Data Structure #4

Stack, Queue

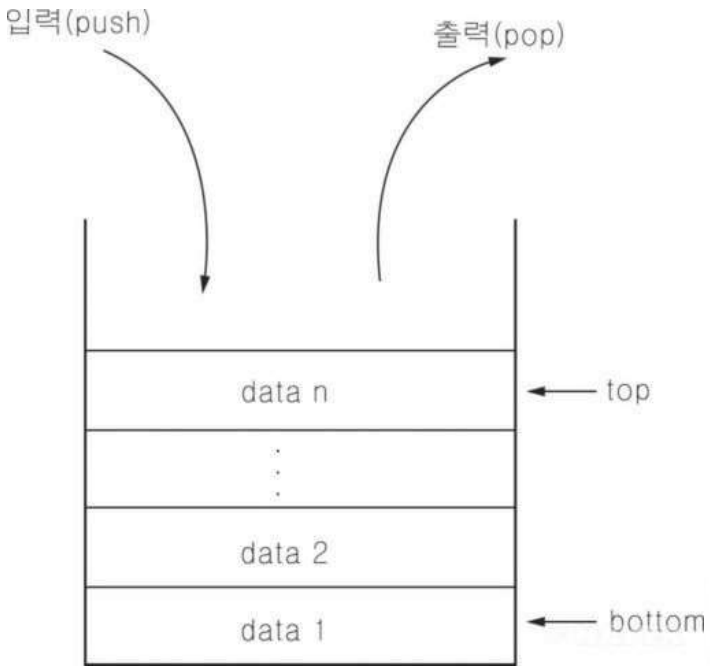
2020년 1학기

Intro.

- **실습 주제**
  - Stack
  - Queue

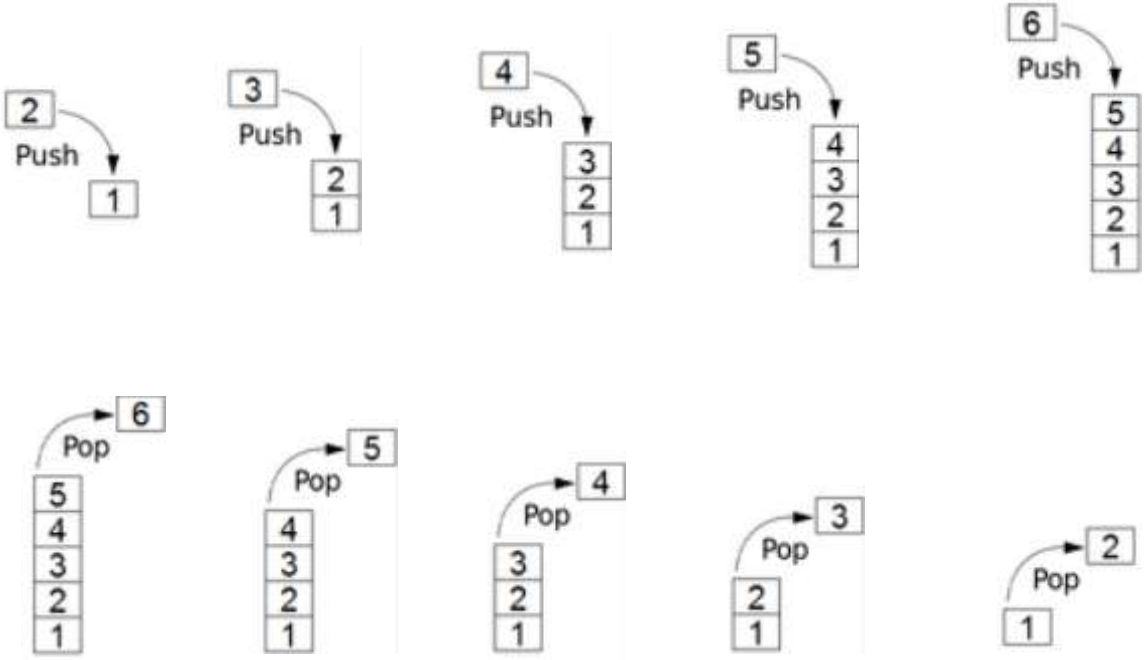
스택

- **스택이란?**
  - 자료의 삽입과 삭제가 한쪽 끝에서만 일어나는 선형 자료 구조
  - 가장 최근에 입력된 데이터가 출력되는 LIFO (Last-In First-Out) 구조



