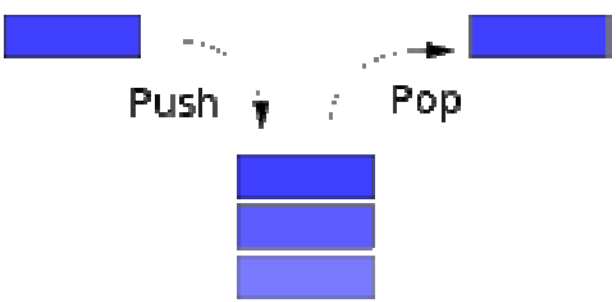
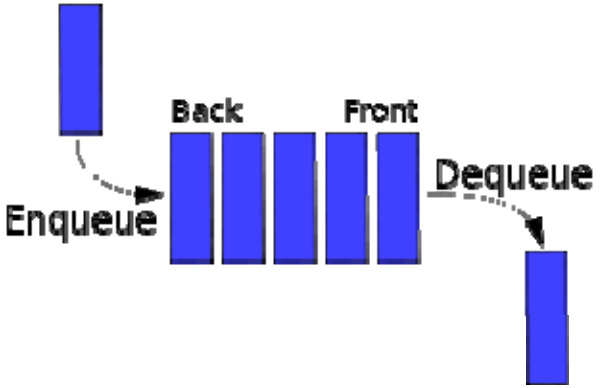
제 4 강의 . 스택과 큐 자료구조





제 4 강 . 스택과 큐 자료구조

학습 목차

1. 스택과 큐 자료구조
2. 스택 자료구조
3. 큐 자료구조
4. 원형 큐의 구현

1. 스택(Stack)과 큐 자료구조

❖ 리스트, 스택과 큐(Stack and Queue)

스택과 큐는 리스트 자료구조의 특별한 경우이다.

✓**리스트**

- 순서가 있다
- 읽기, 삽입(insert)과 삭제(delete)를 리스트의 어느곳에서나 행함

✓**스택**

- 순서가 있다
- 삽입(insert)과 삭제(delete)를 리스트의 한쪽(top이라고 부름)에서 행함

✓**큐**

- 순서가 있다
- 삽입(insert)은 리스트의 한쪽(rear)에서 하고 삭제(delete)는 삽입의 반대쪽 (front)에서 행함

리스트(순서가 있는 자료구조)

리스트

읽기, 삽입(insert), 삭제(delete) :
아무곳에서나 가능

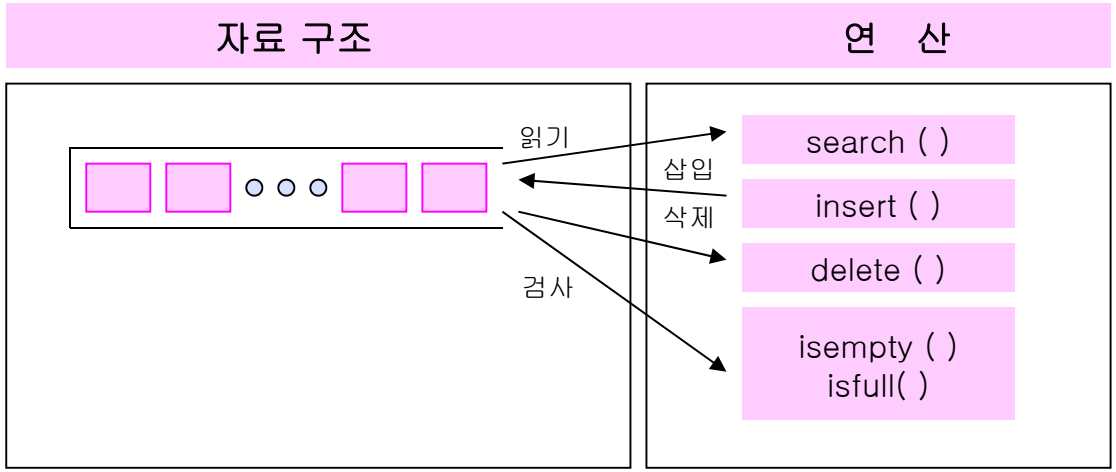
스택

삽입(insert) - 리스트의 한쪽(top이라고 부름)에서 행함
삭제(delete) - 리스트의 한쪽(top이라고 부름)에서 행함

큐

삽입(insert) - 리스트의 한쪽(rear)에서 행함
삭제(delete) - 삽입의 반대쪽(front)에서 행함

❖ 자료구조와 연산 모델



자료구조 = 자료선언 + 연산

자료구조는 자료의 선언과 자료에 대한 연산으로 구성된다.

자료의 선언은 프로그램 언어의 타입을 이용하여 선언 된다.

연산은 자료의 검색과 갱신에 관한 연산으로 이루어 진다.

검색은 원하는 자료를 1개 혹은 일부를 찾는 작업이다.

갱신은 삽입, 삭제, 수정을 말한다.

