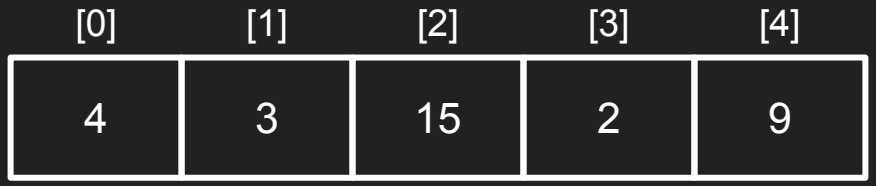
Data structure

Linked List

오경민

1. 배열

- 배열 : 같은 종류의 데이터들이 순차적으로 저장되어, 값의 번호가 곧 배열의 시작점으로부터 값이 저장되어 있는 상대적인 위치가 되는 자료 구조.
- 배열의 이름은 배열의 시작 주소이다.



배열의 구조

2. 배열의 장단점

장점

- 배열은 임의의 원소에 접근이 용이하다.

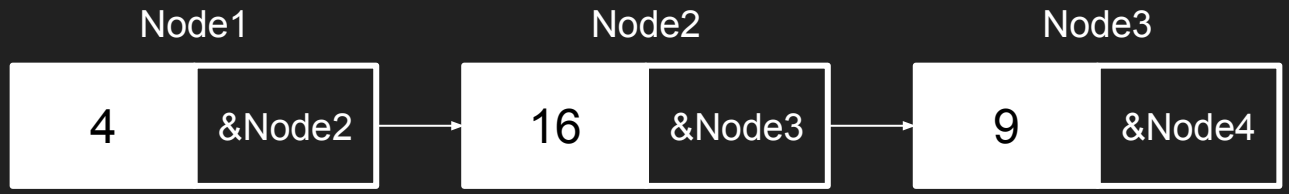
  [ ](index operator), *(pointer operator)를 사용해서 접근이 가능하다.

단점

- 삽입, 삭제를 할 시에 값을 이동시켜야 한다.
- 고정적인 메모리를 가지고 있어 확장이 힘들다.

3. 연결 리스트

- 연결 리스트 : 데이터를 저장할 때 그 다음 순서의 자료가 있는 위치를 데이터에 포함시키는 방식으로 자료를 저장한다.



연결 리스트

4. 연결 리스트의 장단점

장점

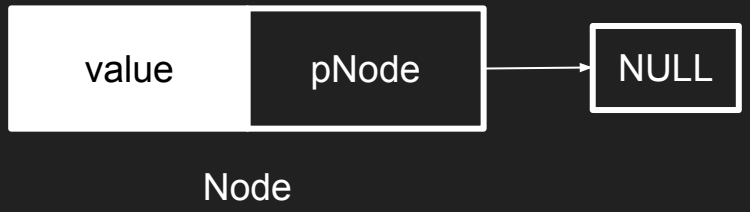
- 삽입, 삭제가 한번에 일어난다.
- 가변 메모리를 가지고 있어 메모리 확장, 축소에 용이하다.

단점

- 값에 접근할 때, 순차적으로 탐색을 거쳐야 함으로 느리다.

5. Single Linked List

- **Node** : 연결 리스트의 원소. 값과 다음 **Node**의 주소를 저장한다.
- 단일 연결 리스트 : 단 방향으로 연결이 되어있는 연결 리스트
  다음 노드가 없으면 **NULL**값을 저장한다.



5. Single Linked List

- Node 구현

C/C++

```c
typedef struct _Node {
    int value;
    struct _Node* next;
} Node;
```