스택

- 스택


2
Push
1
3
Push
2
1
4
Push
3
2
1
5
Push
4
3
2
1
6
Push
5
4
3
2
1
6
Pop
5
4
3
2
1
5
Pop
4
3
2
1
4
Pop
3
2
1
3
Pop
2
1
2
Pop
1

스택

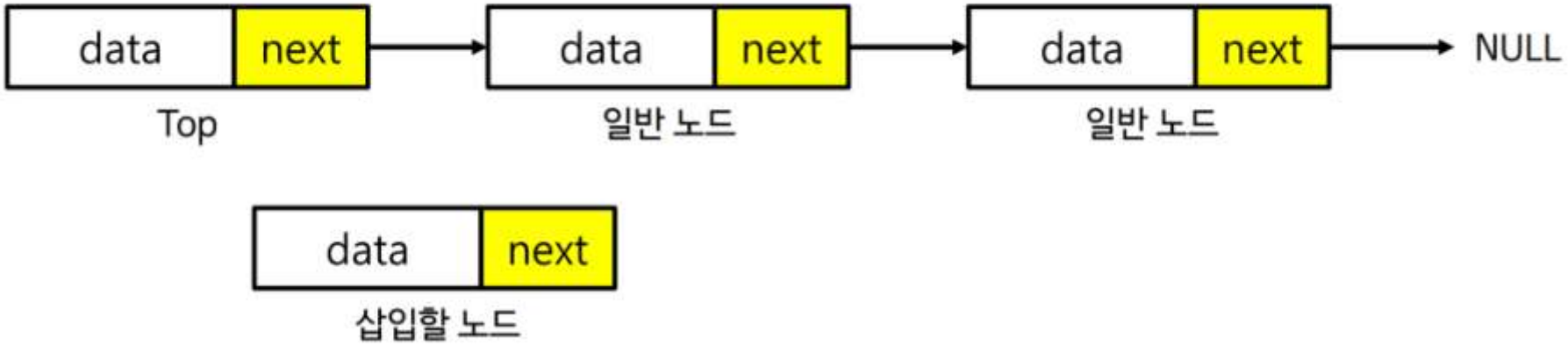
- **스택의 구현 (연결리스트)**
  - 스택의 구조체

```
typedef struct Node {
        int data;
        struct Node *next;
} Node;

typedef struct top {
        Node *top;
} Stack;
```

스택

- **스택의 구현 (연결리스트)**
  - PUSH



스택

- **스택의 구현 (연결리스트)**
  - PUSH

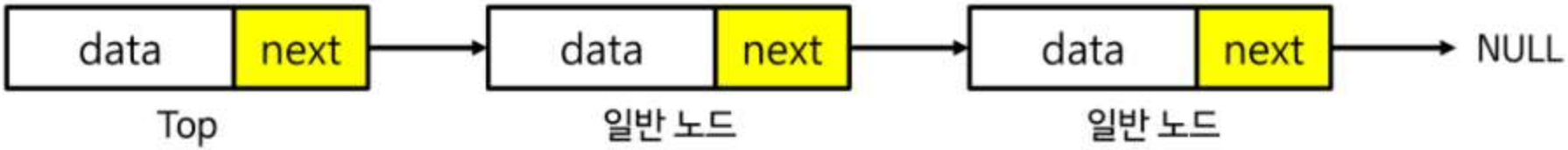
```
void push(Stack *stack, int data) {

        Node *node = (Node*)malloc(sizeof(Node));
        node->data = data;
        node->next = stack->top;
        stack->top = node;

}
```

스택

- 스택의 구현 (연결리스트)
  - POP



스택

- 스택의 구현 (연결리스트)
  - POP

```
int pop(Stack *stack) {

        if(stack->top == NULL) {
                printf("Stack Underflow");
                return -1;
        }
        Node *node = stack->top;
        int data = node->data;
        stack->top = node->next;
        free(node);
        return data;
}
```

스택

- **스택의 구현 (연결리스트)**
  - display

```
void display(Stack *stack) {
        Node *cur = stack->top;
        while(cur != NULL) {
                printf("%d\n",cur->data);
                cur = cur->next;
        }

}
```

