

Data Structure

Week 6
KyuDong SIM

1. 이번 주 실습 내용

- Stack (Linked List)
- Circular Queue (Array)

Stack

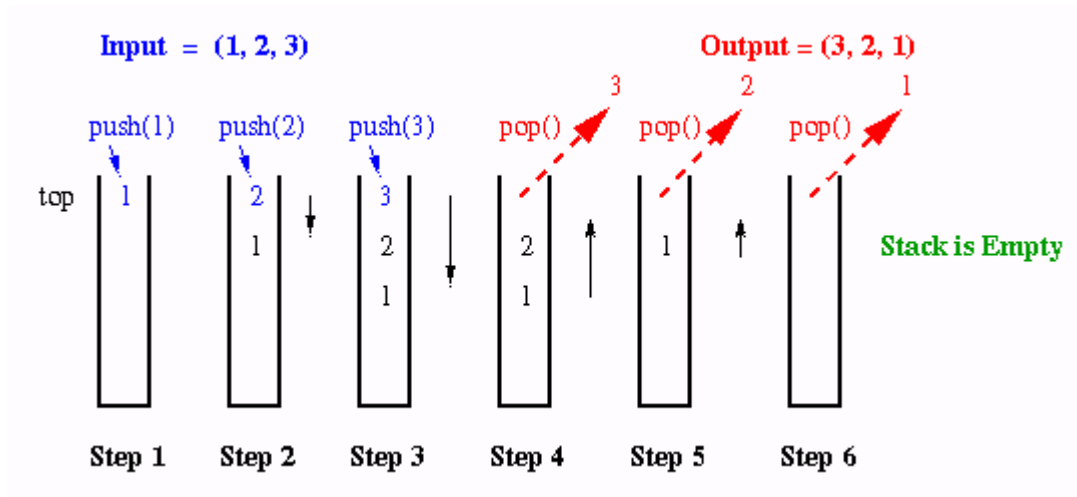
Stack

- 한쪽 끝에서만 데이터를 입/출력 할 수 있는 자료구조
 - 가장 마지막에 넣은 것이 먼저 나옴 : LIFO (Last In First Out)
- 기본 연산 단위
 - push : 데이터를 넣음
 - pop : 데이터를 꺼냄
- 형태 : Array 또는 Linked List로 구성
 - 본 실습에선 Linked List 방식 이용



