# 정리(스택)

- 선형리스트의 끝부분에서만 데이터의 입력과 출력이 가능한 순서 리스트 자료구조
- 마지막에 삽입(Last-In)한 원소는 맨 위에 쌓여 있다가 가장 먼저 삭제(First-Out) 되는
  ☞ 후입선출 구조 (LIFO, Last-In-First-Out)

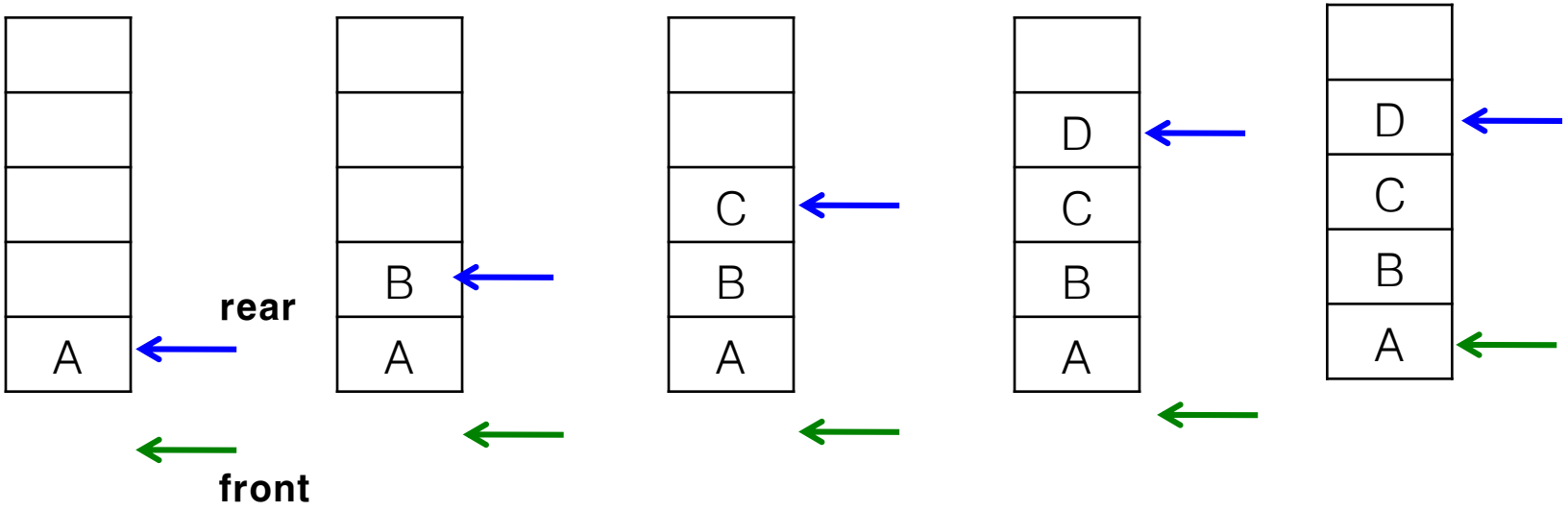
[6주차 과제4] 스택을 활용하여 다음 수식(중위표기식)을 후위표기식으로 바꾸고 그 값을 계산하는 과정에서 스택의 변화를 그리시오.

**8 - 9 / 3 + 6 – 3 * 4**

# 큐의 정의와 구조

## ◆ 큐(Queue)

- 큐는 뒤에서는 삽입만 하고, 앞에서는 삭제만 할 수 있는 리스트 자료구조
  - 삽입한 순서대로 원소가 나열되어 가장 먼저 삽입(First-In)한 원소는 맨 앞에 있다가 가장 먼저 삭제(First-Out)된다.
    - ☞ 선입선출 구조 (FIFO, First-In-First-Out)



# 배열을 이용한 큐의 구현

## ◆ 일반큐

- 1차원 배열을 이용한 큐
  - 큐의 크기 = 배열의 크기(MAX_QUEUE_SIZE)
  - 변수 front : 마지막으로 큐에서 삭제된 데이터의 인덱스 저장
  - 변수 rear : 마지막으로 큐에 들어 온 데이터의 인덱스 저장

