Chapter 06.
스택

11주차

1. 괄호 처리.
2. 전체 이어 붙이기.

실습
(ListBaseStack.h)

지난 주 작업한 소스에 대한 헤더 파일

```
#ifndef _LISTBASESTACK_H_
#define _LISTBASESTACK_H_

#define TRUE      1
#define FALSE     0

typedef int Data;
typedef struct _node {
          Data data;
          struct _node * next;
} Node;
typedef struct _listStack {
          Node * head;
} ListStack;
typedef ListStack Stack;

void StackInit(Stack * pstack);
int SIsEmpty(Stack * pstack);
void SPush(Stack * pstack, Data data);
Data SPop(Stack * pstack);
Data SPeek(Stack * pstack);

#endif // _LISTBASESTACK_H_
```

실습
(Calculator.c)

```
int GetOpPrec(char op) {
        switch (op) {
        case '*':
        case '/':
                return 5;
        case '+':
        case '-':
                return 3;
        case '(':
                return 1;
        }
        return -1;   // 등록되지 않은 연산자
}

int WhoPrecOp(char op1, char op2) {
        int op1Prec = GetOpPrec(op1);
        int op2Prec = GetOpPrec(op2);
        if (op1Prec > op2Prec)
                return 1;
        else if (op1Prec < op2Prec)
                return -1;
        else
                return 0;
}
```

실습
(Calculator.c : ConvToPostfix 함수 1/2)

```
void ConvToPostfix(char exp[]) {
        Stack stack;
        int expLen = strlen(exp);
        char * convExp;
        int i, idx = 0;
        char tok, popOp;

        convExp = (char*)malloc(expLen + 1);
        memset(convExp, 0, sizeof(char)*expLen + 1);

        StackInit(&stack);
        .........
        ......... (중략)
        .........
        while (!SIsEmpty(&stack))
                convExp[idx++] = SPop(&stack);

        strcpy(exp, convExp);
        free(convExp);
}
```

연산식의 길이가 가변적이므로 메모리 할당 사용

스택 초기화

실습 (Calculator.c : ConvToPostfix 함수 2/2)

```
for (i = 0; i<expLen; i++)  {
        tok = exp[i];
        if (isdigit(tok))
                convExp[idx++] = tok;
        else {
                switch (tok) {
                case '(': SPush(&stack, tok); break;
                case ')':
                        while (1) {
                                popOp = SPop(&stack);
                                if (popOp == '(')
                                        break;
                                convExp[idx++] = popOp;
                        }
                        break;
                case '+': case '-': case '*': case '/':
                        while (!SIsEmpty(&stack) &&
                          WhoPrecOp(SPeek(&stack), tok) >= 0)
                                convExp[idx++] = SPop(&stack);
                        SPush(&stack, tok);
                        break;
                }
        }
}
```

피연산자 : 곧바로 결과식으로!

닫는 괄호 : 여는 괄호 나올까지 스택에서 pop.

스택 top 연산자의 우선순위가 현재연산자보다 높거나 같으면 pop!

실습
(Calculator.c : EvalPostfix 함수)

```
int EvalPostfix(char exp[]) {
        Stack stack;
        int expLen = strlen(exp);
        int i;
        char tok, op1, op2;

        StackInit(&stack);
        for(i=0; i<expLen; i++) {
                tok = exp[i];
                if (isdigit(tok) )
                        SPush(&stack, tok - '0');
                else {
                        op2 = SPop(&stack);
                        op1 = SPop(&stack);
                        switch(tok) {
                        case '+': SPush(&stack, op1+op2);  break;
                        case '-':  SPush(&stack, op1-op2); break;
                        case '*':  SPush(&stack, op1*op2); break;
                        case '/':  SPush(&stack, op1/op2); break;
                        }
                }
        }
        return SPop(&stack);
}
```

피연산자: 무조건 push!
(숫자로 변환하여)

먼저 꺼낸 값이 두 번째 피연산자!

