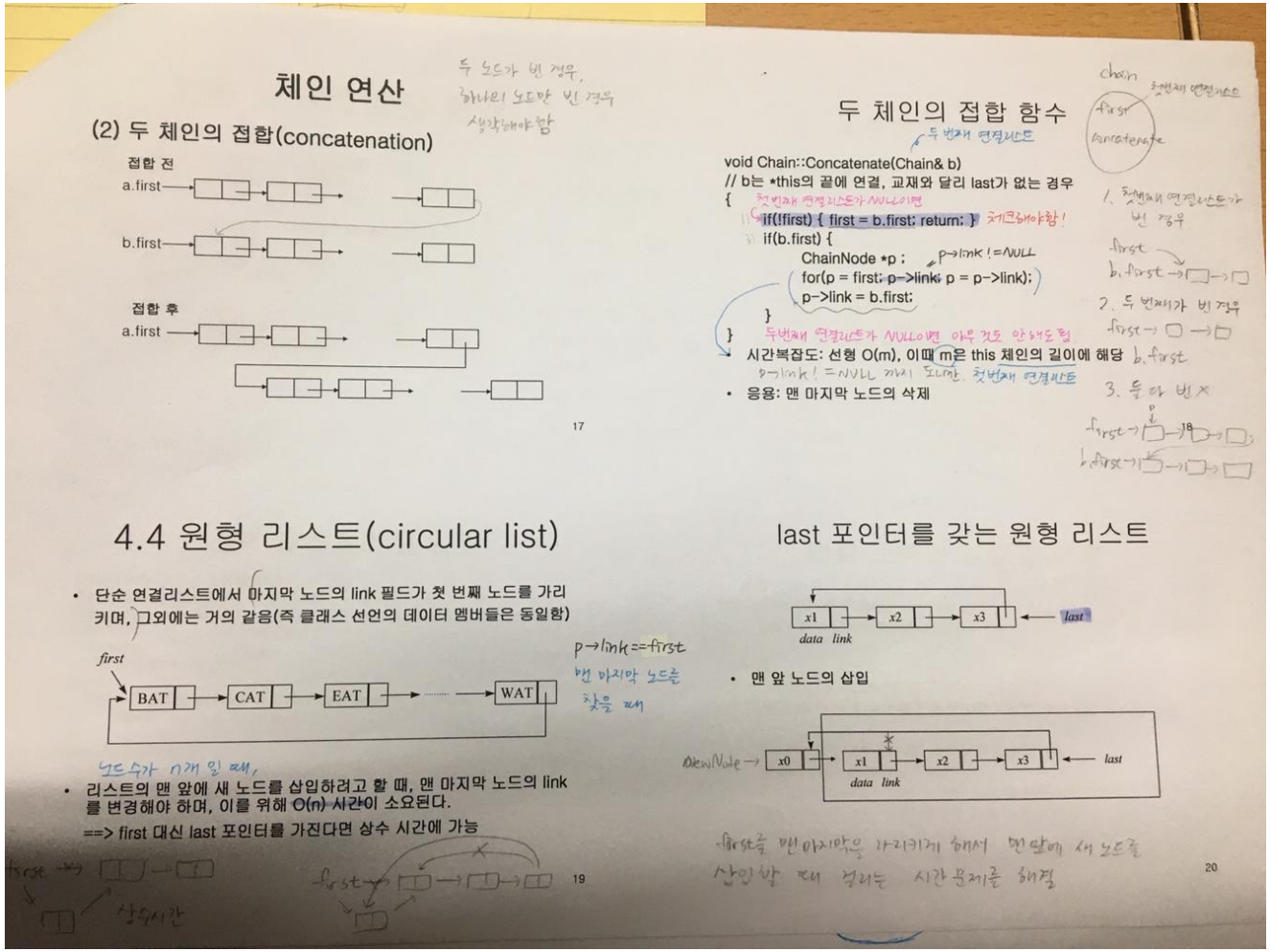
자료구조를 구조하자

2019.10.20

32153180 이상민

32162436 신창우 32163006 이건욱
32164420 조정민 32164959 허전진

학습 내용



체인 연산

(2) 두 체인의 접합(concatenation)