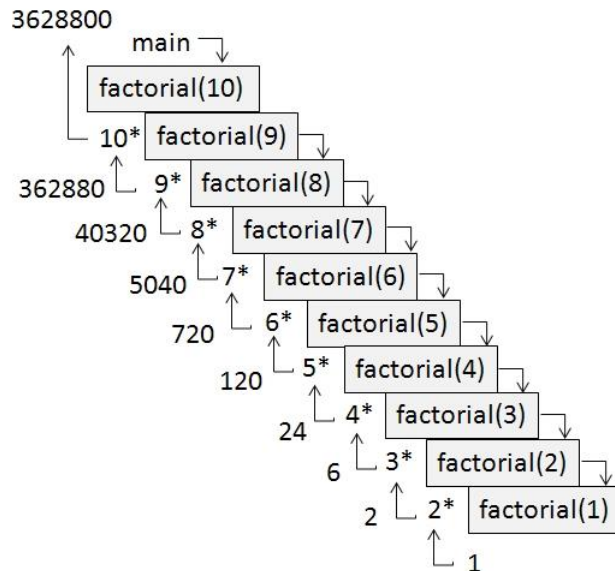
Stack Operation

- 데이터 넣기 : push
- 데이터 빼기 : pop
- 현재 스택 위치 : top



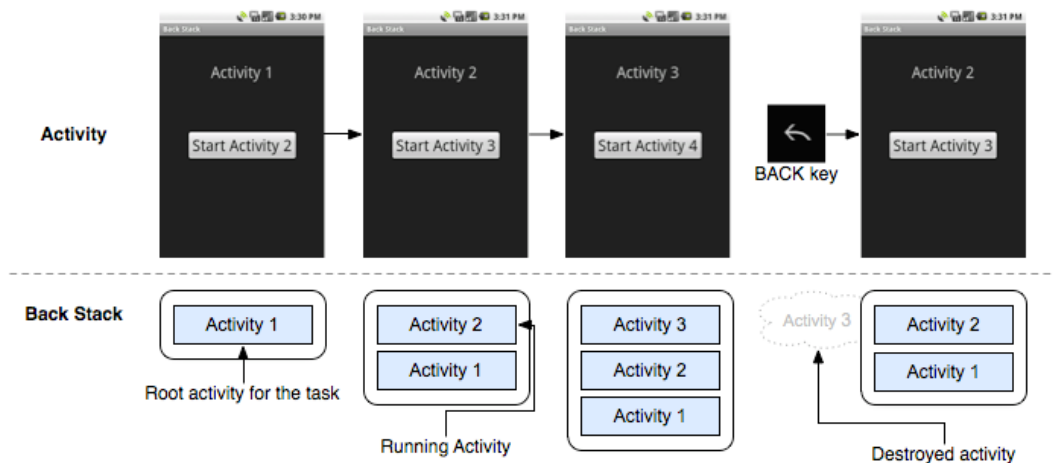
Stack Usage

- Function Call - Function Call Stack



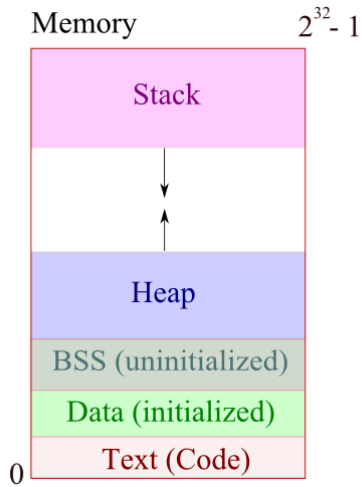
Stack Usage

- Android Graphics Interface - Activity Stack



Stack Usage

- Program's Memory Structure



Stack Usage

- Postfix Notation (후위 표기법)

중위 표기법	전위 표기법	후위 표기법
$2+3*4$	$+2*34$	$234*+$
$a*b+5$	$+5*ab$	$ab*5+$
$(1+2)+7$	$+7+12$	$12+7+$

