

자료구조를 구조하자

2019.10.13

32153180 이상민

32162436 신창우 32163006 이건욱

32164420 조정민 32164959 허전진

학습 내용

```
#include <iostream>
using namespace std;

class ChainNode
{
private:
    int data;
    ChainNode* link;
public:
    ChainNode(int element = 0, ChainNode * next = 0) // 생성자
    {
        data = element; link = next;
    }
};

class LinkedQueue //큐 정의
{
private:
    ChainNode* front, * rear; //처음, 끝을 나타냄
public:
    LinkedQueue() { front = rear = 0; } // 생성자
    void Push(const int&); // 삽입 함수
    int Pop(); // 삭제 함수
    void Print(); // 출력 함수
};

//큐의 삽입
void LinkedQueue::Push(const int& e)
{
    if (front == 0) front = rear = new ChainNode(e, 0); //공백 큐
    else rear = rear->link = new ChainNode(e, 0);
    //노드를 삽입하고 rear를 수정함
}

//큐의 삭제
int LinkedQueue::Pop()
{
    if (front == 0) return 0; // 공백 큐이므로 null을 반환
    ChainNode* delNode = front;
    front = front->link; //앞의 노드를 제거
    delete delNode; //노드 반환
    return 1;
}

//큐 내용 보기
void LinkedQueue::Print()
{
    ChainNode* p;
    for (p = front; p; p = p->link) //p가 null일 때 까지
        cout << p->data << " "; //p의 data 출력
    cout << endl;
}
```

스택의 삭제

main
Stack<int> mystack(s);
int a=5, b;
mystack.push(a);

template <class T>
T* Stack<T>::Pop(T& x)
// stack에서 톱 원소를 삭제한다.
{
 if (top == -1) return 0; // 빈 상태이므로 null을 반환
 x = stack[top--]; // 삭제되는 값을 x에 저장
 return x; // x의 주소를 반환
}

삭제된 값은 x에 저장됨
x에 저장됨

main
Stack<int> mystack(s);
int a=5, b;
mystack.push(a);

만들기
Stack
0 1 2 3 4
a[5] a[0] a[1] a[2] a[3]
mystack

if (!mystack.isEmpty())
 b = mystack.pop();
 + if (1, mystack.isEmpty()) ①번 방법
 b를 사용
 pop해서 return
 되는 값이 있으면
 else 스택이 비어있음
 (= true일 때)

① main → isEmpty 확인
isEmpty: true일지
T Stack(T)::Pop
x = stack[top--];
return x.

3.3 큐(queue)

정해진 순서대로 애가 집 먼저 삭제, ex) 엘리베이터 집 먼저 도착 한사람만 나갈

- 일명 FIFO(First In First Out): 한쪽 끝(rear)에서 삽입이 일어나고 다른 끝(front)에서 삭제가 일어나는 리스트 > 가장 최근에 삽입된 원소의 index > 가장 최근에 삭제될 index

A	AB	ABC	ABCD	ABCDE	BCDE
↑	↑	↑	↑	↑	↑
삽입	삽입	삽입	삽입	삽입	삭제

그림 3.4 큐에서 원소의 삽입과 삭제

front rear capacity
0 1 2 3 4 5
A B C D E
front rear capacity
0 1 2 3 4 5
A B C D E
front rear capacity
0 1 2 3 4 5
A B C D E

같은 값이면 queue가 비어있음 나타냄

단순한 큐의 삽입과 삭제

예: rear = capacity - 1

front	rear	Q[0]	[1]	[2]	[3]	[4]	[5]	[6]	...	상태
-1	-1									초기(empty)
-1	0	J1								J1을 삽입
-1	1	J1	J2							J2를 삽입
-1	2	J1	J2	J3						J3을 삽입
0	2		J2	J3						삭제(J1)
0	3			J2	J3	J4				J4을 삽입
1	3				J3	J4				삭제(J2)

• 배열의 공간이 많이 남은 경우에도 queue full이 발생한다.

신창우

학습 내용

```

2-2 C LinkedStack
1 class ChainNode {
2     friend class LinkedStack; // 연결스택 클래스 프렌드 지정
3 private:
4     int data; // 노드의 데이터 필드
5     ChainNode *link; // 노드의 링크 필드
6 public:
7     ChainNode(int element = 0, ChainNode *next = 0) { // 노드 생성자
8         data = element; link = next;
9     }
10 };
11 class LinkedStack {
12 private:
13     ChainNode *top;
14 public:
15     LinkedStack() { top = 0; }; // 연결스택 생성자 함수
16     void Push(const int&); // 연결스택 삽입 함수
17     int *Pop(int&); // 연결스택 삭제 함수
18 };
19 void LinkedStack::Push(const int& e) {
20     top = new ChainNode(e, top); // 새 노드를 스택의 맨 위에 저장
21 }
22 // 스택 삭제
23 int* LinkedStack::Pop(int &x)
24 //스택에서 톱 노드를 삭제하고 값을 x에 저장한 후에 x의 주소를 반환
25 {
26     if (top == 0) return 0; //빈 스택이면 null을 반환
27     ChainNode* delNode = top;
28     x = top->data; //톱 노드의 data 필드를 x에 저장
29     top = top->link; //톱 위치를 다음 노드로 이동
30     delete delNode; //노드 삭제
31     return &x;
32 }
    
```