2 스택(Stack)

- 스택 $S = (a_0, \cdots, a_{n-1})$

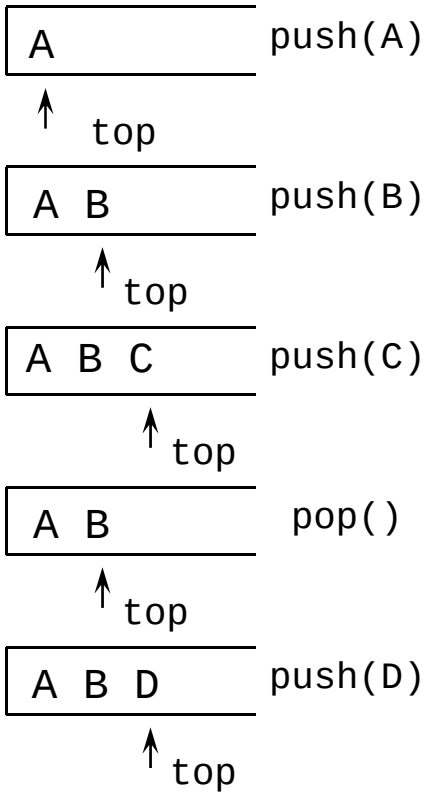
  a_0 : bottom 원소

  a_{n-1} : top 원소

  a_i : 스택 원소 (0<i<n), i+1번째원소

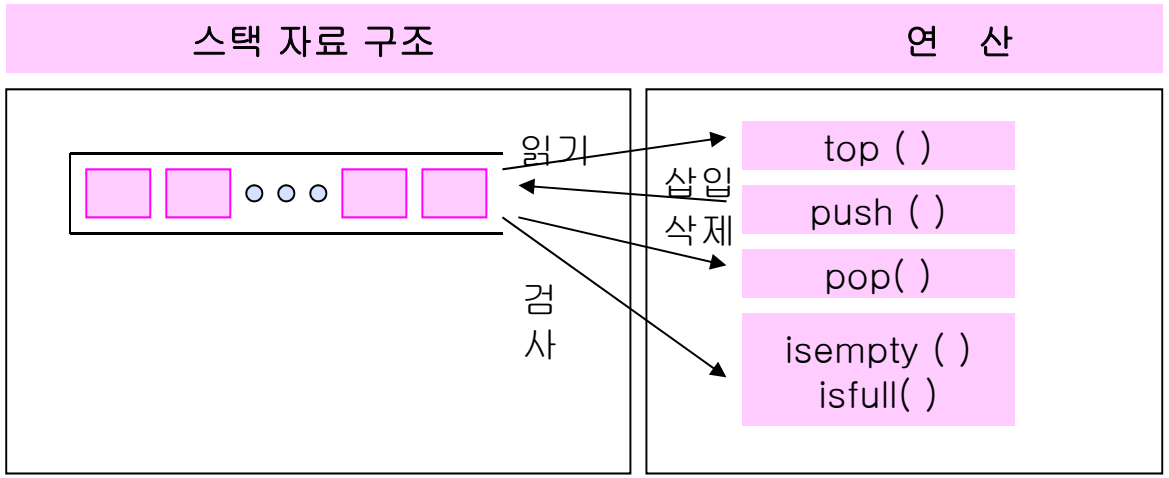
-삽입은 리스트의 마지막에
 삭제도 리스트의 마지막에서 행한다.

- Last-In-First-Out (LIFO) 자료구조



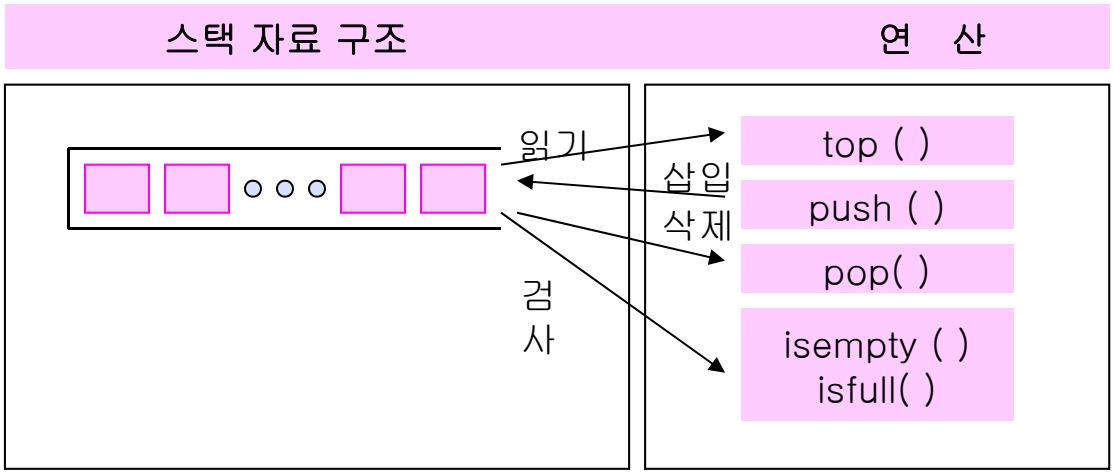
스택에서의 원소의 삽입과 삭제

❖ 스택 자료구조와 연산



top()	: 스택의 맨 위에 있는 데이터 값을 반환한다.
push()	: 스택에 데이터를 삽입한다.
pop()	: 스택에서 데이터를 삭제하여 반환한다.
isempty()	: 스택에 원소가 없으면 'true' 있으면 'false' 값 반환
isfull()	: 스택에 원소가 없으면 'false' 있으면 'true' 값 반환