접합 전

접합 후

17

두 체인의 접합 함수

```
void Chain::Concatenate(Chain& b)
// b는 *this의 끝에 연결, 교재와 달리 last가 없는 경우
{
    if(!first) { first = b.first; return; }
    if(b.first) {
        ChainNode *p ;
        for(p = first; p->link; p = p->link);
        p->link = b.first;
    }
}
```

- 시간복잡도: 선형 O(m), 이때 m은 this 체인의 길이에 해당
- 응용: 맨 마지막 노드의 삭제

4.4 원형 리스트(circular list)

- 단순 연결리스트에서 마지막 노드의 link 필드가 첫 번째 노드를 가리키며, 그외에는 거의 같음(즉 클래스 선언의 데이터 멤버들은 동일함)

- 리스트의 맨 앞에 새 노드를 삽입하려고 할 때, 맨 마지막 노드의 link를 변경해야 하며, 이를 위해 O(n) 시간이 소요된다.

==> first 대신 last 포인터를 가진다면 상수 시간에 가능

19

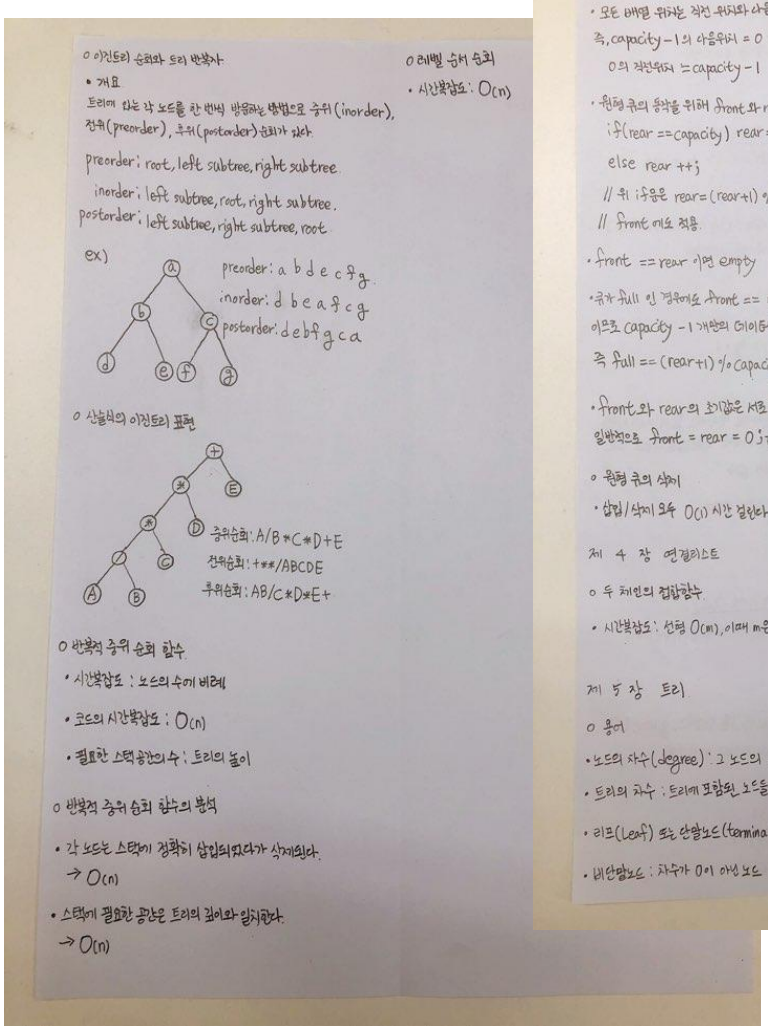
last 포인터를 갖는 원형 리스트