스택

- **스택의 구현 (연결리스트)**
  - main 함수

```
int main(void) {
        Stack s;
        s.top = NULL;
        push(&s,7);
        push(&s,6);
        push(&s,5);
        push(&s,4);
        push(&s,3);
        pop(&s);
        display(&s);
        return 0;
}
```

```
4
5
6
7
```

스택 (개인 실습 #1)

- **스택의 구현 시 필요한 요소 (배열)**
  - 스택이 공백 상태인지 검사
  - 스택이 포화 상태인지 검사
  - 삽입 연산 push
  - 삭제 연산 pop
  - 스택의 최상단 요소를 확인하는 peek

스택

- **스택의 구현**
  - 스택의 구조체

```
#define MAX_STACK_SIZE  100
typedef int element;
typedef struct {
        element stack[MAX_STACK_SIZE];
        int top;
} StackType;
```

  - 스택 초기화 함수

```
void init(StackType *s)
{
        s->top = -1;
}
```

스택

- 스택의 구현

PUSH(7) - PUSH(5) - PUSH(4)

스택

- **스택의 구현**
  - 삽입 함수 push

```
void push(StackType *s, element item)
{
        if ( is_full(s) ) {
                fprintf ( stderr, “stack full\n” );
                return;
        }
        else  //top <- top+1
              //stack[top] <- item
```

스택

- **스택의 구현**
  - 삭제 함수 pop

```
element pop( StackType *s )
{
        if( is_empty(s) ) {
                fprintf( stderr, “stack empty\n” );
                exit(1);
        }
        else   // item<-stack[top]
}               // top <- top-1
                //return item;
```

스택

- **스택의 구현**
  - 공백 상태 검출 함수

```
Int is_empty(StackType *s)
{
        //if top = -1
        //then return TRUE
}       //else return FALSE
```

  - 표화 상태 검출 함수

```
Int is_full (StackType *s)
{
      //if top = MAX_STACK_SIZE -1
      //then return TRUE
}     //else return FALSE
```

스택

- **스택의 구현**
  - peek 함수

```
element peek( StackType *s )
{
        if( is_empty(s) ) {
                fprintf( stderr, “stack empty\n” );
                exit(1);
        }
        else
                //return item<-stack[top]
}
```

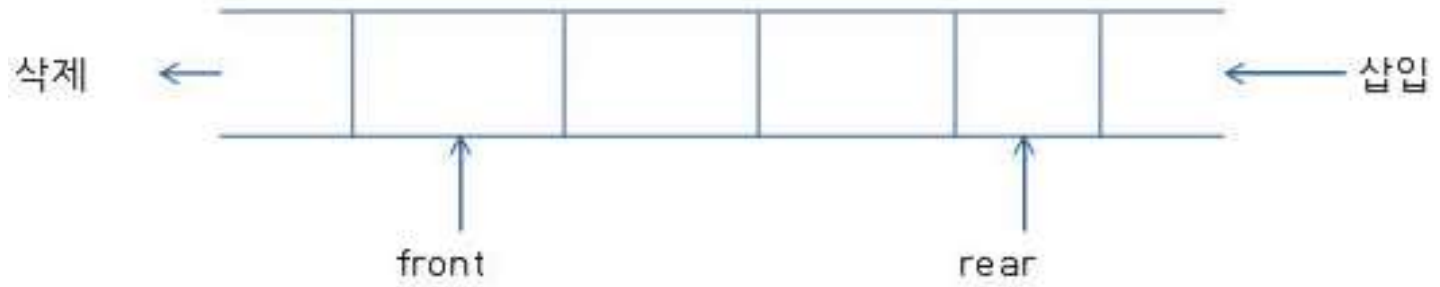
- 스택의 구현
  - main 함수

```
void main()
{
        StackType s;

        init(&s);
        push( &s, 1 );
        push( &s, 2 );
        push( &s, 3 );
        printf( “%d\n”, pop(&s) );
        printf( “%d\n”, peek(&s) );
        printf( “%d\n”, pop(&s) );
        printf( “%d\n”, is_empty(&s) );
}
```

큐

- **큐란?**
  - 한쪽 끝에서는 원소들이 삭제되고 반대쪽 끝에서는 원소들이 삽입만 가능한 선형 자료구조
  - 가장 먼저 입력된 데이터가 먼저 출력되는 FIFO (First-In First-Out) 구조