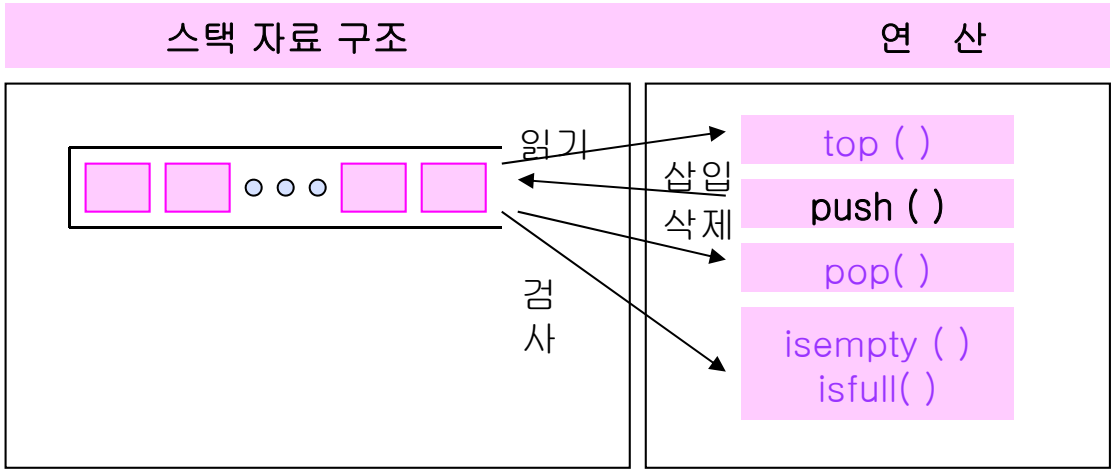
❖ 스택의 생성



스택의 생성

```
- 1 차원 배열의 사용
  #define MAX_STACK_SIZE 100
  int stack[MAX_STACK_SIZE];
  int top = -1;
/* top의 초기 값은 스택이 비어있을 때 -1이다. */
```

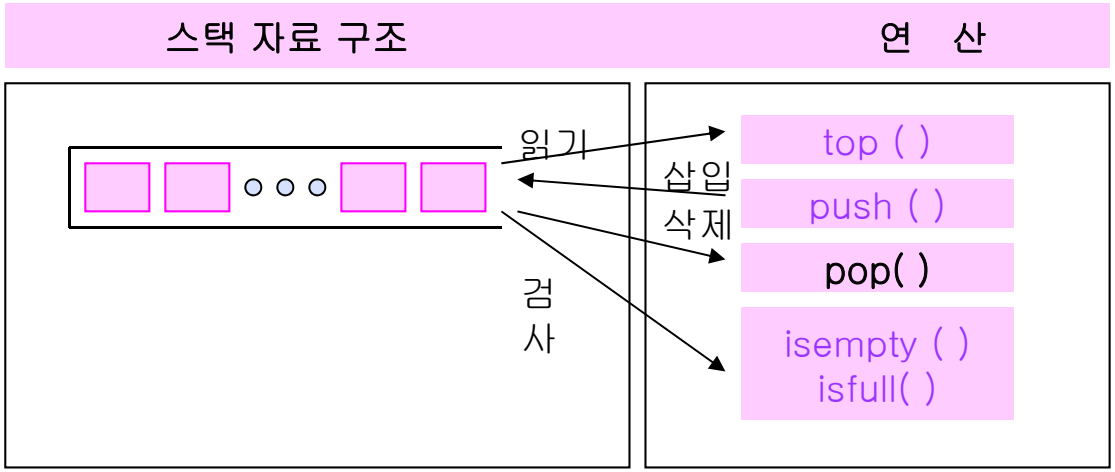
❖스택의 구현 – push() 함수



스택의 원소 삽입 (push)

```
void push(int item)
{
        if(top >= MAX_STACK_SIZE - 1) {
          stack_full();
          return;
        }
          stack[++top] = item;
          // top++; stack[top] = item;
}
```

❖ 스택의 구현 – pop( ) 함수

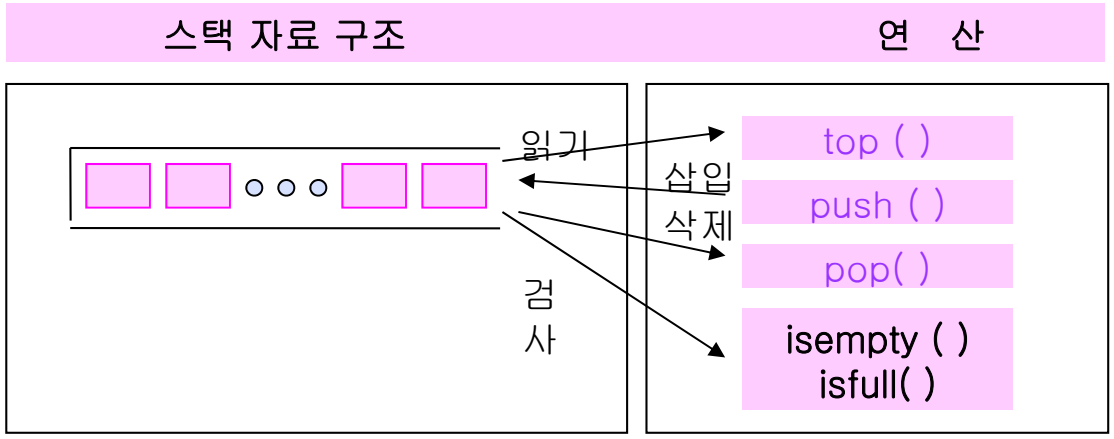


스택의 원소 삭제 (pop)

```
int pop( ) {
    if(top == -1)
        return stack_empty();
    return stack[top--];
    // int x; x = stack[top]; top--; return x;
}
```

