땅울림

04

시퀀스
Sequence

자료구조 스터디 4회차

- 시퀀스 개념설명
- 시퀀스 구현방법
- 유형 설명

데이터 요소들이 순서대로 저장된 자료구조
벡터와 리스트의 확장된 구조(인덱스와 반복자로 접근 가능)

->벡터 혹은 리스트를 통해 구현
실습에서는 **이중 연결 리스트**를 통해 시퀀스를 구현

배열과의 차이점?
시퀀스 내 요소들은 다른 자료형일 수 있고
크기를 변경할 수 있다.

—

시퀀스
Sequence

반복자 Iterator

시퀀스의 각 요소를 하나씩 순차적으로 접근할 수 있도록 도와주는 객체
요소의 위치를 나타냄

배열 - 인덱스로 요소 접근
리스트 - 반복자를 통해 요소 접근

—

시퀀스
Sequence

시퀀스 구현
: 시퀀스 멤버함수

---

insert(p, e)
: 위치 p에 e를 삽입

erase(p)
: 위치 p의 원소를 삭제

begin(), end()
: 시작 위치와 끝 위치

print()
: 순회하며 요소 출력

find(e)
: e값을 가진 요소 인덱스 출력

시퀀스 구현
: 반복자(Iterator)

연산자 오버로딩

++연산자
: 다음 노드로 이동

--연산자
: 이전 노드로 이동

이중연결리스트를 사용한 시퀀스

구현 1

시퀀스 구현 (이중연결리스트)

노드

```
class Node {
public:
    Node *next = nullptr;
    Node *prev = nullptr;
    int value = 0;
};
```

반복자

```
class Iterator {
public:
    Node *node = nullptr;
```

선언(main)

```
Sequence seq = Sequence();
Iterator p = Iterator();
```

시퀀스

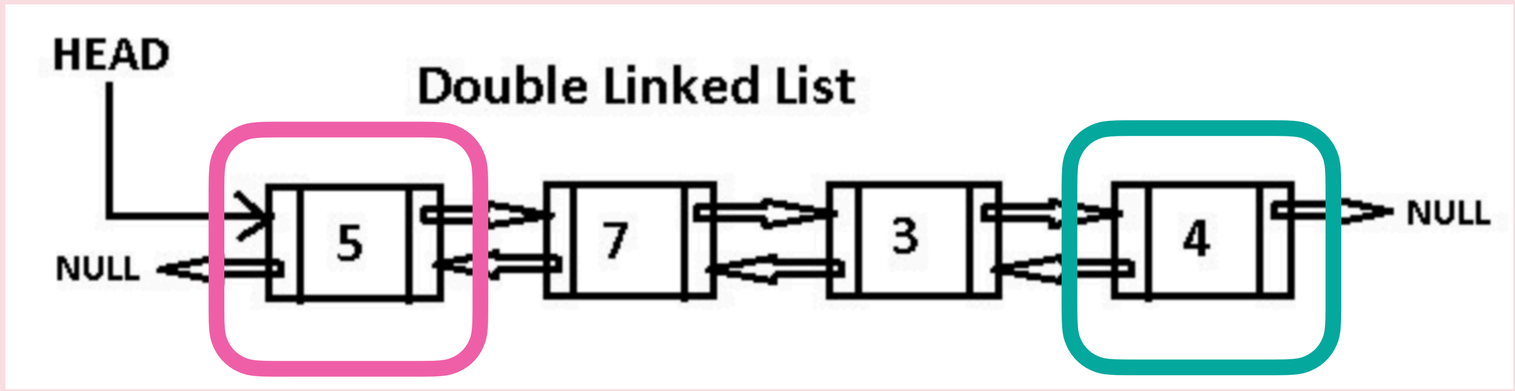
```
class Sequence {
public:
    Node *head;
    Node *tail;
    int size;

    Sequence() {
        head = new Node;
        tail = new Node;

        head->next= tail;
        tail->prev = head;
        size = 0;
    }
```

반복자 클래스

```
class Iterator {
public:
    Node *node = nullptr;

    Iterator *operator++() {
        node = node->next;
        return this;
    }
    Iterator *operator--() {
        node = node->prev;
        return this;
    }
};
```

++연산자 오버로딩
: 다음 노드로 이동

--연산자 오버로딩
: 이전 노드로 이동



시퀀스 구현 (이중연결리스트)

시퀀스 클래스

```
Node *begin() {
    return head->next;
}

Node *end() {
    return tail;
}
```