실습
(Calculator.c : EvalInfix 함수)

```
int EvalInfix(char exp[])
{
        int len = strlen(exp);
        int ret;
        char * expcpy = (char*)malloc(len + 1);
        strcpy(expcpy, exp);

        ConvToPostfix(expcpy);
        ret = EvalPostfix(expcpy);

        free(expcpy);
        return ret;
}
```

중위표현식 -> 후위표현식

후위표현식 계산

Chapter 07.
큐

11주차

큐의 ADT
(First-In-First-Out (FIFO), 선입선출)

- `void QueueInit(Queue * pq);`
  - 큐의 초기화를 진행한다.
  - 큐 생성 후 제일 먼저 호출되어야 하는 함수이다.

- `int QIsEmpty(Queue * pq);`
  - 큐가 빈 경우 TRUE(1)을, 그렇지 않은 경우 FALSE(0)을 반환한다.

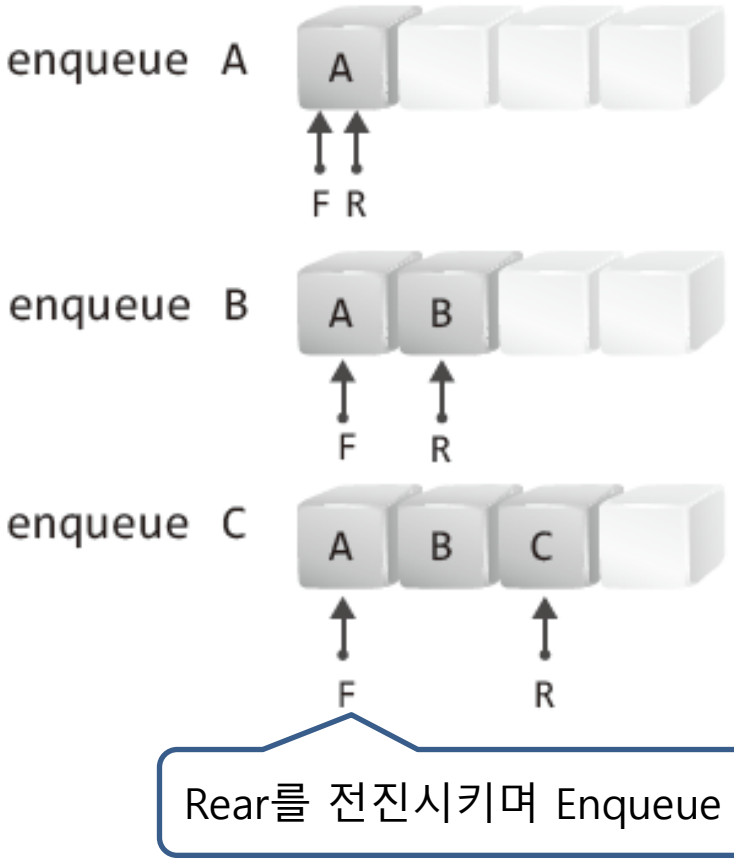
- `void Enqueue(Queue * pq, Data data);`
  - 큐에 데이터를 저장한다. 매개변수 data로 전달된 값을 저장한다.