Python

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
```

=

DataType Value
Node* next

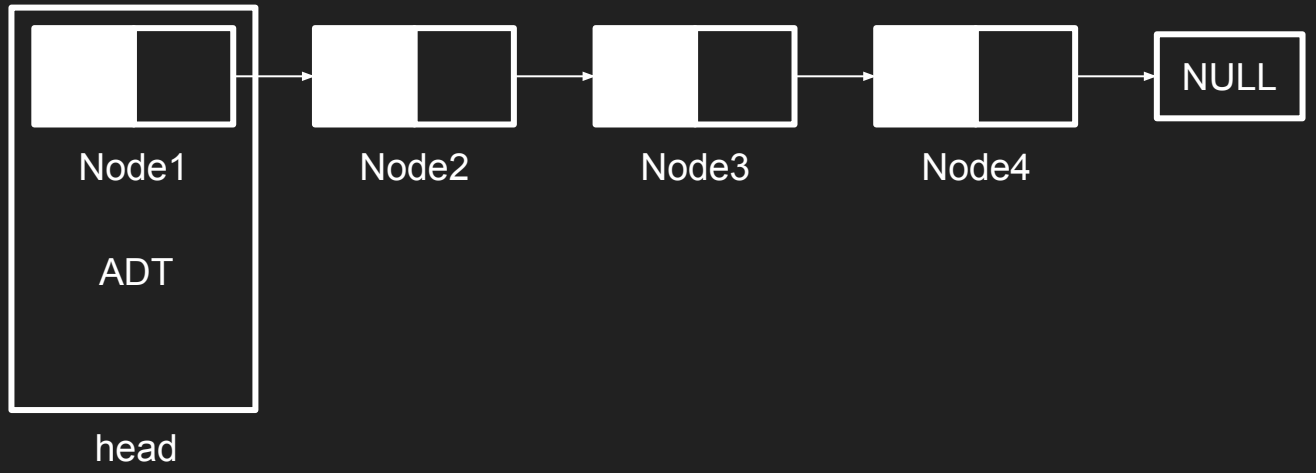
5. Single Linked List

ADT

1. **initialize** : 처음 노드가 생성되는 head 부분을 초기화한다.
2. **push_back** : 노드의 끝 부분에 새로운 노드를 붙인다.
3. **pop** : 노드의 끝 부분을 삭제한다.
4. **insert** : index에 해당하는 부분에 노드를 붙인다.
5. **erase** : index에 해당하는 부분의 노드를 삭제한다.
6. **empty** : 노드가 비어있는지 유무를 반환한다.
7. **size** : 노드의 개수를 반환한다.

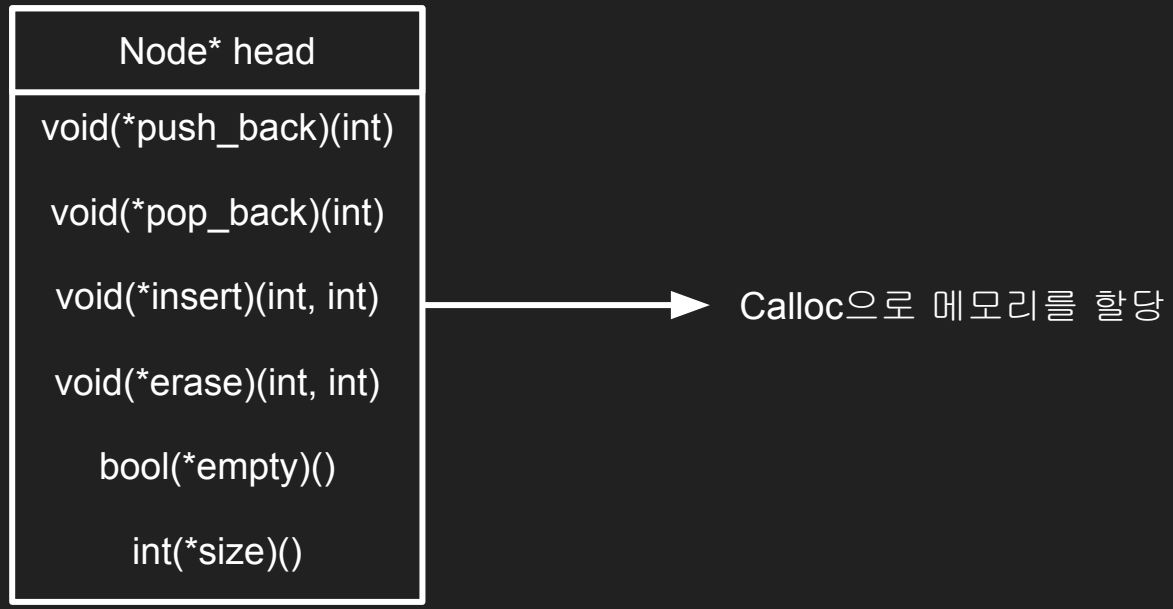
5. Single Linked List

- 구조



5. Single Linked List

- initialize()



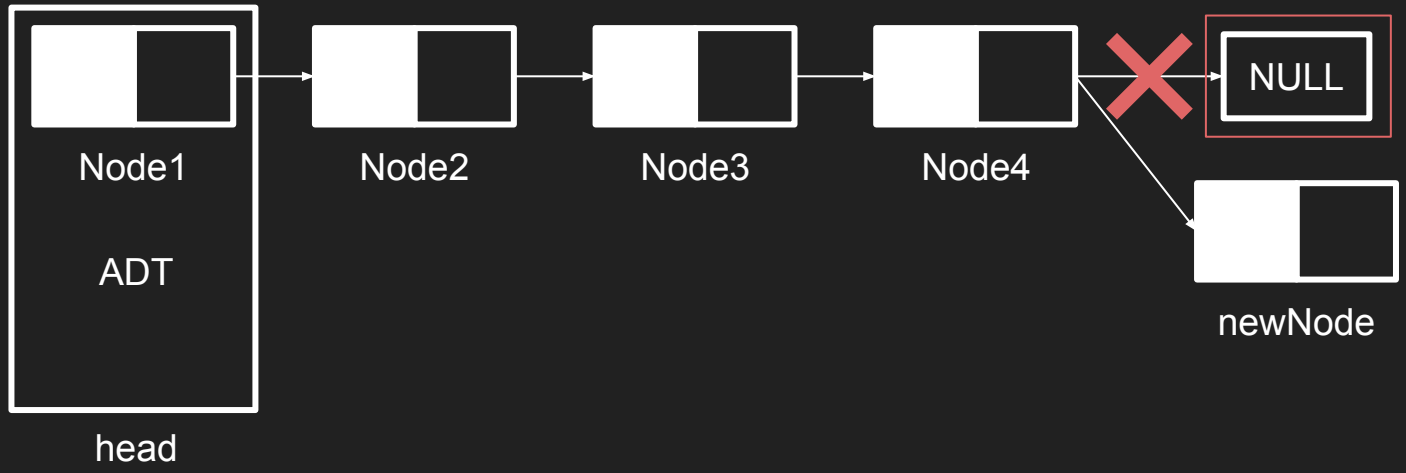
5. Single Linked List

- push_back()

1. Node에 next가 null인지 검사한다. (next가 null이면 노드의 마지막이다)