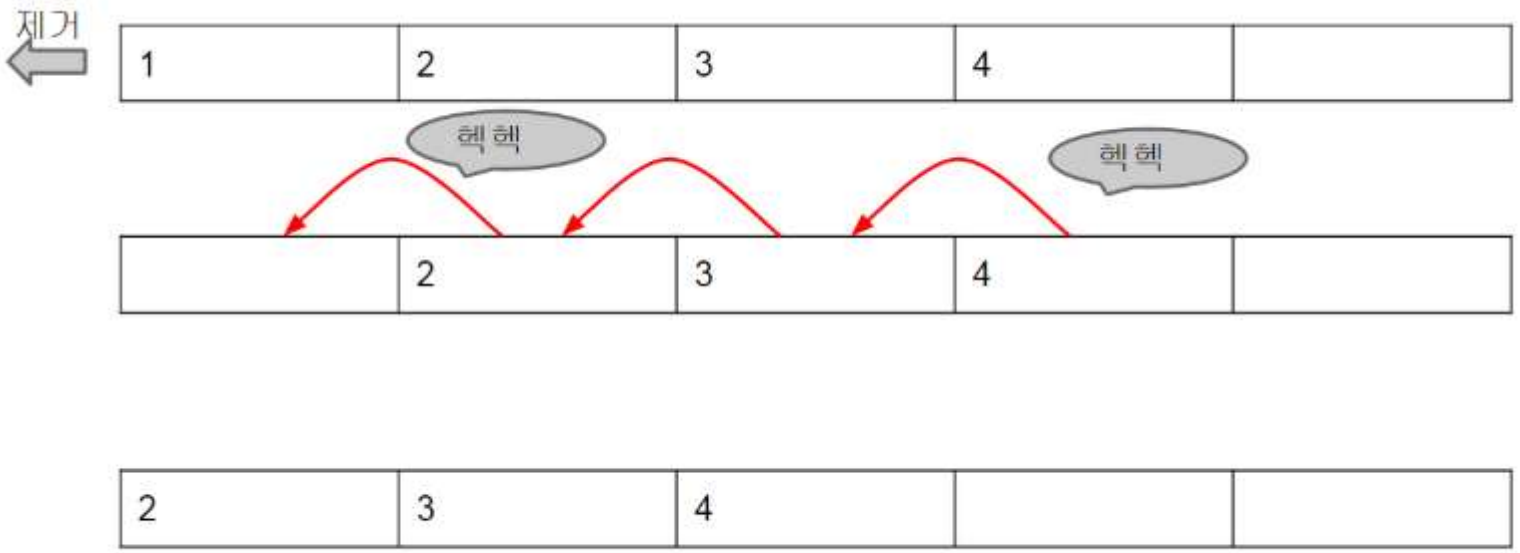


front : 저장된 원소 중에서 첫 번째 원소

rear : 저장된 원소 중에서마지막 원소

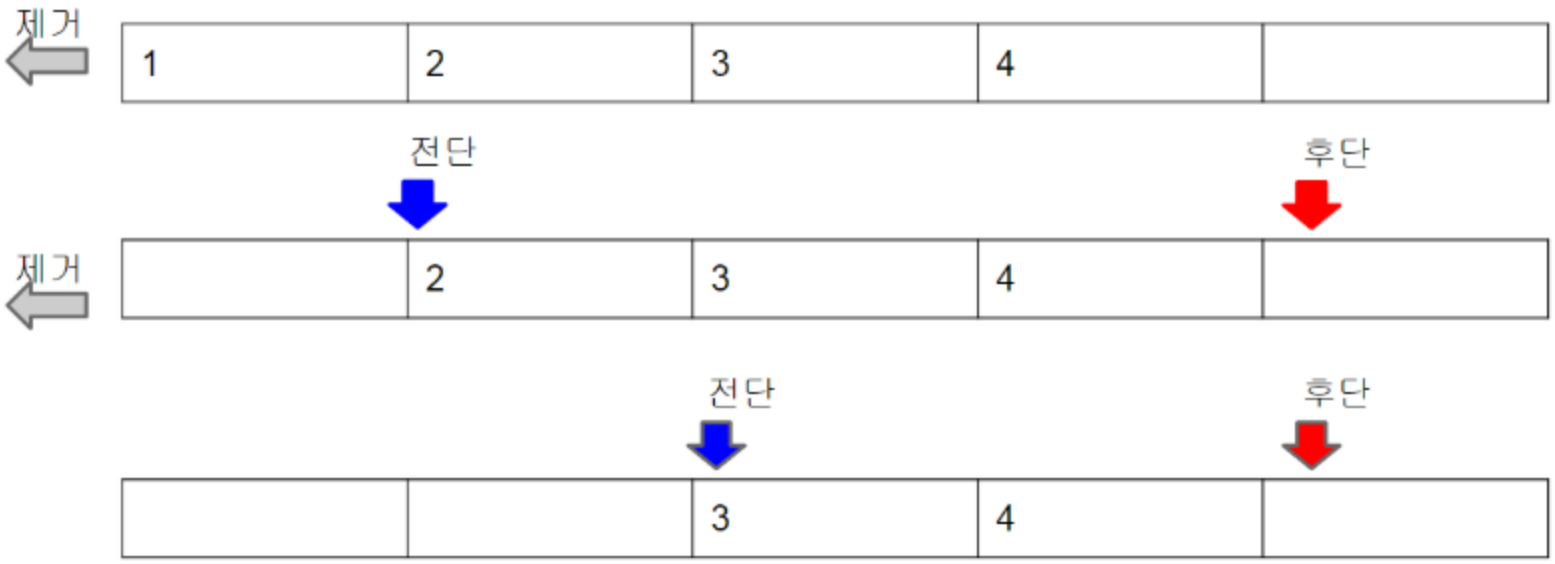
큐

- **선형 큐**