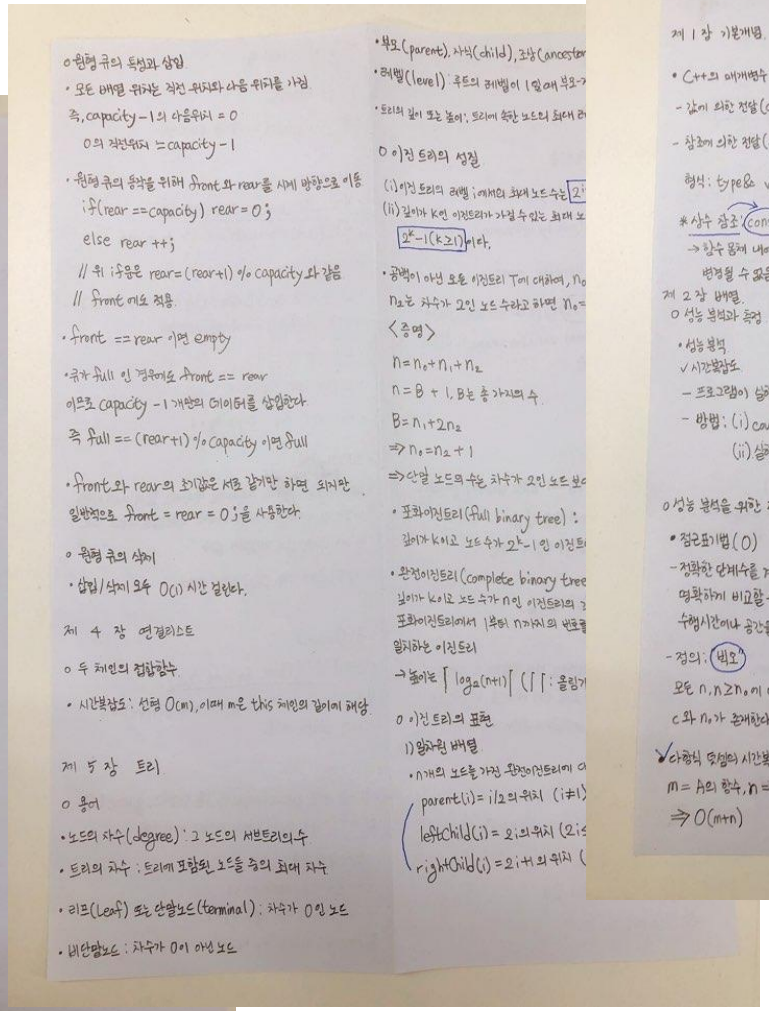
- 맨 앞 노드의 삽입

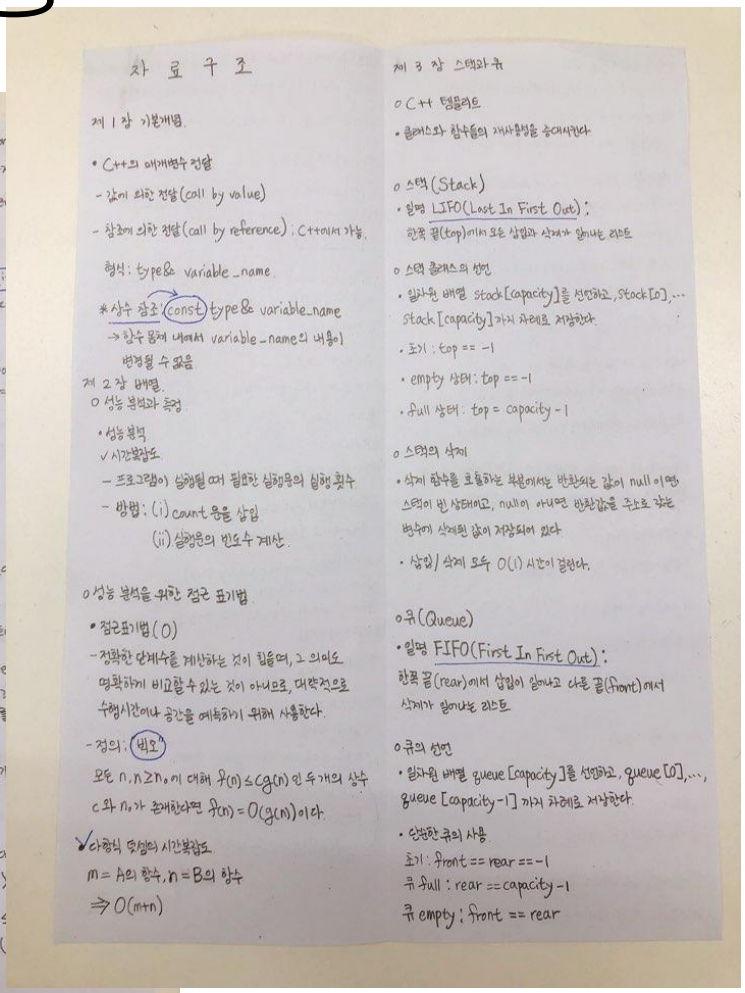
20

신창우

학습 내용

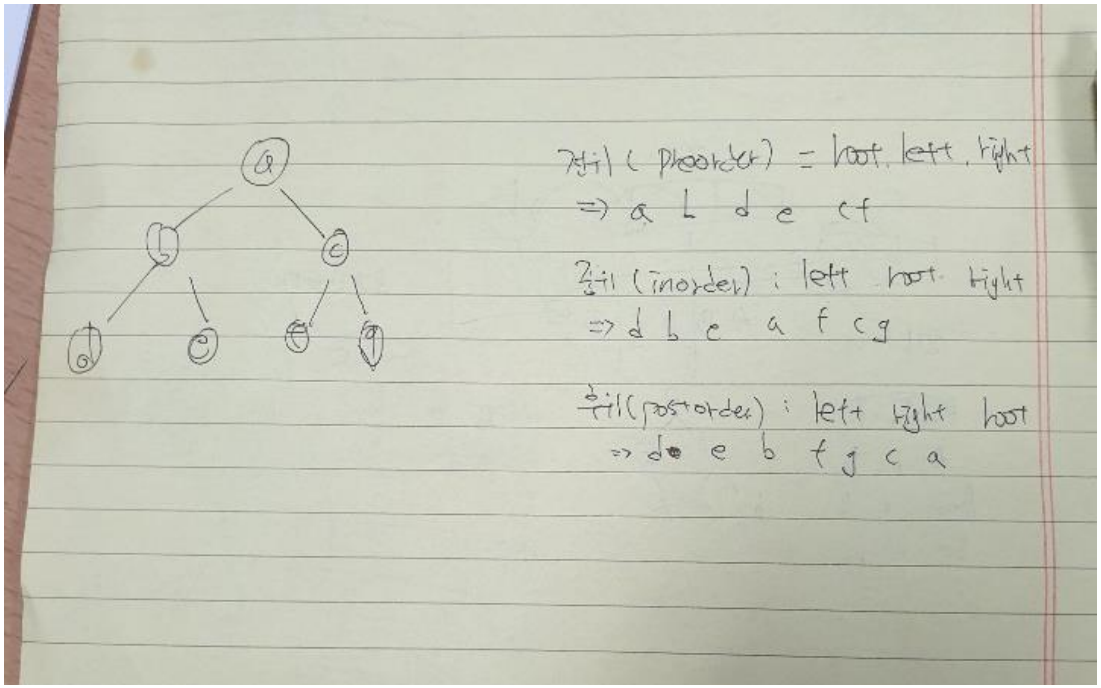

이진트리 순회와 트리 반복자
개요
preorder: root, left subtree, right subtree
inorder: left subtree, root, right subtree
postorder: left subtree, right subtree, root
ex)
preorder: a b d e c f g
inorder: d b e a f c g
postorder: d e b f g c a
산술식의 이진트리 표현
중위순회: A/B*C*D+E
전위순회: +**/ABCDE
후위순회: AB/C*D*E+
반복적 중위 순회 함수
시간복잡도 : 노드의 수에 비례
코드의 시간복잡도 : O(n)
필요한 스택 공간의 수 : 트리의 높이
반복적 중위 순회 함수의 특징
각 노드는 스택에 정확히 삽입되었다가 삭제된다.
→ O(n)
스택에 필요한 공간은 트리의 깊이와 일치한다.
→ O(n)
레벨 순서 순회
시간복잡도: O(n)