2. 새로운 Node를 생성한다.
3. 마지막 Node의 next에 새로운 Node를 넣는다.

5. Single Linked List

- push_back()



5. Single Linked List

- pop_back()

1. Node가 적어도 한개 이상 있는지 확인한다.
2. 없다면 반환, 있으면 삭제를 계속한다.
3. Node의 개수가 1개인지 2개 이상인지 검사한다.
4. 1개면 Node를 바로 free()한다.
5. 2개 이상이면 Node의 next의 next가 null인지 검사한다.
6. Node의 next를 free()하고 null값으로 바꾼다.

5. Single Linked List

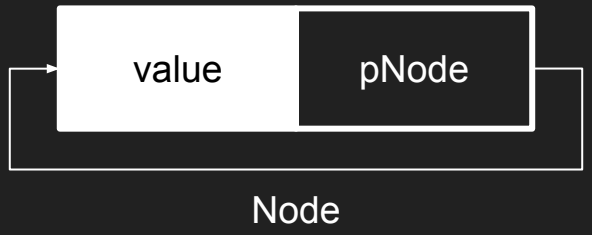
- insert()

5. Single Linked List

- erase()

6. Single Circular Linked List

- Node : 연결 리스트의 원소. 값과 다음 Node의 주소를 저장한다.
- 단일 원형 연결 리스트 : 단 방향으로 연결이 되어있는 연결 리스트
  다음 노드는 처음 Node의 주소 값을 저장한다.



6. Single Circular Linked List

- Node 구현

C/C++