수행 내용

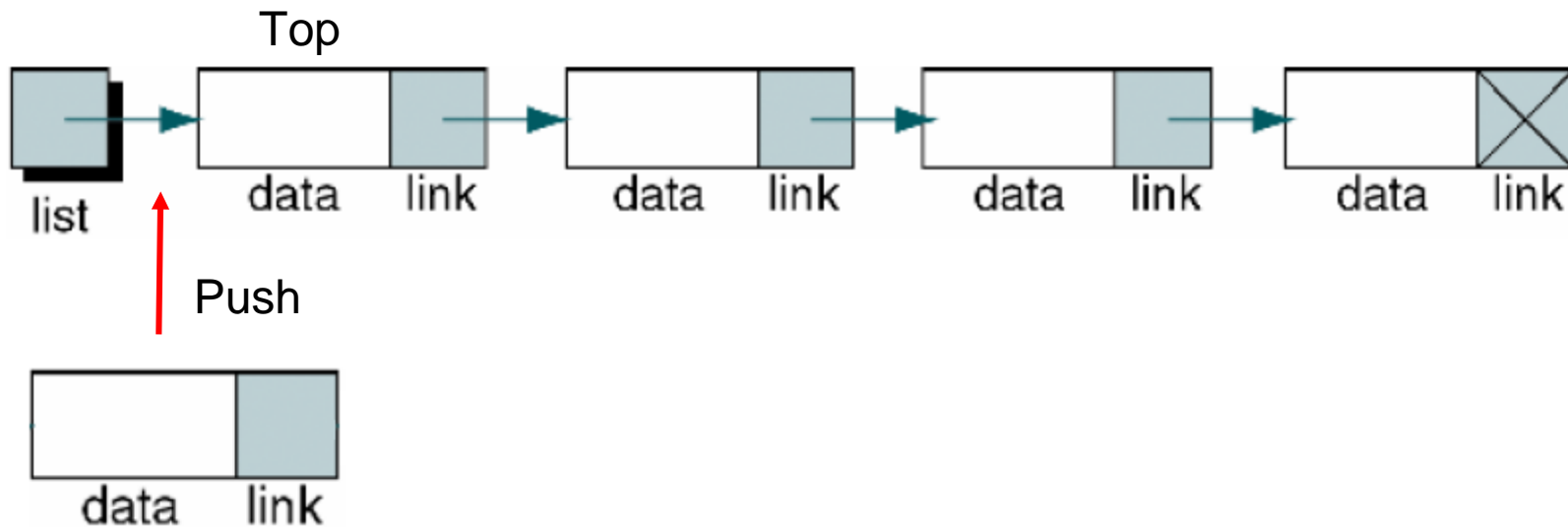
Push Stack에 데이터 입력

Pop Stack에서 데이터 제거

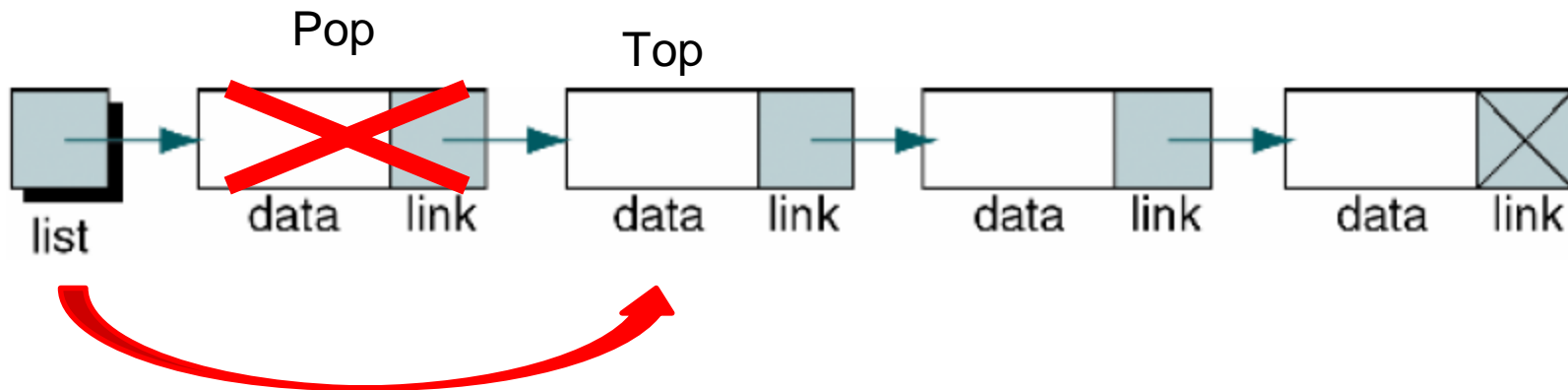
Top Stack의 최상단 데이터 출력

Quit Stack의 메모리 반환후 종료

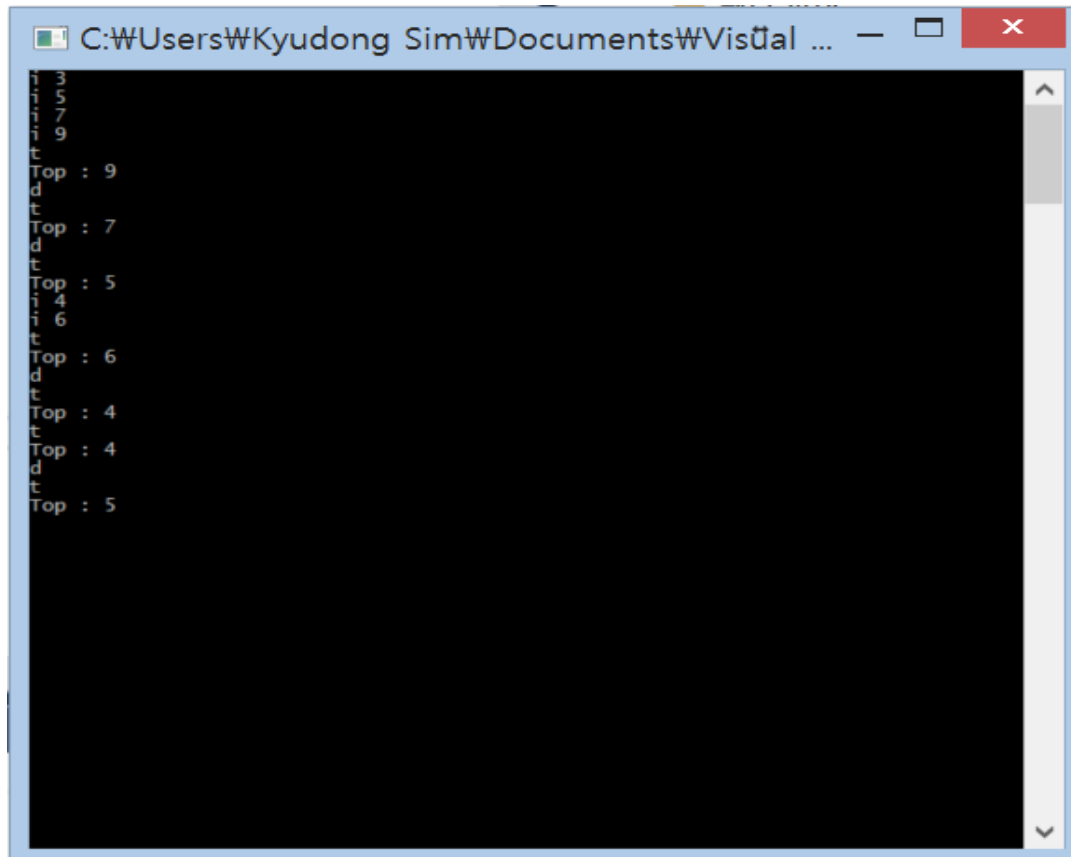
Linked list로 구현된 Stack



Linked list로 구현된 Stack



결과의 예)

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Users\Kyudong Sim\Documents\Visual ...'. The command prompt displays a sequence of numbers and 'Top' values. The numbers are 3, 5, 7, 9, 4, 6, 4, 4, 5. The 'Top' values are 9, 7, 5, 6, 4, 4, 5. The output is as follows:

```
3
5
7
9
Top : 9
4
6
Top : 7
4
4
Top : 5
6
4
4
Top : 6
4
4
Top : 4
4
4
Top : 4
4
4
Top : 5
```

Node 선언, IsEmpty 함수

```
struct Node;  
typedef struct Node *PtrToNode;  
typedef PtrToNode Stack;
```

```
struct Node {  
    int data;  
    PtrToNode Next;  
};
```

```
int IsEmpty(Stack S) {  
    /*비어있는지 확인, 비어있으면 1, 아니면 0을 반환*/  
}
```

Stack의 구조체

Pop, Push

```
void Pop(Stack S) {  
    PtrToNode FirstCell;  
  
    if (IsEmpty(S))  
        printf("Empty Stack");  
    else {  
        FirstCell = S->Next;  
        S->Next = S->Next->Next;  
        free(FirstCell);  
    }  
}
```

```
void Push(int data, Stack S) {  
    PtrToNode TmpCell;  
  
    TmpCell = (PtrToNode)malloc(sizeof(struct Node));  
  
    if (TmpCell == NULL)  