< C 자료구조 입문 >

❖ 스택의 구현 – isempty(), isfull()



스택의 검사

```
스택이 비어있는지 검사하는 함수
int isempty()
{ if( top < 0 ) return(1); else return(0); }

스택이 가득 차 있는지 검사하는 함수
int isfull()
{ if ( top >= MAX_STACK_SIZE -1 ) return(1); else
return(0); }
```

❖ 스택의 구현 – C 언어 – 아래 프로그램을 실행해보자

– 실행후 다음과 같은 점을 생각해보자
스택의 원소 타입이 int 가 아니면 어떻게 고쳐야하는가?
스택(stack)을 2개 사용하려면 어떻게 해야하는가?

```
#include <stdio.h>
#define MAX_STACK_SIZE 100

int stack[MAX_STACK_SIZE];
int top = -1;

void push(int item)
{
  if(top >= MAX_STACK_SIZE - 1) {
    printf("stack_full()\n");
    return;
  }
  stack[++top] = item;
}

int pop() {
        if(top == -1)
        {
                printf("stack_empty()\n"); exit();
        }
        return stack[(top)--];
}
```

❖ 스택의 구현 – 계속

```
int isempty()
{ if( top == -1 ) return(1); else return(0); }

int isfull()
{ if ( top >= MAX_STACK_SIZE -1 ) return(1);
  else return(0); }

int main()
{
        int e;
        push(20);
        push(40);
        push(60);

        printf(" Begin Stack Test ...\n");

        while(!isempty())
        {
                e = pop();
                printf("value = %d\n", e);
        }
}
```

Stack연습

(스택의 실험) stacktest.c를 바꾸어 다음과 같은 프로그램을 완성하여 보아라.(스택과 스택 관련 함수를 2개씩만든다)

(실험 1)

수유역에 도착하는 임의의 마을버스 이름 4개를 스택에 입력한다.

어느 프로야구 팀의 선수 번호 4개를 스택에 입력한다.

마을버스와 야구선수 번호를 양쪽 스택에서 하나씩 pop() 하여 출력한다.

(실험 2)

실험 1에서 야구선수 번호 대신 야구선수 이름을 스택에 입력한다.

(실험 3)

실험 2에서 (야구선수번호, 야구선수이름)을 구조체로 만들어 스택에 입력한다.

❖ 스택의 구현 – Java 언어

```
// Stack.java
import java.io.*;                    // for I/O
class StackX
{  private int maxSize;          // size of stack array
   private double[] stackArray;
   private int top;              // top of stack
   public StackX(int s)          // constructor
   {  maxSize = s;               // set array size
      stackArray = new double[maxSize];  // create array
      top = -1;                  // no items yet
    }
   public void push(double j)  // put item on top of stack
      { stackArray[++top] = j;} // increment top, insert item
   public double pop()         // take item from top of stack
      { return stackArray[top--]};  // decrement top
   public double peek()        // peek at top of stack
      { return stackArray[top];      }
   public boolean isEmpty()    // true if stack is empty
      { return (top == -1);       }
   public boolean isFull()     // true if stack is full
      { return (top == maxSize-1);     }
}  // end class StackX
```

❖ 스택의 구현 – Java 언어

– 실행 후 다음과 같은 점을 생각해보자
스택의 원소 타입이 int 가 아니면 어떻게 고쳐야하는가?
스택(stack)을 2개 사용하려면 어떻게 해야하는가?

```
//////////////////////////////////////////////////////////////
class StackApp
   {
   public static void main(String[] args)
      {
      StackX theStack = new StackX(10);  // make new stack
      theStack.push(20);                 // push items onto stack
      theStack.push(40);
      theStack.push(60);
      theStack.push(80);

      while( !theStack.isEmpty() )       // until it's empty,
         {                               // delete item from stack
         double value = theStack.pop();
         System.out.print(value);        // display it
         System.out.print(" ");
         }  // end while
      System.out.println("");
      }  // end main()
   }  // end class StackApp
```

C 언어와 Java의 스택 구현 방법 비교

```
int stack[SIZE];

void push(int item);
int pop();
int isempty();
int isfull();
```

```
int main()
{
   int e;
   push(20);
   push(40);
   push(60);
     ...
```

```
class StackX

private int maxSize;
private .. stackArray;
private int top;

public void push( )
public double pop()
public boolean isEmpty()
public boolean isFull()
```

```
class StackApp

public … main()
{ StackX theStack =
        new StackX(10);
  theStack.push(20);
  theStack.push(40);
    . . .
```