- 상태 표현
  - 초기 상태 : front = rear = -1
  - 공백 상태 : front == rear
  - 포화 상태 : rear = MAX_QUEUE_SIZE-1

# 배열을 이용한 큐의 구현

```
# define MAXQ_SIZE 100    /* 큐의 최대 크기 */

typedef struct {
    int job_num;
    char grade;
} element;

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

# 큐의 연산(1)

큐에 element type의 데이터의 삽입

```
void addq(element item)
{
     /* queue에 item을 삽입 */
   if (rear == MAX_QUEUE_SIZE -1)
      printf("Queue is full!!");
   else
      queue[++rear] = item;
}
```

# 큐의 연산(2)

**큐로부터 element type의 데이터 삭제**

```
element deleteq ()
{
     /* queue의 앞에서 원소를 삭제 */
   if (front == rear)
      printf("Queue is Empty");
   else
      return queue[++front];
}
```

# 큐의 연산 활용 실습

```
main()
{
   int i, jnum, out, gradeA=0;
   element temp;

   printf("오늘 작업한 작업 수는? ");
   scanf("%d", &jnum);
   printf("작업번호와 작업상태 입력 :\n");

   for (i=0; i < jnum; i++) {
      scanf("%d %c",
            &temp.job_num, &temp.grade);
      addq(temp);
   }

   printf("\nfront = %d :: rear = %d\n\n",
         front, rear);
```

```
오늘 작업한 수는? 5
작업번호  작업상태
100       D
199       A
170       A
200       C
177       B

front=-1 :: rear=4
```

# 큐의 연산 활용 실습

```
    for (i=0; i < jnum; i++) {
        temp = deleteq();
        if (temp.grade == 'A') {
            printf("%d\t%c\n", temp.job_num, temp.grade);
            gradeA++;
        }
        else
            addq(temp);
    }

    printf("A 등급인 job은 %d개입니다. \n", gradeA);
    printf("\n\nfront = %d :: rear = %d\n\n", front, rear);

}
```

오늘 작업한 수는? 5

| 작업번호 | 작업상태 |
|---|---|
| 100 | D |
| 199 | A |
| 170 | A |
| 200 | C |
| 177 | B |

A 등급인 job은 2개입니다.
front=4 :: rear=7

# 큐의 동작 시뮬레이션

[큐의 연산 활용 실습 프로그램] 다음 빨간 부분과 같이 입력을 줄 경우 변수 front rear의 값을 출력하는 두 번의 printf 문장에 의해 출력되는 값을 쓰고 실습을 통하여 확인하시오.

오늘 작업한 수는? 5

| 작업번호 | 작업상태 |
|---|---|
| 100 | D |
| 199 | A |
| 170 | A |
| 200 | C |
| 177 | B |

front=-1 :: rear=4
front=4 :: rear=7

# 큐의 동작 시뮬레이션

◆ 삽입, 삭제 연산 n=7 (큐의 크기)

| front | rear | Q[0] | Q[1] | Q[2] | Q[3] | Q[4] | Q[5] | Q[6] |
|---|---|---|---|---|---|---|---|---|
| -1 | -1 | | | | | | | |
| -1 | 3 | J0 | J1 | J2 | J3 | | | |
| 1 | 3 | | | | J2 | J3 | | |
| 1 | 6 | | | | J2 | J3 | J4 | J5 J6 |
| 4 | 6 | | | | | | | J5 J6 |

◆ 일반큐의 개선

일반큐는 rear가 큐의 마지막 위치를 가리키면 큐가 꽉 찬 것으로 인식하지만 실제로는 앞에 서비스 되어 나갔기 때문에 비어 있는 경우가 많다.

→ 원형큐의 개념 도입

# 원형큐의 정의

- 1차원 배열을 사용하면서
  논리적으로 배열의 처음과 끝이 연결되어 있다고 가정

- 원형 큐의 구조
  - 초기 공백 상태 : front = rear = 0
  - 공백 상태와 포화 상태 구분을 하기 위해서 (큐의 크기-1)개 데이터만 저장하게됨

# 원형큐 데이터 처리 예

큐의 크기가 7 인 경우
- J0, J1, J2, J3 을 삽입
- 2개 삭제
- J4, J5, J6, J7 을 삽입

이때 표에서 보는 바와 같이 front=2, rear=10이 되고

다음 데이터를 삽입하려고 하면 QUEUE가 FULL 되어 삽입할 수 없다.

즉 이 경우 최대 6개의 공간만을 사용할 수 있다.

| front | rear | Q[0] | Q[1] | Q[2] | Q[3] | Q[4] | Q[5] | Q[6] |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | | | | | | | |
| 0 | 4 | | J0 | J1 | J2 | J3 | | |
| 2 | 4 | | | | | J2 | J3 | |
| 2 | 1 | J6 | J7 | | | J2 | J3 | J4 | J5 |
| | | | | | | | | |

# 원형큐의 연산(1)

## ◆ 원형 큐의 삽입 연산

원형큐의 처음과 마지막이 연결되어 있는 것으로 인덱스를 찾아가기 위하여

rear = (rear + 1) % MAXQ_SIZE

와 같은 문장을 사용한다.