        printf("Out of Space");  
    else  
    {  
        TmpCell->data = data;  
        TmpCell->Next = S->Next;  
        S->Next = TmpCell;  
    }  
}
```

Pop은 데이터가 없으면 Pop실패,
있을경우 최상위 Node 1개 제거

Push는 Node를 하나 생성해서
Node생성에 성공하면 Stack의
최상단에 삽입

Delete

```
int Top(Stack S) {  
    if (!IsEmpty(S))  
        return S->Next->data;  
    printf("Empty Stack");  
    return 0;  
}  
  
void MakeEmpty(Stack S) {  
    /*Stack의 모든 Node의 메모리 반환*/  
}
```

```
Stack CreateStack(void) {  
    Stack S;  
  
    S = (Stack )malloc(sizeof(struct Node));  
    if (S == NULL)  
        printf("Out of Space!!!");  
    S->Next = NULL;  
  
    return S;  
}
```

Top은 Stack의 최상위 Node값을
출력

CreateStack은 Stack의 첫 Node를
생성 (Data가 들어있지 않은 Node)

Stack 생성 및 반복수행

```
int main(void) {  
    Stack S = CreateStack();  
  
    int data;  
    char command;  
    int condition = 1;  
    while (condition) {  
        scanf("%c", &command);  
        switch (command) {  
            case 'i':  
                /*Push*/  
                break;  
            case 't':  
                /*Top의 data 출력*/  
                break;  
            case 'd':  
                /*Pop*/  
                break;  
            case 'q':  
                /*메모리 반환(Make Empty) 후 종료*/  
                break;  
        }  
    }  
}
```