: 시작 위치 나타냄

: 끝 위치 나타냄

시퀀스 클래스 - 삽입연산

'반복자 p의 앞에 삽입'

시퀀스 구현 (이중연결리스트)

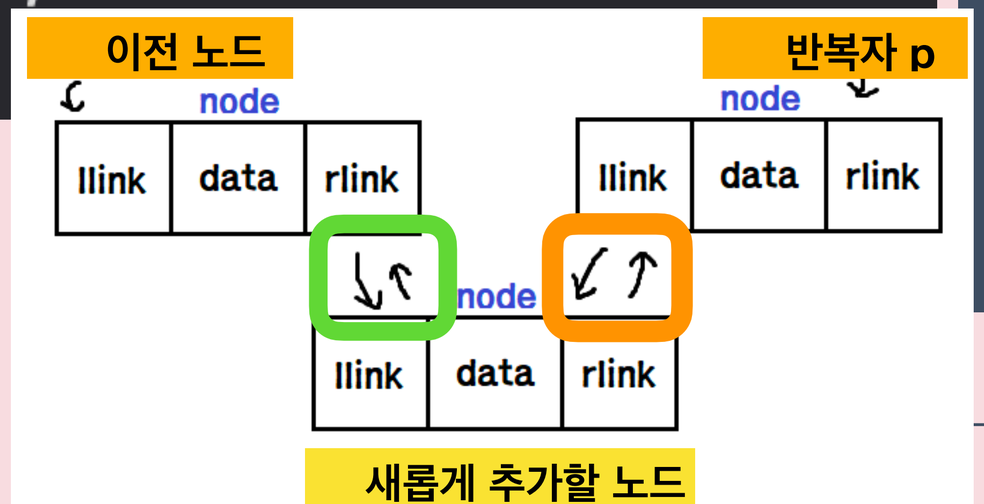
```
void insert(Iterator &p, int e) {
    Node *newNode = new Node();
    newNode->value = e;

    p.node->prev->next = newNode;
    newNode->prev = p.node->prev;

    newNode->next = p.node;
    p.node->prev = newNode;

    size++;
}
```

노드 생성 후 값 할당

이전 노드와 새 노드 연결

다음 노드와 새 노드 연결

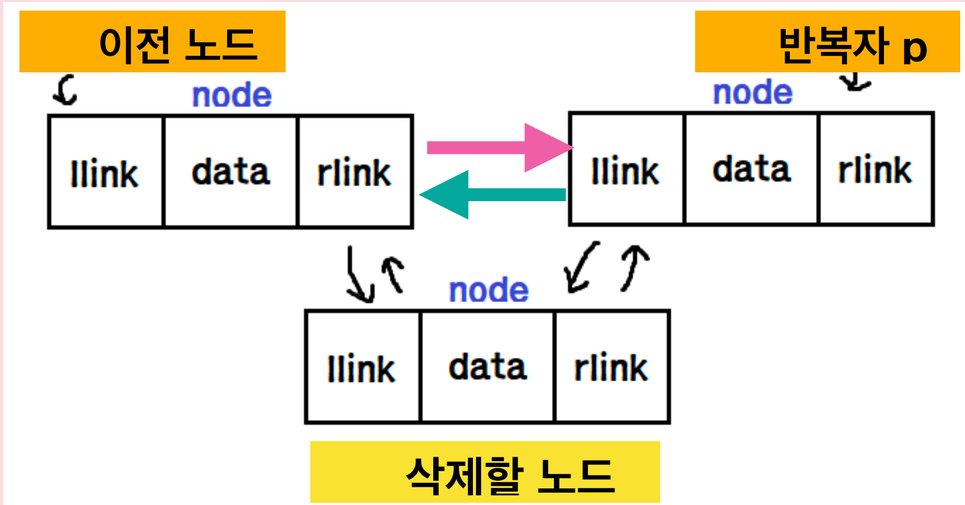
사이즈 증가



시퀀스 구현 (이중연결리스트)

시퀀스 클래스 - 삭제연산

```
void erase(Iterator &p) {
    if (size == 0) {
        cout << "Empty\n";
    } else {
        Node *deleteNode = p.node;

        deleteNode->prev->next =deleteNode->next;
        deleteNode->next->prev = deleteNode->prev;

        p.node = begin();
        size--;

        delete deleteNode;
    }
}
```