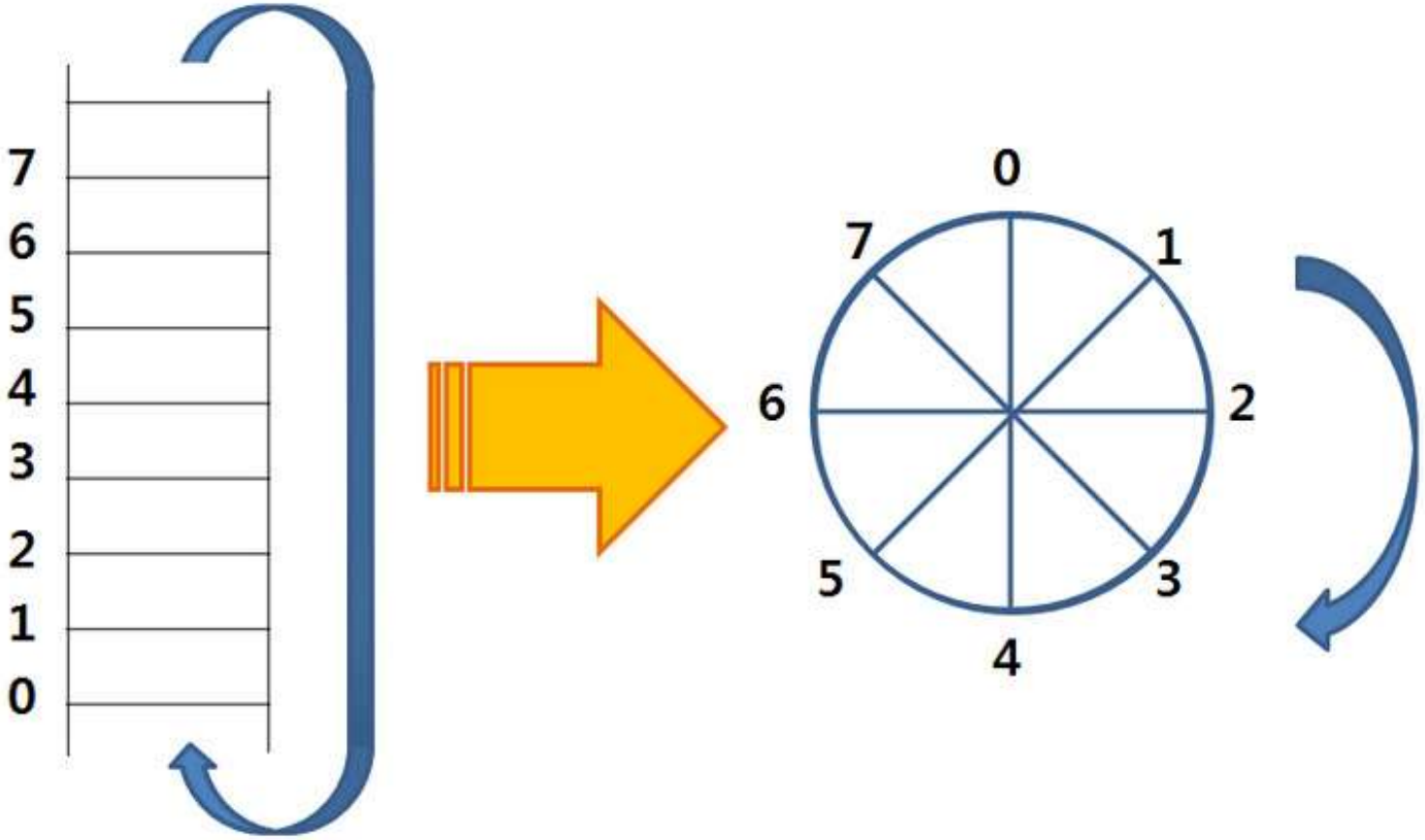
큐

- **선형 큐**



큐

- 원형 큐

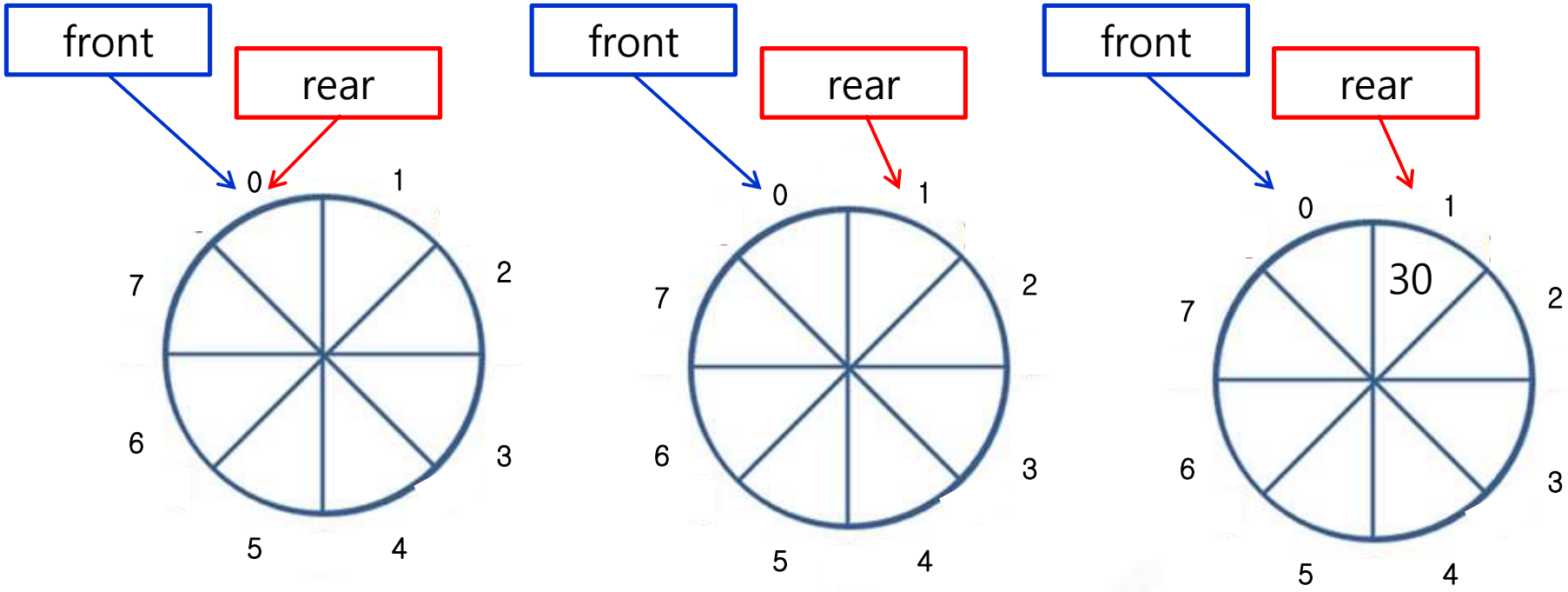


큐

- **원형 큐**
  - 선형 큐의 문제를 해결
  - front는 첫 번째 요소의 하나 앞의 요소, rear는 큐의 마지막 요소
  - 삽입 시에는 rear가 먼저 증가되고, 증가된 위치에 새로운 데이터 삽입
  - 삭제 시에는 front가 먼저 증가되고, 증가된 위치에 데이터 삭제
  - front와 rear의 값이 같으면 원형 큐가 공백 상태
  - 하나의 자리를 비워 포화 상태와 공백 상태를 구분