원형 큐의 특징과 삽입
모든 배열 위치는 절전 위치와 다음 위치를 가짐
즉, capacity-1의 다음위치 = 0
0의 직전위치 = capacity-1
원형 큐의 동작을 위해 front와 rear를 시계 방향으로 이동
if(rear == capacity) rear = 0;
else rear ++;
// 위 if문은 rear = (rear+1) % capacity와 같음
// front에도 적용
front == rear 이면 empty
큐가 full 인 경우에도 front == rear
이므로 capacity - 1 개만큼 데이터를 삽입한다
즉 full == (rear+1) % capacity 이면 full
front와 rear의 초기값은 서로 같기만 하면 되지만
일반적으로 front = rear = 0;을 사용한다.
원형 큐의 삽입
삽입/삭제 모두 O(1) 시간 걸린다.
제 4 장 연결리스트
두 체인의 접합함수
시간복잡도: 선형 O(m), 이때 m은 this 체인의 길이에 해당
제 5 장 트리
용어
노드의 차수(degree): 그 노드의 서브트리의 수
트리의 차수: 트리에 포함된 노드들 중의 최대 차수
리프(Leaf) 또는 단말노드(terminal): 차수가 0인 노드
비단말노드: 차수가 0이 아닌 노드
부모(parent), 자식(child), 조상(ancestor
레벨(level): 루트의 레벨이 1일때 부모-
트리의 길이 또는 높이: 트리에 속한 노드의 최대 레
이진 트리의 성질
(i) 이진 트리의 레벨 i에서의 최대 노드수는 2^
(ii) 길이가 k인 이진트리가 가질 수 있는 최대 노
2^k - 1 (k≥1) 이다.
공백이 아닌 모든 이진트리 T에 대하여, n0
n2는 차수가 2인 노드 수라고 하면 n0=
<증명>
n = n0 + n1 + n2
n = B + 1, B는 총 가지의 수
B = n1 + 2n2
⇒ n0 = n2 + 1
⇒ 단말 노드의 수는 차수가 2인 노드 보
포화이진트리(full binary tree):
길이가 k이고 노드 수가 2^k - 1인 이진트
완전이진트리(complete binary tree
길이가 k이고 노드 수가 n인 이진트리의
포화이진트리에서 1부터 n까지의 번호를
일치하는 이진트리
→ 높이는 ⌈log2(n+1)⌉ (⌈ ⌉ : 올림
이진 트리의 표현
1) 일차원 배열
n개의 노드를 가진 완전이진트리에
parent(i) = i/2의 위치 (i≠1)
leftChild(i) = 2i의 위치 (2i
rightChild(i) = 2i+1의 위치 (