큐(Queue)

FIFO(First In First Out) : 한쪽 끝(rear)에서 삽입이 일어나고 다른 끝(front)에서 삭제가 일어나는 리스트

✓ 원형 큐의 특성과 삽입

- 원형 큐의 동작을 위해 front와 rear를 시계 방향으로 이동시킴

if (rear == capacity - 1) rear = 0;

else rear++;

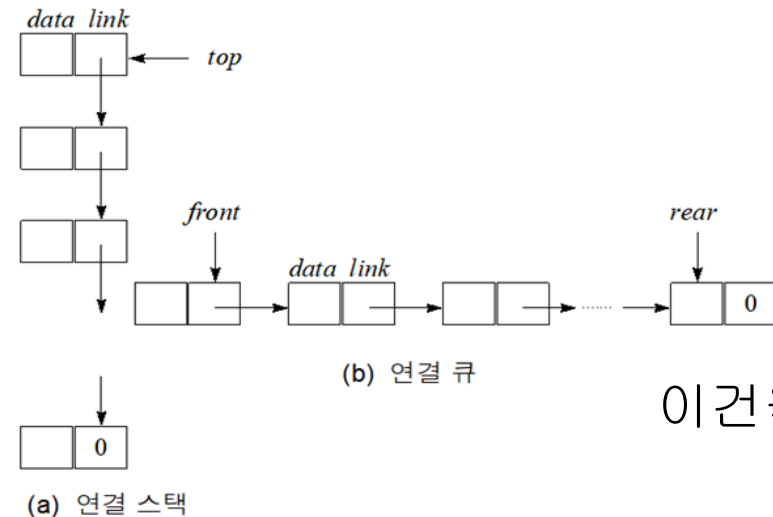
// 위 if 문은 rear = (rear + 1) % capacity와 같음

- front == rear 이면 empty

- 큐가 full인 경우에도 front == rear 이므로 capacity - 1개만의 데이터를 삽입한다. 즉 front == (rear + 1) % capacity 이면 full

- 연결 스택과 큐

> 스택과 큐를 연결 리스트로 구현하는 것



이건 목록

학습 내용

```
32164420 조정민.cpp 32164420 조정민.cpp
기타 파일 ChainNode 기타 파일 ChainNode

1  #include<iostream>
2  using namespace std;
3
4  class LinkedStack;      // 연결스택 전방선언
5  class LinkedQueue;      // 연결큐 전방선언
6  class ChainNode
7  {
8      friend LinkedStack;  // LinkedStack freind 지정
9      friend LinkedQueue;  // LinkedQueue freind 지정
10 private:
11     int data;             // data field
12     ChainNode* link;      // link field
13 public:
14     ChainNode(int element = 0, ChainNode* next = 0) // (
15     {
16         data = element;
17         link = next;
18     }
19 };
20 class LinkedStack
21 {
22 private:
23     ChainNode* top;
24 public:
25     LinkedStack()          // LinkedStack 생성자
26     {
27         top = 0;
28     }
29     void Push(const int&);  // LinkedStack 삽입함수
30     int* Pop(int&);        // LinkedStack 삭제함수
31     void Print();          // LinkedStack 출력함수
32 };

33 void LinkedStack::Push(const int& x)
34 {
35     top = new ChainNode(x, top);    // 새 노드를 top에 저장
36 }
37 int* LinkedStack::Pop(int& x)
38 {
39     if (top == 0)                   // 스택이 비어있을 경우
40         return 0;
41     ChainNode* temp = top;
42     x = temp->data;                  // top 노드의 data field를 x에 저장
43     top = top->link;                 // top을 다음 노드로 이동
44     delete temp;                    // 노드 삭제
45     return &x;                      // x의 주소 반환
46 }
47 void LinkedStack::Print()
48 {
49     ChainNode* p = top;
50     if (top == 0)                   // 스택이 비어있을 경우
51     {
52         cout << "비어있는 스택" << endl;
53         return;
54     }
55     else
56     {
57         cout << "스택(LIFO순) : ";
58         for (p; p->link; p = p->link) // p가 top부터 다음 노드로 이동
59         {
60             cout << p->data << ' '; // p의 data field 출력
61         }
62         cout << p->data << endl;    // 마지막 data field 출력
63     }
64 }
```

조정민

학습 내용

```
class LinkedStack {  
private:  
    ChainNode* top;  
public:  
    LinkedStack() { top = 0; }; //생성자  
    void Push(const int&); //스택 삽입 함수  
    int* Pop(int&); //스택 삭제 함수  
    void Print(); //스택 출력 함수  
};
```

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

```
LinkedStack::Pop(int& x)
```