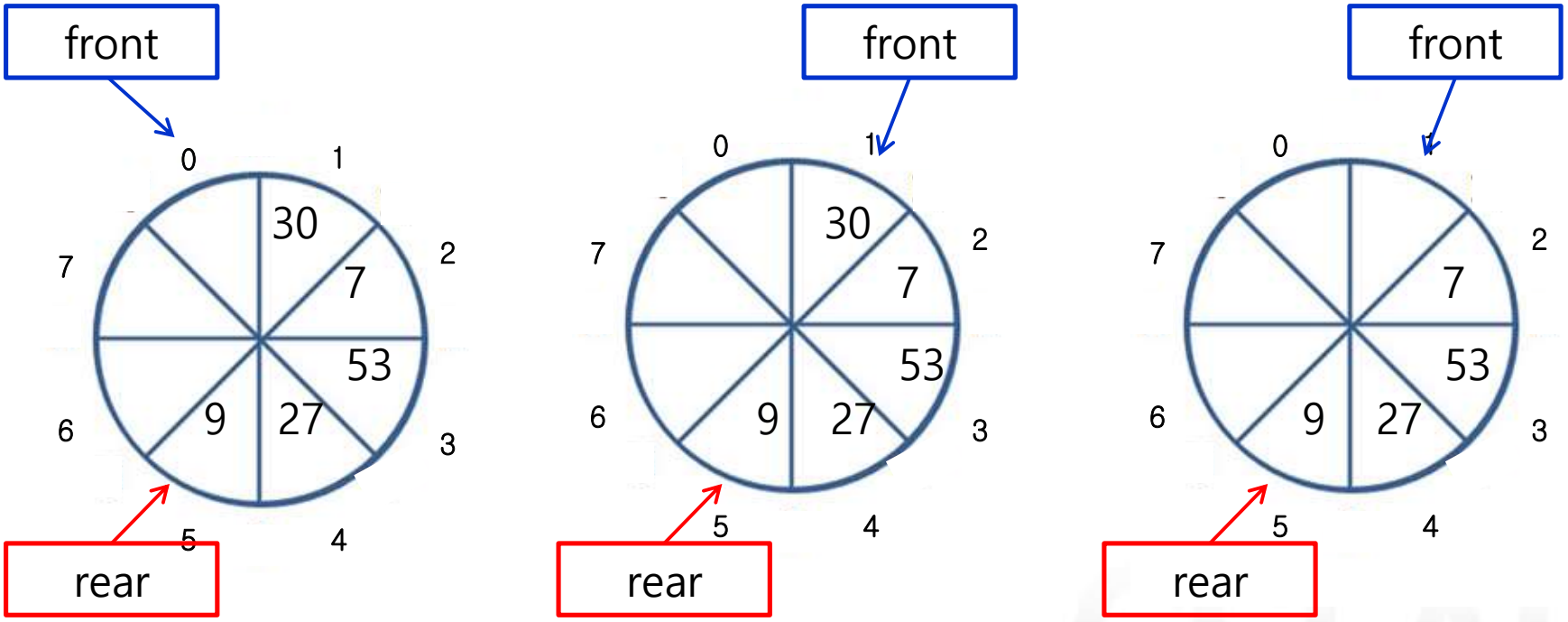
큐

- 원형 큐의 삽입


front
rear
0
1
2
3
4
5
6
7
front
rear
0
1
2
3
4
5
6
7
front
rear
0
1
2
3
4
5
6
7
30

큐

- 원형 큐의 삭제


front
0
1
2
3
4
5
6
7
30
7
53
9
27
rear
front
0
1
2
3
4
5
6
7
30
7
53
9
27
rear
front
0
2
3
4
5
6
7
7
53
9
27
rear

큐 (개인 실습 #2)

- **원형 큐의 구현 시 필요한 요소**
  - 큐의 초기화
  - 큐가 공백 상태에 있는지 검사
  - 큐가 포화 상태에 있는지 검사
  - 삽입 연산 enqueue
  - 삭제 연산 dequeue
  - 큐의 맨 앞에 요소를 확인하는 peek