자 료 구 조
제 1 장 기본개념
C++의 매개변수 전달
- 값에 의한 전달(call by value)
- 참조에 의한 전달(call by reference) : C++에서 가능
형식: type& variable_name
*상수 참조: const type& variable_name
→ 함수 몸체 내에서 variable_name의 내용이 변경될 수 없음
제 2 장 배열
성능 분석과 측정
성능 분석
✓시간복잡도
- 프로그램이 실행될 때 필요한 실행문의 실행 횟수
- 방법: (i) count 변수를 삽입
(ii) 실행문의 빈도수 계산
성능 분석을 위한 점근 표기법
점근표기법(O)
- 정확한 연산수를 계산하는 것이 힘들며, 그 의미도 명확하게 비교할 수 있는 것이 아니므로, 대략적으로 수행시간이나 공간을 예측하기 위해 사용한다.
- 정의: (빅오)
모든 n, n≥n0에 대해 f(n)≤cg(n)인 두 개의 상수 c와 n0가 존재한다면 f(n) = O(g(n))이다.
✓다항식 연산의 시간복잡도
m = A의 항수, n = B의 항수
⇒ O(m+n)
제 3 장 스택과 큐
C++ 템플릿으로
클래스와 함수들의 재사용성을 증대시킨다.
스택(Stack)
일명 LIFO(Last In First Out):
한쪽 끝(top)에서 모든 삽입과 삭제가 일어나는 리스트
스택 클래스의 선언
일차원 배열 stack[capacity]를 선언하고, stack[0], … stack[capacity]까지 차례로 저장한다.
초기 : top = -1
empty 상태 : top == -1
full 상태 : top = capacity - 1
스택의 삭제
삭제 함수를 호출하는 부분에서는 반환되는 값이 null이면 스택이 빈 상태이고, null이 아니면 반환값을 주소로 갖는 변수에 삭제된 값이 저장되어 있다
삽입/삭제 모두 O(1) 시간이 걸린다.
큐(Queue)
일명 FIFO(First In First Out):
한쪽 끝(rear)에서 삽입이 일어나고 다른 끝(front)에서 삭제가 일어나는 리스트
큐의 선언
일차원 배열 queue[capacity]를 선언하고, queue[0], … queue[capacity-1]까지 차례로 저장한다.
순환적 큐의 사용
초기 : front == rear == -1
큐 full : rear == capacity - 1
큐 empty : front == rear