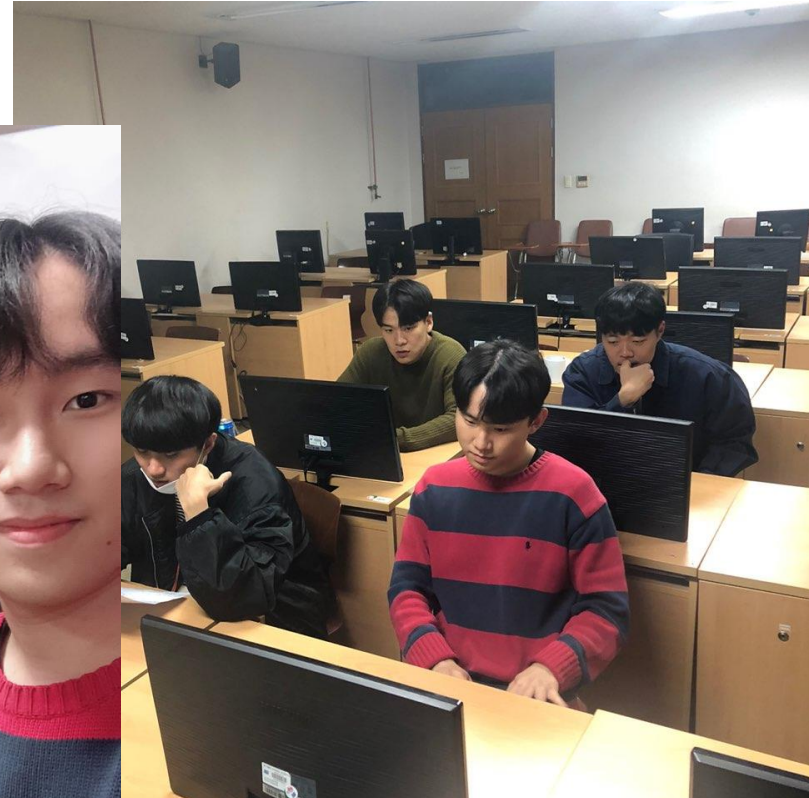
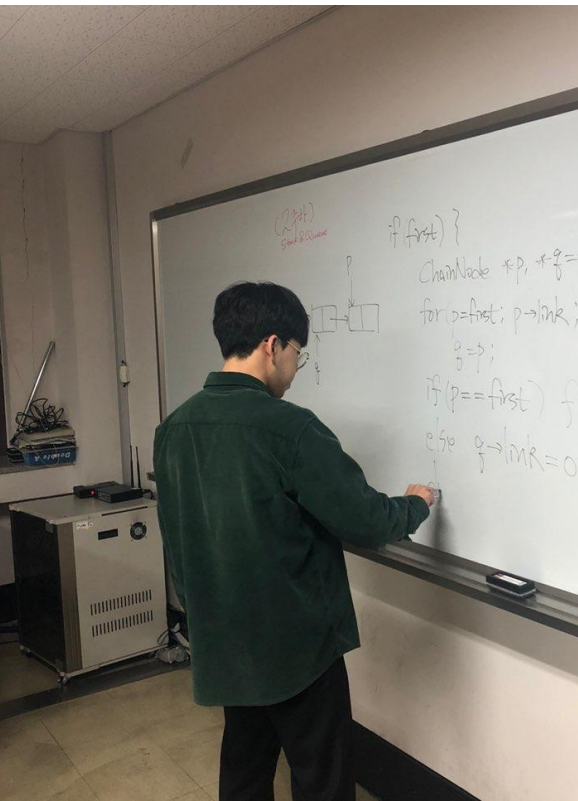
```
if (top == 0) return 0; //빈 스택이면 0 반환  
ChainNode* delNode = top;  
x = top->data; //톱 노드의 data필드 x에 저장  
top = top->link; //top 위치 -> 다음 노드  
delete delNode; //노드 삭제  
return &x; //x 주소 반환
```

```
LinkedStack::Print()
```

```
if (top == 0)  
    cout << "스택이 비었습니다." << endl;  
else { //스택이 있으면 LIFO순 출력  
    ChainNode* temp = top;  
  
    cout << "스택(LIFO순) : ";  
    while (temp != 0)  
    {  
        cout << temp->data << " ";  
        temp = temp->link;  
    }  
    }  
fgetc(stdin);  
//fflush(stdin);  
getchar();
```

허전진

활동 사진



참고자료

LIFO

Last In First Out

제일 마지막에 들어온 데이터가
제일 먼저 나간다

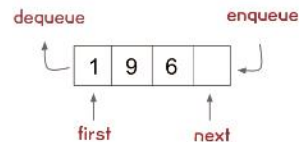
<https://www.youtube.com/watch?v=DsZHDmth6Pc>

<https://www.youtube.com/watch?v=BdsyG5yP1cQ>

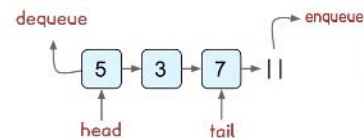
Gunny's Algorithm Together

구현방법

정적 어레이



동적 어레이(리스트)



중요함수들

- Enqueue : Queue 에 값을 집어넣는다.
- Dequeue : Queue 에서 값을 빼내온다.
- Size : Queue 의 크기를 확인한다.
- Empty : 큐가 비었는지 확인한다.