```
void caddq(element item)
{
        int  next_rear = (rear + 1) % MAXQ_SIZE;
        if  (next_rear  == front)
            printf( “Queue is Full !!!” );
        else {
            rear = next_rear;
            queue[rear] = item;
        }
}
```

# 원형큐의 연산(2)

◆ 원형 큐에서의 삭제 연산

```
element cdeleteq()
{
      if  (rear  == front)
          printf( “Queue is Empty!!!” );
      else  {

          front= (front + 1) % MAXQ_SIZE;
          return queue[front]

      }
}
```

# 원형큐의 연산

◆ 주어진 원형큐의 최대 크기가 10  이고, front=2, rear=9일 때

두 번의 삭제와 두 번의 삽입을 한 후에 front, rear 의 값은 어떻게 변하는가?

front = 4,  rear = 1

# 원형큐의 연산 활용 실습

```
#define MAXQ_SIZE  7
typedef struct {
    int job_num;
    char grade;
} element;
element queue[MAXQ_SIZE];
int rear = 0;
int front = 0;
void caddq(element item);
element cdeleteq();
```

# 원형큐의 연산 활용 실습

```
main()
{
     int i, job_num, out, cond=1;
    element temp;
    while (cond) {
        printf("완료한 작업 수는? ");
        scanf("%d", &job_num);
        printf("작업번호와 작업상태 입력 :\n");
        for (i=0; i < job_num; i++) {
            scanf("%d %c", &temp.job_num, &temp.grade);
            caddq(temp);
        }
        printf( “출고할 작업의 수는? ");
        scanf("%d", &out);
        for (i=0; i < out; i++) {
            temp = cdeleteq();
            printf("%d\t%c\n", temp.job_num, temp.grade);
        }
        printf("front = %d :: rear = %d\n", front, rear);
        printf("작업의 입출력을 계속하실래요(1/0)?");
        scanf("%d", &cond);
    }
}
```

```
완료한 작업한 수는? 5
작업번호와 작업상태 입력
100     D
199     A
170     A
200     C
177     B

출고할 작업의 수는? 3
100     D
199     A
170     A

  front=3 :: rear=5
```

# 원형큐의 연산 활용 실습

```
main()
{
     int i, job_num, out, cond=1;
    element temp;
    while (cond) {
        printf("완료한 작업 수는? ");
        scanf("%d", &job_num);
        printf("작업번호와 작업상태 입력 :\n");
        for (i=0; i < job_num; i++) {
            scanf("%d %c", &temp.job_num, &temp.grade);
            caddq(temp);
        }
        printf( “출고할 작업의 수는? ");
        scanf("%d", &out);
        for (i=0; i < out; i++) {
            temp = cdeleteq();
            printf("%d\t%c\n", temp.job_num, temp.grade);
        }
        printf("front = %d :: rear = %d\n", front, rear);
        printf("작업의 입출력을 계속하실래요(1/0)?");
        scanf("%d", &cond);
    }
}
```

```
  front=3 :: rear=5


작업의 입출력을 계속하실래요(1/0)?   1
완료한 작업한 수는? 4
   작업번호와 작업상태 입력
   111        C
   333        A
   177        B
   250        C
 => rear(2)
   출고할 작업의 수는? 4
=> front(0)
```