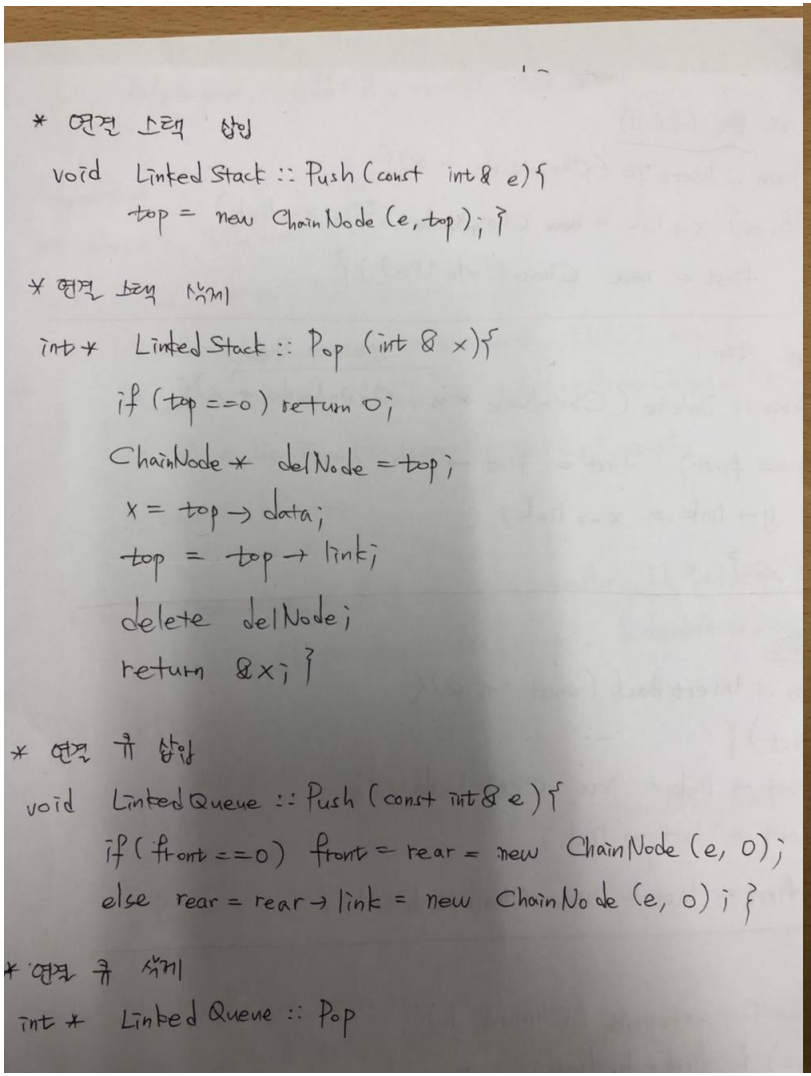
이건욱

학습 내용


a
b
c
d
e
f
g
전위 (preorder) = root, left, right
⇒ a b d e c f
중위 (inorder) : left root right
⇒ d b e a f c g
후위(postorder) : left right root
⇒ d e b f g c a

조정민

학습 내용



* 연결 스택 삽입

```
void LinkedStack::Push(const int& e){
    top = new ChainNode(e, top); }
```

* 연결 스택 삭제

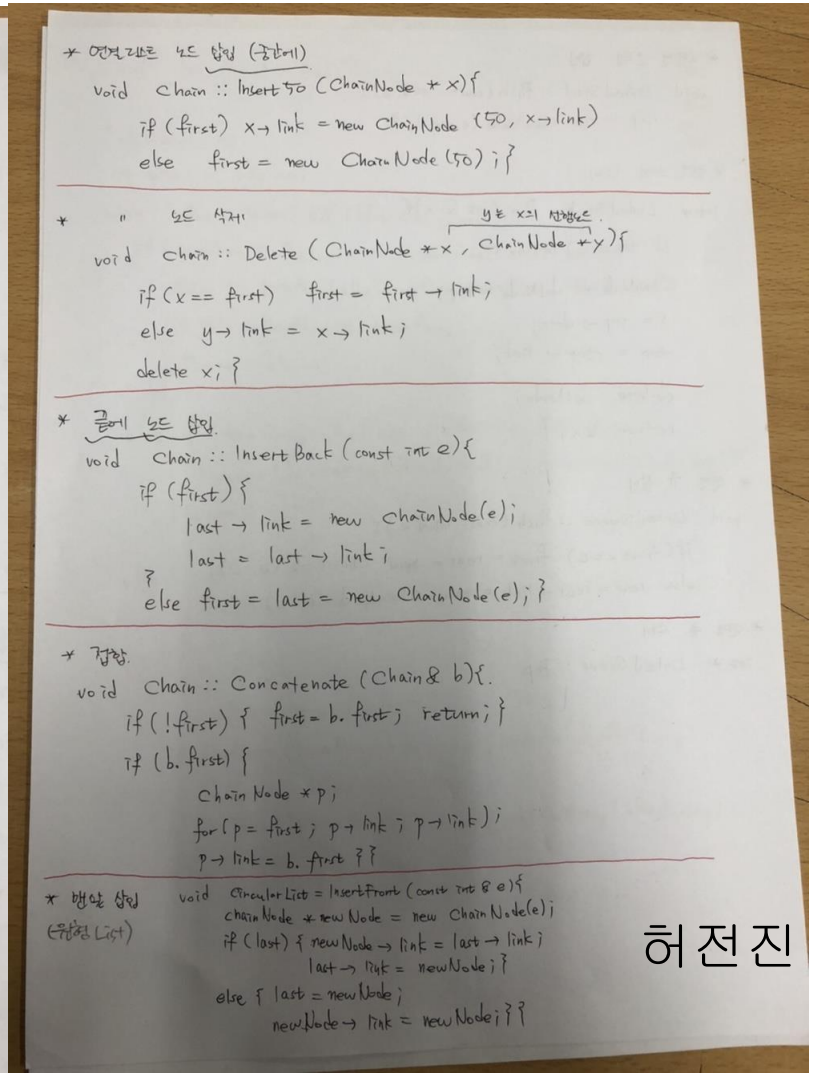
```
int* LinkedStack::Pop(int& x){
    if (top == 0) return 0;
    ChainNode* delNode = top;
    x = top->data;
    top = top->link;
    delete delNode;
    return &x; }
```