C 언어 – 1개의 프로그램에서 2개의 스택 사용

```
int stack1[SIZE];

void push1(int item);
int pop1();
int isempty1();
int isfull1();
```

```
int stack2[SIZE];

void push2(int item);
int pop2();
int isempty2();
int isfull2();
```

```
int main()
{
   int e;
   push1(20);
   push2(40);
   push1(60);
     ...
```

스택 Class 정의

```
class StackX
    private int maxSize;
    private .. stackArray;
    private int top;

    public void push( )
    public double pop()
    public boolean isEmpty()
    public boolean isFull()
```

Java – 1개의 프로그램에서 2개의 스택 사용

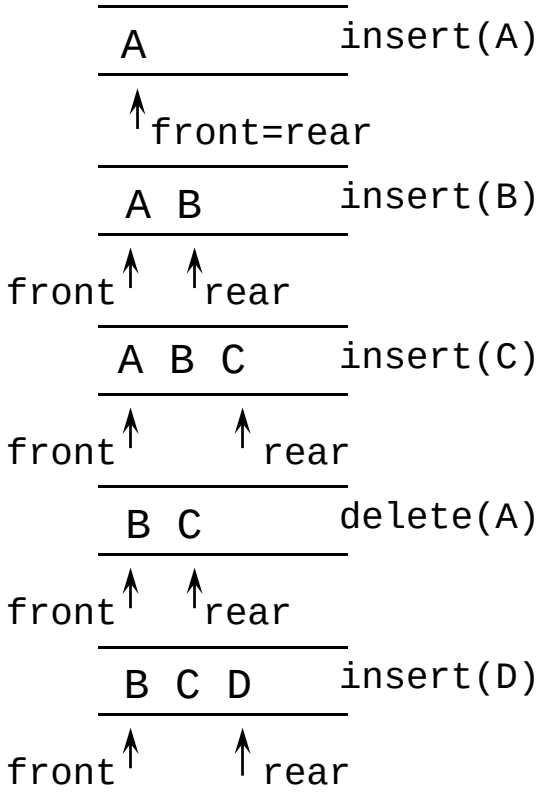
```
class StackApp
 public … main()
 { StackX theStack1 =
         new StackX(10);
   StackX theStack2 =
         new StackX(10);

   theStack1.push(20);
   theStack2.push(40);
     . . .
```

Java – 다른 프로그램에서 스택 사용

```
class StackApp2
 public … main()
 { StackX myStack =
         new StackX(10);
   myStack.push(20);
   myStack.push(40);
     . . .
```

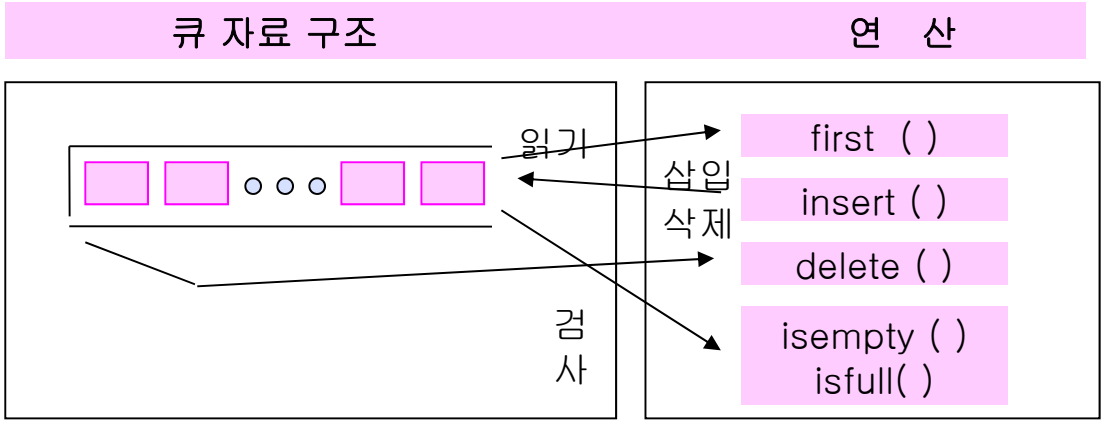
3. 큐 자료구조

- 순서 있는 리스트
- 삽입 연산은 한쪽 끝에서 일어난다(rear라고 부른다)
- 삭제 연산은 삽입의 반대쪽에서 일어난다.(front라고 부른다.)
- FIFO(First In First Out)