Circular Queue

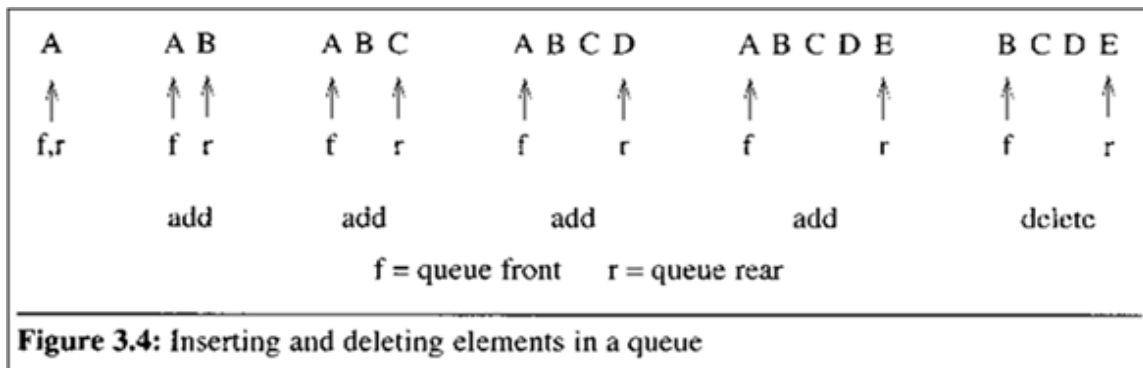
Queue

- 한쪽 끝에서만 데이터를 입/출력 할 수 있는 자료구조
 - 가장 먼저 넣은 것이 먼저 나옴 : FIFO (First In First Out)
- 기본 연산 단위
 - enqueue (add queue) : 데이터를 넣음
 - dequeue (delete queue) : 데이터를 꺼냄
- 형태 : Array 또는 Linked List로 구성



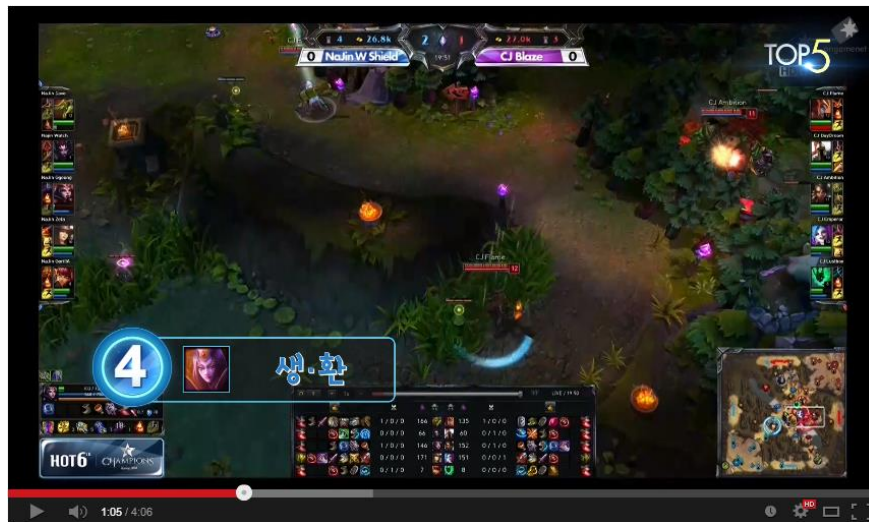
Queue Operation

- 데이터 넣기 : enqueue (add queue)
- 데이터 빼기 : dequeue (delete queue)
- 현재 위치
 - front : 가장 앞쪽 (가장 먼저 넣은 데이터 위치)
 - rear : 가장 뒤쪽 (가장 마지막에 넣은 데이터 위치)