* 연결 큐 삽입

```
void LinkedQueue::Push(const int& e){
    if (front == 0) front = rear = new ChainNode(e, 0);
    else rear = rear->link = new ChainNode(e, 0); }
```

* 연결 큐 삭제

```
int* LinkedQueue::Pop
```



* 연결그래프 노드 삽입 (강의에)

```
void Chain::InsertTo(ChainNode* x){
    if (first) x->link = new ChainNode(50, x->link)
    else first = new ChainNode(50); }
```

* 〃 노드 삭제 (y는 x의 선행노드)

```
void Chain::Delete(ChainNode* x, ChainNode* y){
    if (x == first) first = first->link;
    else y->link = x->link;
    delete x; }
```

* 끝에 노드 삽입

```
void Chain::InsertBack(const int e){
    if (first){
        last->link = new ChainNode(e);
        last = last->link;
    }
    else first = last = new ChainNode(e); }
```

* 결합

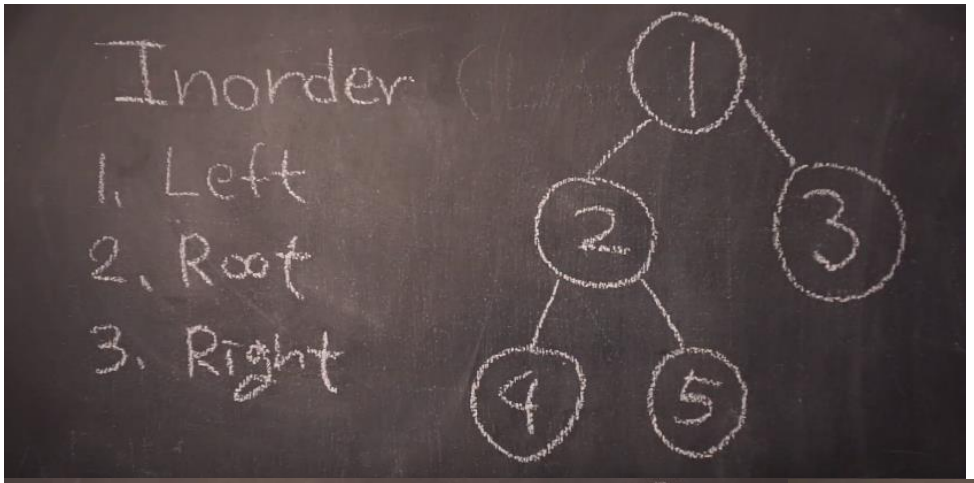
```
void Chain::Concatenate(Chain& b){
    if (!first) { first = b.first; return; }
    if (b.first) {
        ChainNode* p;
        for(p = first; p->link; p->link);
        p->link = b.first; } }
```

* 맹점 삽입 (순환형 List)

```
void CircularList::InsertFront(const int& e){
    ChainNode* newNode = new ChainNode(e);
    if (last) { newNode->link = last->link;
                last->link = newNode; }
    else { last = newNode;
           newNode->link = newNode; } }
```

허전진

참고자료



https://www.youtube.com/watch?v=QN1rZYX6QaA