큐에서의 원소의 삽입과 삭제

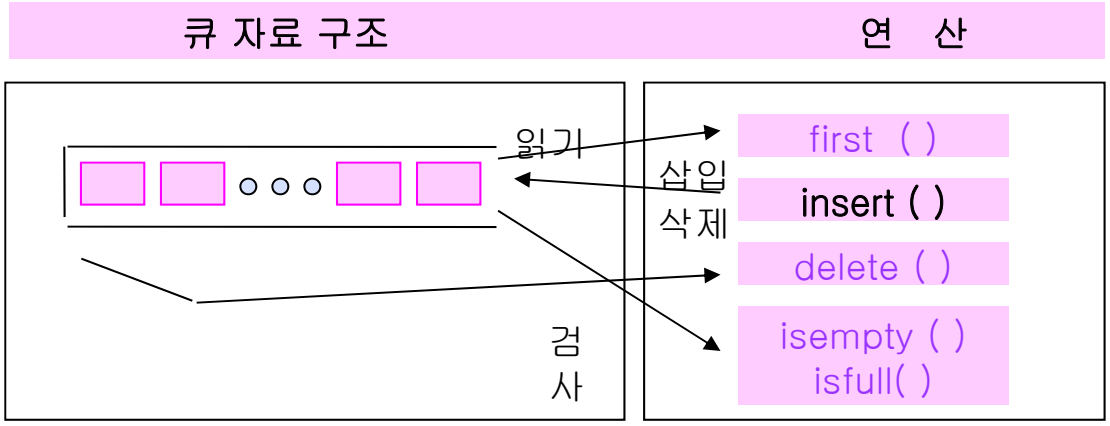
❖ 큐의 생성



큐의 구현

```
/* 1차원 배열의 사용, front와 rear 변수 사용 */
#define MAX_QUEUE_SIZE 100
typedef struct {
        int key;
        /* other fields */
} element;

element queue[MAX_QUEUE_SIZE];
int rear = -1;
int front = -1;
```

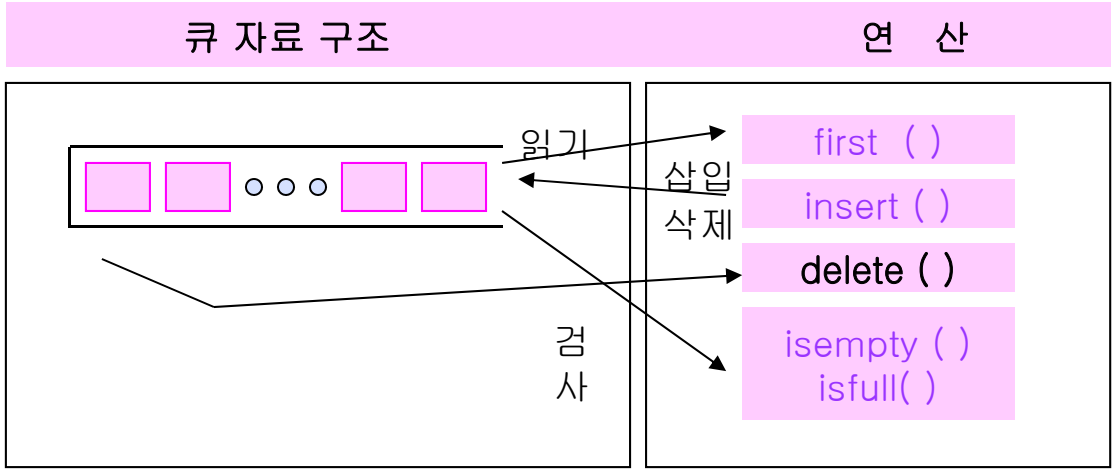
❖ 큐의 구현 – insert, add()



큐
원소의 삽입

```
void insert(int *rear, element item) {
    if(*rear == MAX_QUEUE_SIZE - 1) {
        queue_full();
        /* 처리방법 - 큐의 원소를 모두 원쪽으로 이동 ? */
        return;
    }
    queue[++*rear] = item;
}
```

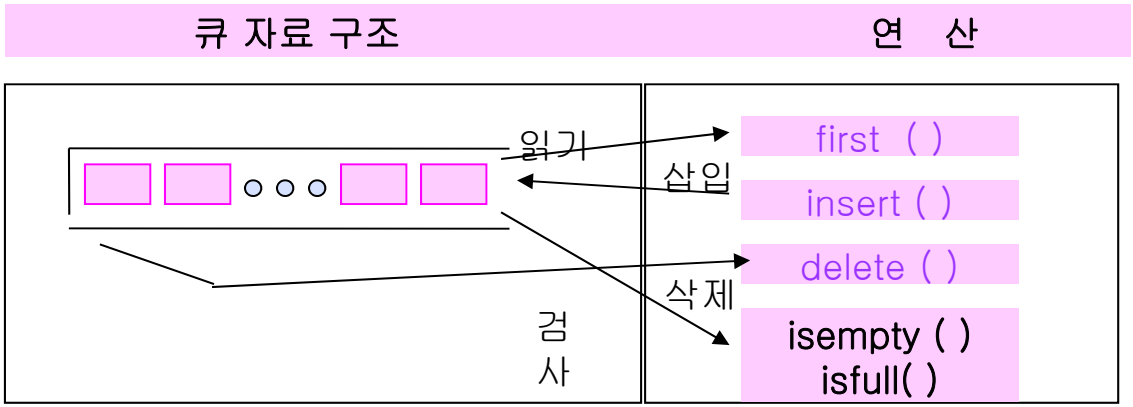
❖ 큐의 구현 – delete



큐
원소의 삭제

```
element delete(int *front, int rear) {
    if(*front == rear)
        return queue_empty();
            /* return an error key */
    return queue[++*front];
}
```

❖ 큐의 구현 – isempty(), isfull()



큐
원소 검사

```
큐가 비어있는지 검사하는 함수
 int isempty()
{ if( front == rear ) return(1); else return(0); }

큐가 가득 차 있는지 검사하는 함수
int isfull()
{ if ( rear == MAX_QUEUE_SIZE -1 ) return(1);
  else return(0); }
```

❖ 큐의 예

•예 – 고객 작업 처리

은행, 미용실 등 고객은 큐를 구성한다.