Queue Usage

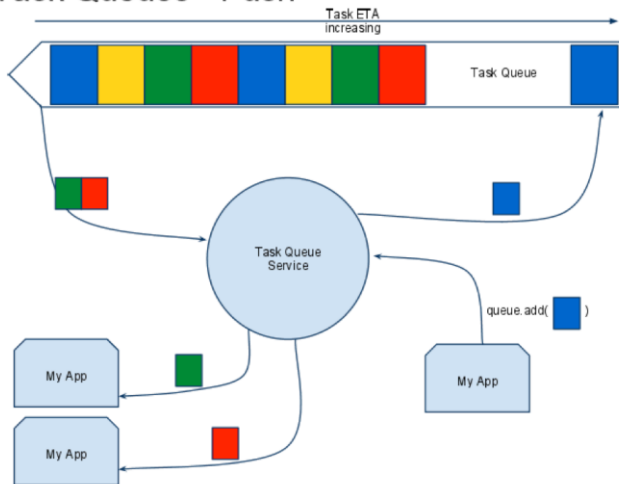
VOD Player - Buffering



Queue Usage

Process Queue - Scheduling

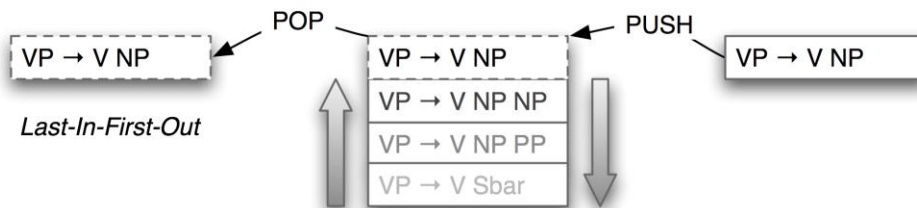
Task Queues - Push



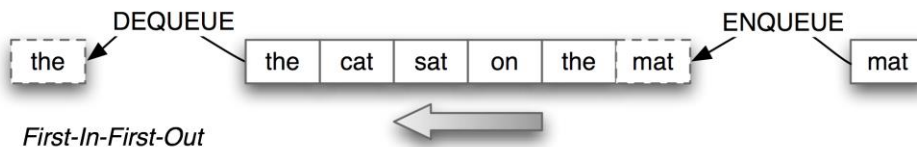
Stack vs Queue

- Stack
 - Backtracking
- Queue
 - 순차 처리
 - 서로 속도 차이가 있을 때
 - 버퍼 역할
 - 비동기 처리

STACK



QUEUE



Circular Queue



enqueue(1)



enqueue(3)



dequeue()



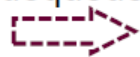
dequeue()



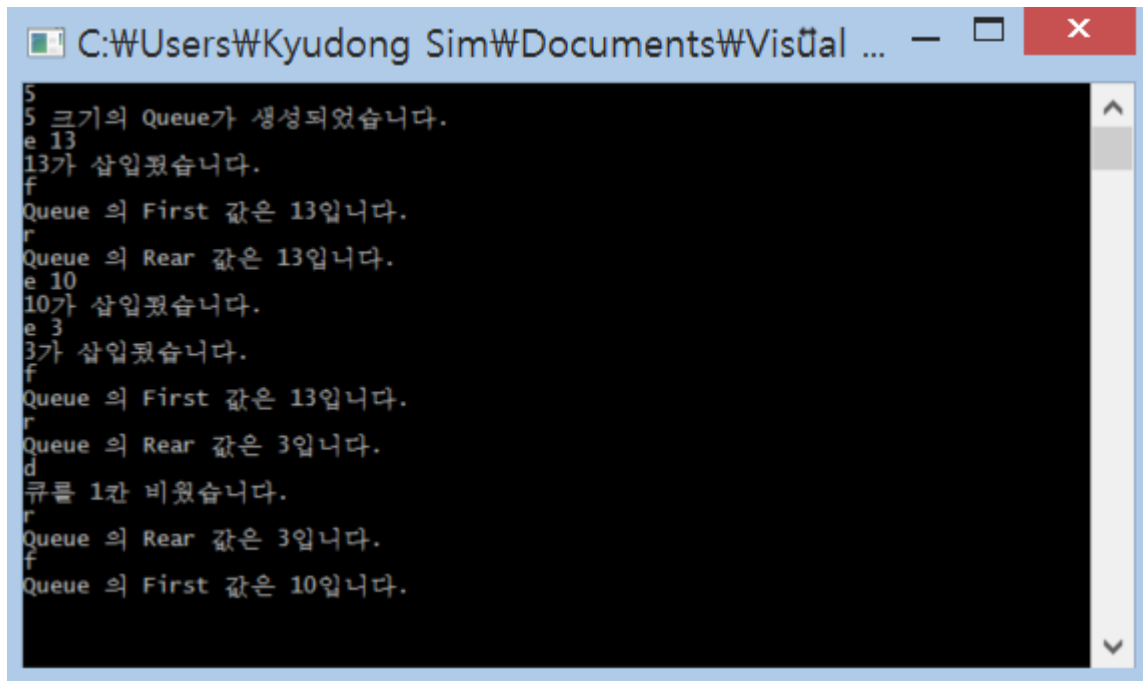
dequeue()



dequeue()