```c
typedef struct _Node {
    int value;
    struct _Node* next;
} Node;
```

Python

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
```

=

DataType Value
Node* next

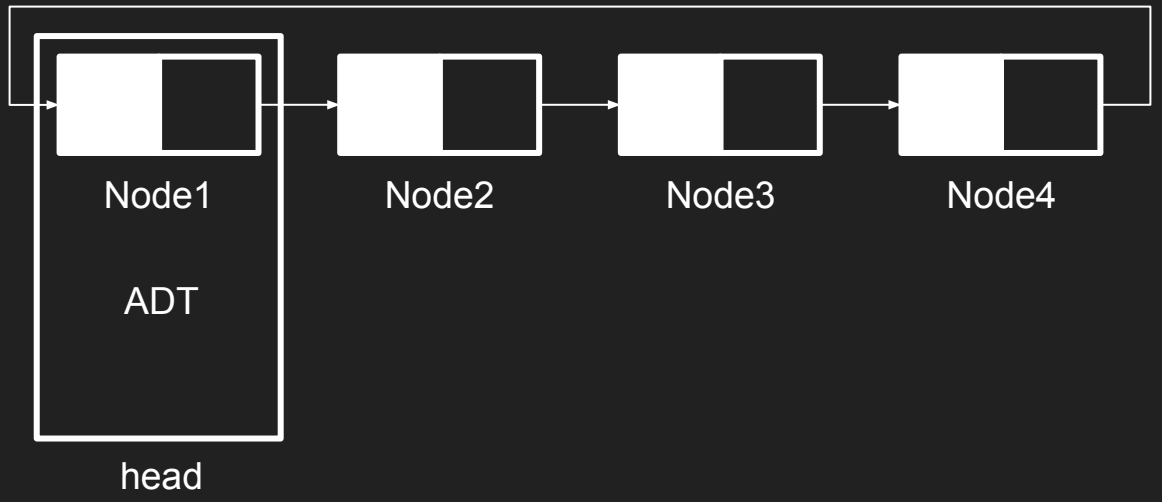
6. Single Circular Linked List

ADT

1. **initialize** : 처음 노드가 생성되는 head 부분을 초기화한다.
2. **push_back** : 노드의 끝 부분에 새로운 노드를 붙인다.
3. **pop** : 노드의 끝 부분을 삭제한다.
4. **insert** : index에 해당하는 부분에 노드를 붙인다.
5. **erase** : index에 해당하는 부분의 노드를 삭제한다.
6. **empty** : 노드가 비어있는지 유무를 반환한다.
7. **size** : 노드의 개수를 반환한다.

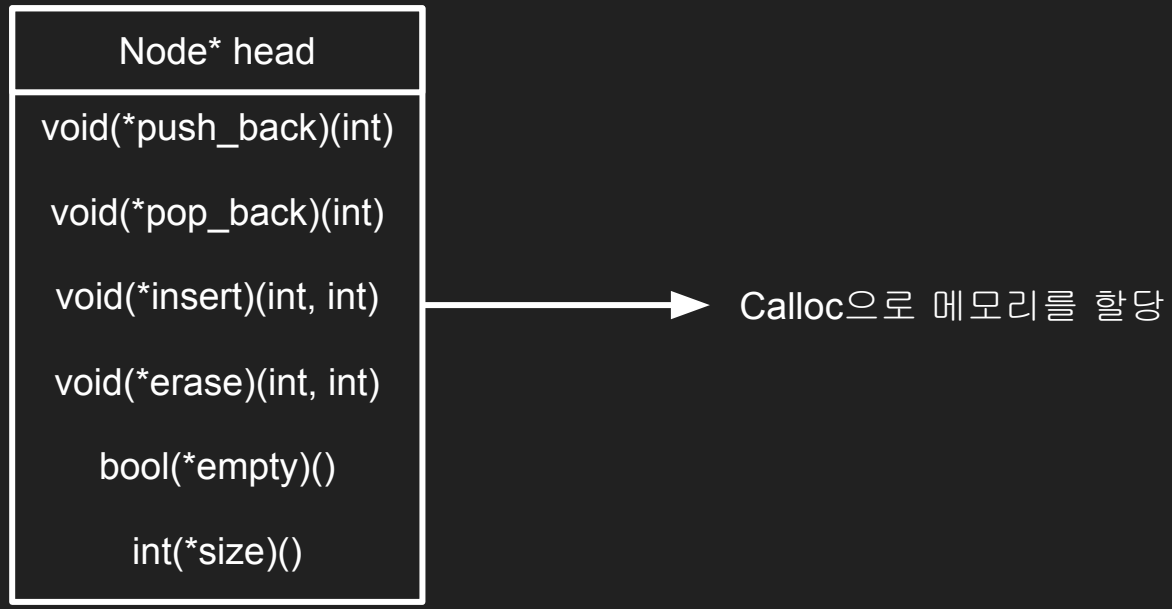
6. Single Circular Linked List

- 구조



6. Single Circular Linked List

- initialize()



6. Single Circular Linked List

- push_back()

6. Single Circular Linked List

- pop_back()

6. Single Circular Linked List

- insert()

6. Single Circular Linked List

- erase()