각 고객은 점원에게 한 개의 처리 작업이 된다.(job)

점원은 작업을 한 개씩 처리한다.

front	rear	Q[0]	Q[1]	Q[2]	Q[3]	설명 s
-1	-1					초기 상태
-1	0	J1				Job 1 삽입
-1	1	J1	J2			Job 2 삽입
-1	2	J1	J2	J3		Job 3 삽입
0	2		J2	J3		Job 1 삭제
1	2			J3		Job 2 삭제

큐에 작업이 들어온 상태

4. 원형 큐(circular queue)

큐의 구현에서 문제점

작업 큐에서 MAX_QUEUE_SIZE 만큼 원소가 삽입되면?

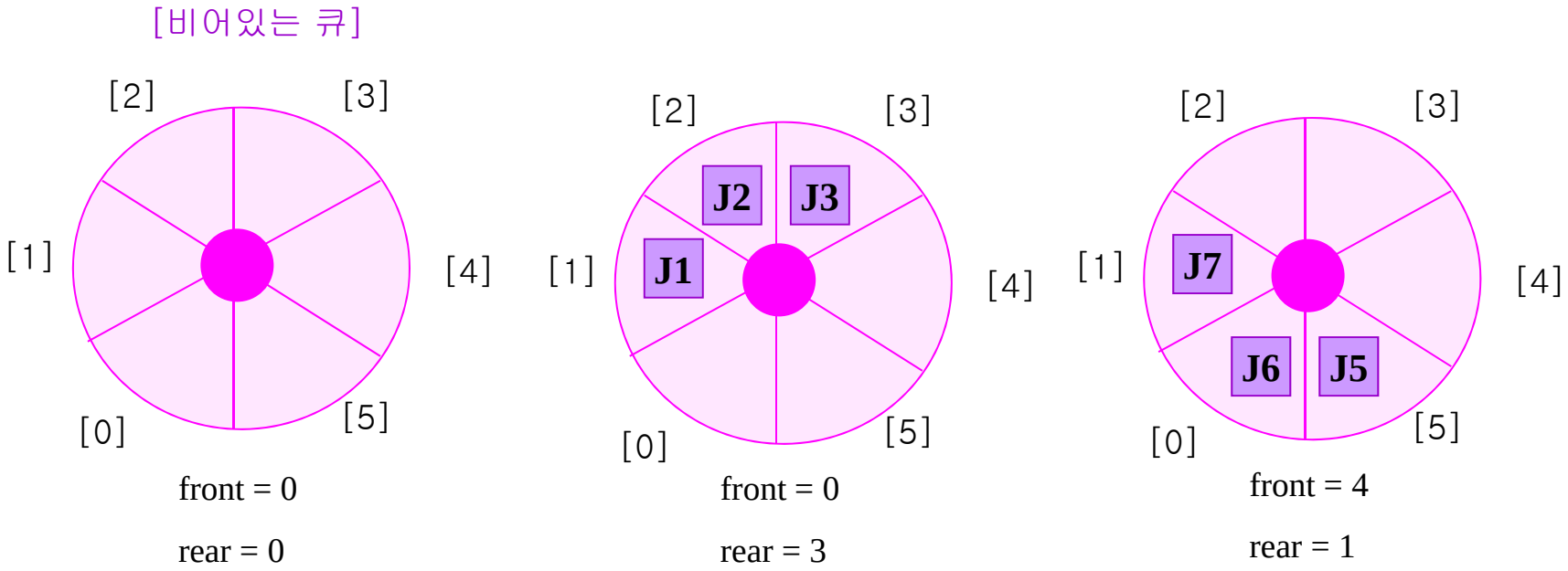
-> 큐가 full이 된다.

(issempty(), isfull() 동시에 참인 경우도 있다)

