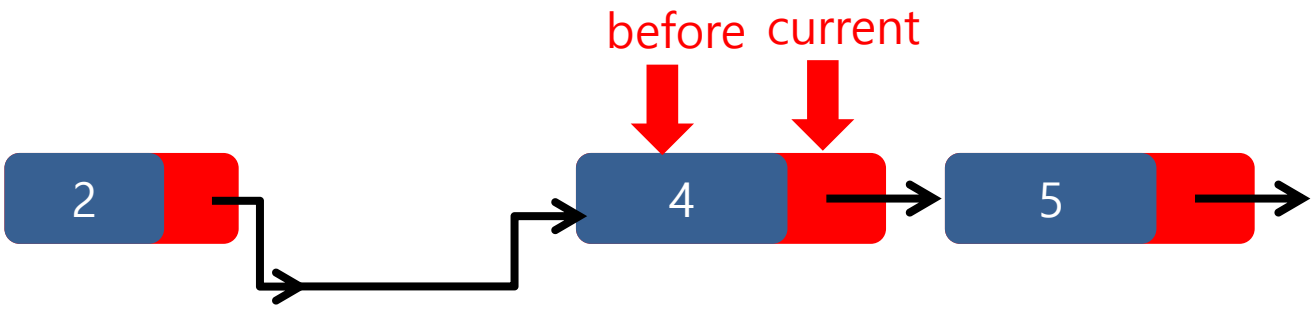
Dummy LinkedList의 구현

멤버함수 구현

13. delete 함수에서 current를 before로 옮기는 이유

```
bool LinkedList<T>::Next(T * d)
{
    if (current->next == 0)
        return false;

    before = current;
    current = current->next;
    *d = current->data;
    return true;
}
```



Dummy LinkedList의 구현

멤버함수 구현

13. delete 함수에서 current를 before로 옮기는 이유

```
bool LinkedList<T>::Next(T * d)
{
    if (current->next == 0)
        return false;

    before = current;
    current = current->next;
    *d = current->data;
    return true;
}
```

4는 반환되지 않는다!!!