큐

- **큐의 구현**
  - 큐의 구조체

```
#define MAX_QUEUE_SIZE 5
typedef int element;
typedef struct {
        element queue[MAX_QUEUE_SIZE ];
        int front, rear;
} QueueType;
```

  - 초기화 함수

```
void init(QueueType *q)
{
        q->front = q->rear = 0;
}
```

큐

- **큐의 구현**
  - 에러 메시지

```
void error(char *message)
{
        fprintf(stderr,"%s\n",message);
        exit(1);
}
```

큐

- **큐의 구현**
  - 공백 상태 검출 함수

```
Int is_empty(QueueType *q)
{           //if front = rear
            //then return TRUE
}           //else return FALSE
```

  - 포화 상태 검출 함수

```
Int is_full (QueueType *q)
{       //if front = (rear+1) mod MAX
        //then return TRUE
}       //else return FALSE
```

큐

- **큐의 구현**
  - 삽입 함수 enqueue

```
void enqueue( QueueType *q, element item )
{
        if(is_full(q)) error("Queue is full");

        //rear <- (rear+1) mod MAX_QUEUE_SIZE
        //Q[rear] <- item
}
```

큐

- **큐의 구현**
  - 삭제 함수 dequeue

```
element dequeue(QueueType *q)
{
        if(is_empty(q)) error("Queue is empty");

            //front <- (front+1) mod MAX_QUEUE_SIZE
            return Q[front]
}
```

큐

- **큐의 구현**
  - peek 함수

```
element peek(QueueType *q)
{
        // return queue[(front+1) mod MAX_QUEUE_SIZE]
}
```

큐

- **큐의 구현**
  - main 함수

```
void main()
{
        QueueType q;
        init(&q);
        printf("front = %d, rear = %d\n", q.front, q.rear);
        enqueue( &q, 1 );
        enqueue( &q, 2 );
        enqueue( &q, 3 );
        printf( "dequque() = %d\n", dequeue(&q) );
        printf( "dequque() = %d\n", dequeue(&q) );
        printf( "dequque() = %d\n", dequeue(&q) );
        printf("front = %d, rear = %d\n", q.front, q.rear);
}
```

제출

- **제출**
  - 개인 실습(#1, #2)
    - 오늘 자정까지 제출 (~ 2020/4/10 23:59)
    - 스택 (배열)
    - 큐
  - 과제
    - Circular Queue 구현 (lab4.docx)
    - 다음 주 목요일 자정까지 제출 (~ 2020/4/16 23:59)

과제

- **Lab4.docx**
  - Enqueue (e)
    - Queue에 새로운 element 삽입
    - Queue가 full일 경우, 에러 메세지 출력
  - Dequeue (d)
    - Queue의 첫 번째 element 삭제
    - Queue가 empty일 경우, 에러 메세지 출력
  - PrintFirstElem (f)
    - Queue의 첫 번째 element 출력
    - Queue가 empty일 경우, 에러 메세지 출력
  - PrintLastElem (r)
    - Queue의 마지막 element 출력
    - Queue가 empty일 경우, 에러 메세지 출력

과제

- 함수
  - void MakeEmpty(CircularQueue *Q, int max);
  - int IsEmpty(CircularQueue *Q);
  - int IsFull(CircularQueue *Q);
  - void Dequeue(CircularQueue *Q);
  - void Enqueue(CircularQueue *Q, int X);
  - void PrintFirst (CircularQueue *Q);
  - void PrintRear (CircularQueue *Q);

과제

- **초기 선언**

```
typedef struct CircularQueue{
    int *key;
    int front;
    int rear;
    int max_queue_size;
}CircularQueue;
```

  - void MakeEmpty(CircularQueue *Q, int max)
    - max크기의 큐를 생성 (배열)
      - int *key 사용 (동적 할당)
    - front, rear 의 값을 0으로 설정
    - max_queue_size = max

과제

- **프로그램**
  - 파일포인터 사용 (input.txt)
  - 첫 줄 읽고, 큐 초기화
  - 명령문 처리 (반복문, switch)

input

```
n 5
f
r
e 13
d
d
e 10
e 3
f
r
e 1
e 5
e 6
e 15
f
r
```

output

```
cslab@CSLAB-Computing-Server:~/DS/week4$ ./a.out input.txt
Queue is Empty
Queue is Empty
Queue is Empty
First Element : 10
Last Element : 3
Queue is full
Queue is full
First Element : 10
Last Element : 5
```