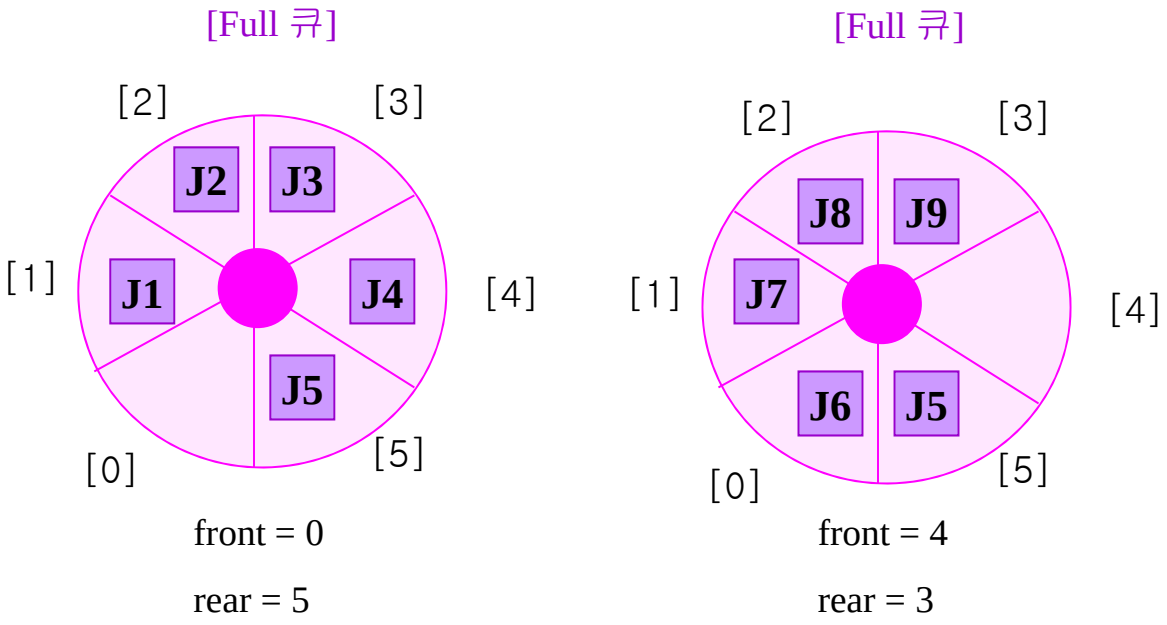
해결

- 배열을 **원형**으로 만들고 원형 배열에 데이터를 저장한다.
- 초기값 front와 rear는 0으로 만든다(-1이 아님) - 두값이 같으면 empty 이다.
- front 변수는 (큐의 처음 값 위치에서 **1을 뺀** 값이다)
- rear 변수는 (큐의 **마지막 값** 위치)를 가리킴
- 최대 MAX_QUEUE_SIZE - 1 개를 저장한다.(MAX_QUEUE_SIZE 가 아님!!)

❖원형 큐 의 예



비어있는 큐와 3개가 삽입된 큐의 상태


[Full 큐]
[2]
[3]
J2
J3
[1]
J1
J4
[4]
J5
[0]
[5]
front = 0
rear = 5
[Full 큐]
[2]
[3]
J8
J9
[1]
J7
[4]
J6
J5
[0]
[5]
front = 4
rear = 3

Full 원형큐

원형 큐 구현에 주의할 점

-insert 연산 후 rear의 값

```
*rear = (*rear +1) % MAX_QUEUE_SIZE ;
```

-delete 연산 후 front의 값

```
front = (front +1) % MAX_QUEUE_SIZE ;
```

❖ 원형 큐 구현

원형 큐의 삽입 - add()

```
void add(int front, int *rear, element item) {
        *rear = (*rear + 1) % MAX_QUEUE_SIZE;
        if(front == *rear) {
                queue_full(rear);
                        /* reset rear and print error */
                return;
        }
        queue[*rear] = item;
}
```

원형 큐의 삭제 - delete ()

```
element delete(int *front, int rear) {
/* remove front element from the queue and put it in item */
        element item;
        if(*front == rear)
                return queue_empty();
                        /*queue_empty returns an error key*/
        *front = (*front + 1) % MAX_QUEUE_SIZE;
        return queue[*front];
}
```

❖ 원형 큐 구현 - 정리

- full 큐와 empty 큐의 구별 방법(1가지를 선택하여 실천한다.)

  방법 1 : 최대 MAX_QUEUE_SIZE – 1 만 저장한다(front==rear+1 이면 full)

  (MAX_QUEUE_SIZE가 아님)

  방법 2 : 변수를 추가하여 두 상태를 구별한다,(데이터 개수 n을 항상 계산한다.)

- 큐를 유지하기위해 데이터 이동이 불 필요해진다.

  queue: O(n) – circular queue: O(1)

(생각할점) front와 rear의 초기값을 -1로 하면 무슨 문제가 생기는가?

Review

⦿ 리스트는 가장 많이 사용되는 자료의 형태이다. 리스트 자료에 대한 연산은 검색과 변경이다. 변경은 삽입과 삭제 연산을 말한다.

⦿ 스택은 리스트의 특별한 형태로 삽입과 삭제가 한쪽 끝에서만 일어난다. 마치 문이 한 개인 관광버스에 손님들을 타고 내릴 때, 먼저 탄 사람이 맨 나중에 내리는 것과 같은 구조이다(First In Last Out). 다시 말하면 맨 나중에 탄 사람이 맨 먼저 내리는 구조이다(Last in First Out, LIFO).

⦿ 큐 또한 리스트의 특별한 형태로 삽입과 삭제가 한쪽 끝에서만 일어난다. 삽입과 삭제는 서로 반대쪽에서 일어나는 것이 스택과 다르다. 문이 두개인 마을버스의 타는 곳과 내리는 곳이 따로 있어서 먼저 탄 사람이 먼저 내리게 되는 구조이다(First In First Out, FIFO).

⦿ 스택과 큐는 이렇게 리스트의 특별한 형태이지만 많이 쓰이는 자료 구조이다. 스택과 큐는 배열을 이용하여 구현할 수 있다.

⦿ 큐를 구현할 때는 일반 배열을 사용하면 자료를 이동해야 하는 불편함이 있어서 일반 배열을 원형으로 만들어서 원형 배열을 이용하여 구현한다.