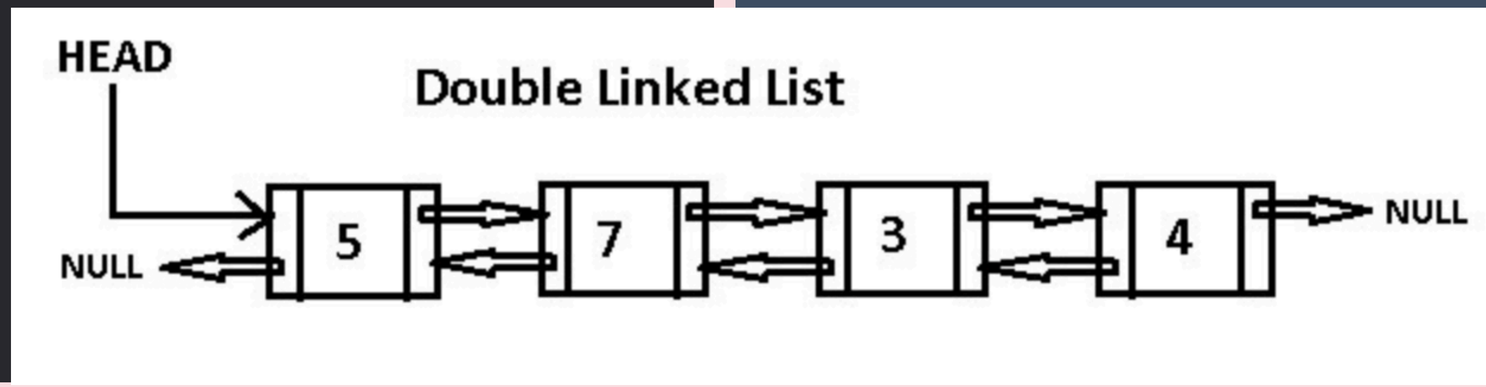
시퀀스 클래스 - 출력연산

시퀀스 구현 (이중연결리스트)

```
void print() {
    if (size == 0) {
        cout << "Empty\n";
    } else {
        Node *curNode = head->next;

        while (curNode != tail) {
            cout << curNode->value << " ";
            curNode = curNode->next;
        }

        cout << "\n";
    }
}
```

순회용 노드 초기화

tail 만나기 전까지 순회



시퀀스 클래스 - 인덱스 찾는 연산

```
void find(int e) {
    if (size == 0) {
        cout << "Empty\n";
    } else {
        Node *curNode = head->next;
        int index = 0;

        while (curNode != tail) {
            if (curNode->value == e) {
                cout << index << "\n";
                return;
            } else {
                curNode = curNode->next;
                index++;
            }
        }
        cout << "-1\n";
    }
}
```

시퀀스 구현 (이중연결리스트)

순회용 노드 초기화

tail 만나기 전까지 순회

값 같은 노드 발견하면 인덱스 출력 후 리턴

아니라면 다음 노드로 이동 후 인덱스 증가

문제 유형

1. 이중연결리스트로 시퀀스 구현
2. 이전 노드, 현 노드, 다음 노드 값들로 연산하기
   - 최대 풀링(최대값 찾기)
   - 컨볼루션(그냥 연산하기)
   - 세 노드값 합구하기
   - 최대값과 최소값 차이 구하기

응용 2 : 이웃노드 값으로 연산

반복자로 현재 노드, 이전 노드, 다음 노드의 값 접근해서 조건대로 연산하면 된다.

Solve)

```
void solving(Iterator &p) {

  int num1 = p.node->prev->value;      // 이전 노드
  int num1 = p.node->value;            // 현재 노드
  int num1 = p.node->next->value;      // 다음 노드

  /*
   이부분은 문제 조건대로 구현, 출력
  */

}
```

응용 2 : 이웃노드 값으로 연산

```
void maxPooling(Iterator &p) {
    int num1 = p.node->prev->value;
    int num2 = p.node->value;
    int num3 = p.node->next->value;

    /*
     문제 조건에 맞춰서 이부분 구현
     -> 이 문제는 세 수 비교해서 max 값 출력
     */
    if (num1 > num2 && num1 > num3) {
        cout << num1 << " ";
    } else if (num2 > num1 && num2 > num3) {
        cout << num2 << " ";
    } else if (num3 > num1 && num3 > num2) {
        cout << num3 << " ";
    }
}
```

문제 풀어보기~

시퀀스
Sequence