Queue 실행 예



```
S 크기의 Queue가 생성되었습니다.
e 13
13가 삽입되었습니다.
f
Queue 의 First 값은 13입니다.
r
Queue 의 Rear 값은 13입니다.
e 10
10가 삽입되었습니다.
e 3
3가 삽입되었습니다.
f
Queue 의 First 값은 13입니다.
r
Queue 의 Rear 값은 3입니다.
d
큐를 1칸 비웠습니다.
r
Queue 의 Rear 값은 3입니다.
f
Queue 의 First 값은 10입니다.
```

**시작: Queue의 크기
입력**

e x : Queue에 x 삽입

D : 데이터 1개 제거

F : Queue의 첫 데이터
출력

r : Queue의 마지막
데이터 출력

Data type

```
struct QueueRecord
{
    int Capacity;
    int Front;
    int Rear;
    int Size;
    int *Array;
};

typedef struct QueueRecord *Queue;
```

Capacity : 큐의 최대 용량입니다.

Front : 큐의 첫 번째 위치입니다.

Rear : 큐의 마지막 위치입니다.

Size : 현재 큐의 크기입니다. Capacity와 같아지면 더 enqueueer 될 수 없습니다.

Array : 큐의 데이터 행렬입니다.

MakeEmpty, Enqueue

```
void MakeEmpty(Queue Q)
{
    Q->Size = 0;
    Q->Front = 1;
    Q->Rear = 0;
}

void Enqueue(int X, Queue Q)
{
    if(IsFull(Q))
        printf("Full Queue");
    else
    {
        Q->Size++;
        Q->Rear = Succ(Q->Rear, Q);
        Q->Array[Q->Rear] = X;
    }
}
```

MakeEmpty : 큐의 초기설정입니다.
크기는 0이고 Front가 Rear 의 뒤에
있습니다.

Enqueue : 큐에 데이터 "X"를
삽입합니다.

IsEmpty, Succ

```
int IsEmpty(Queue Q)
{
    return Q->Size == 0;
}

static int Succ(int value, Queue Q)
{
    if(++value == Q->Capacity)
        value = 0;
    return value;
}
```

IsEmpty : 큐가 비어있는지 확인합니다.

Succ : Circular Queue의 다음 위치를 반환합니다.

CreateQueue, Printfront, PrintRear

```
Queue CreateQueue(int Capacity)
{
```

```
void Printfront(Queue Q)
{
}
```

```
void PrintRear(Queue Q)
{
}
```

CreateQueue : 큐와 데이터 행렬의
메모리 할당, 초기화를 하여 큐를
반환합니다.

Printfront : 첫 값을 출력합니다.

PrintRear : 마지막 값을 출력합니다.

IsFull, Dequeue

```
int IsFull(Queue Q)
{

}

void Dequeue(Queue Q)
{

}
```

IsFull : Capacity 와 비교하여 큐가
가득찼는지 확인합니다.

Dequeue : 큐가 비어있는지 확인하고
데이터 하나를 지웁니다.

Queue 생성 및 반복수행

```
int main(void)
{
    int number;
    scanf("%d", &number); //queue 크기 입력
    Queue Q = CreateQueue(number);
    char command, condition;

    while (condition)
    {
        scanf("%c", &command); //명령어 스캔
        switch (command)
        {
            case 'e':                //Queue 삽입
                /*Enqueue*/
                break;
            case 'd':                //Queue 제거
                /*Dequeue*/
                break;
            case 'f':                //Queue Front 출력
                /*printfront*/
                break;
            case 'r':                //Queue Rear 출력
                /*Printrear*/
                break;
            case 'q':                //Queue 종료
                /*quit*/
                break;
        }
    }
}
```

제출 및 알림

수업 중 확인 or 메일제출 (이름, 학번, 소스코드)

메일 제출 :

주소 : (89kdsim@naver.com)

기한 : ~2016-04-13