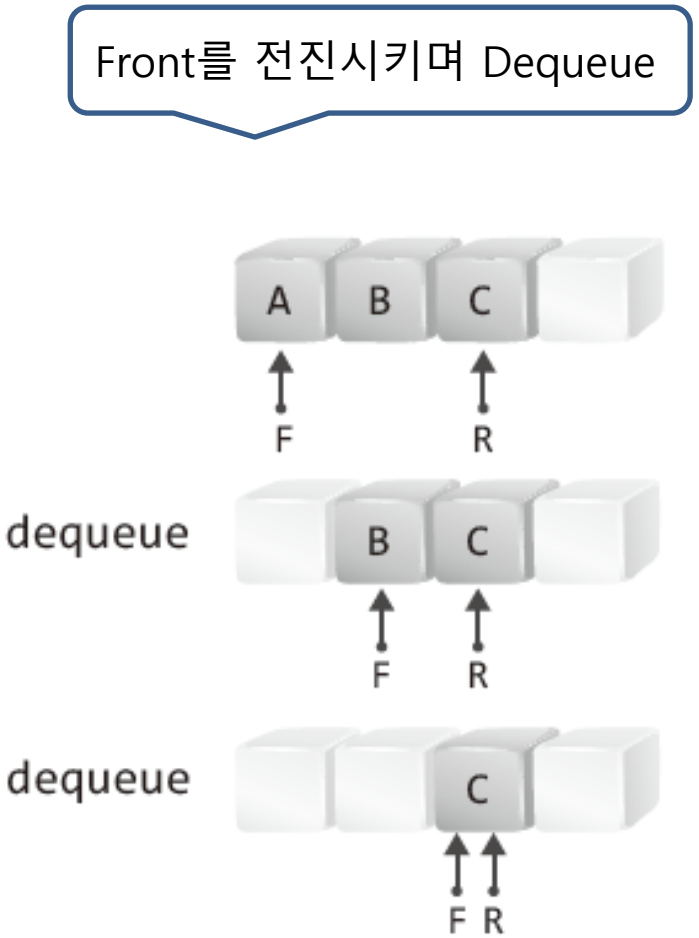
- `Data Dequeue(Queue * pq);`
  - 저장순서가 가장 앞선 데이터를 삭제한다.
  - 삭제된 데이터는 반환된다.
  - 본 함수의 호출을 위해서는 데이터가 하나 이상 존재함이 보장되어야 한다.

큐의 기본 연산

- `Data QPeek(Queue * pq);`
  - 저장순서가 가장 앞선 데이터를 반환하되 삭제하지 않는다.
  - 본 함수의 호출을 위해서는 데이터가 하나 이상 존재함이 보장되어야 한다.

큐의 배열기반 구현
(Front, Rear 2개의 index 사용)





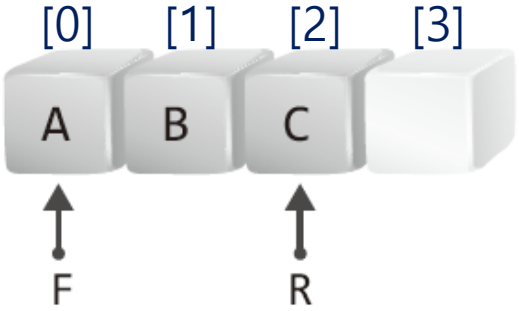
front : 다음 삭제할 위치. (초기값=-1)
rear : 최종 삽입한 위치. (초기값=-1)
Empty : rear+1 == front
Full : rear == N-1

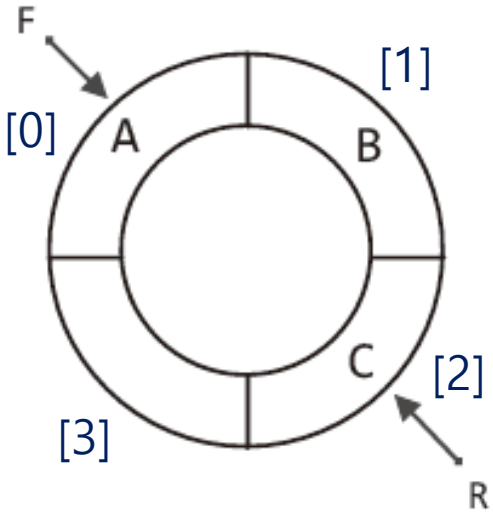
큐의 배열기반 구현의 문제점
(Front, Rear 2개의 index 사용)

배열에 여유 자리가 있지만 Rear는 끝까지 갔다!

데이터를 더 추가하기 위해서는 R을 인덱스가 0인 위치로 이동시키기.
F도 끝까지 가면 0인 위치로 이동시키기.
-> **'원형 큐'**

원형큐의 소개

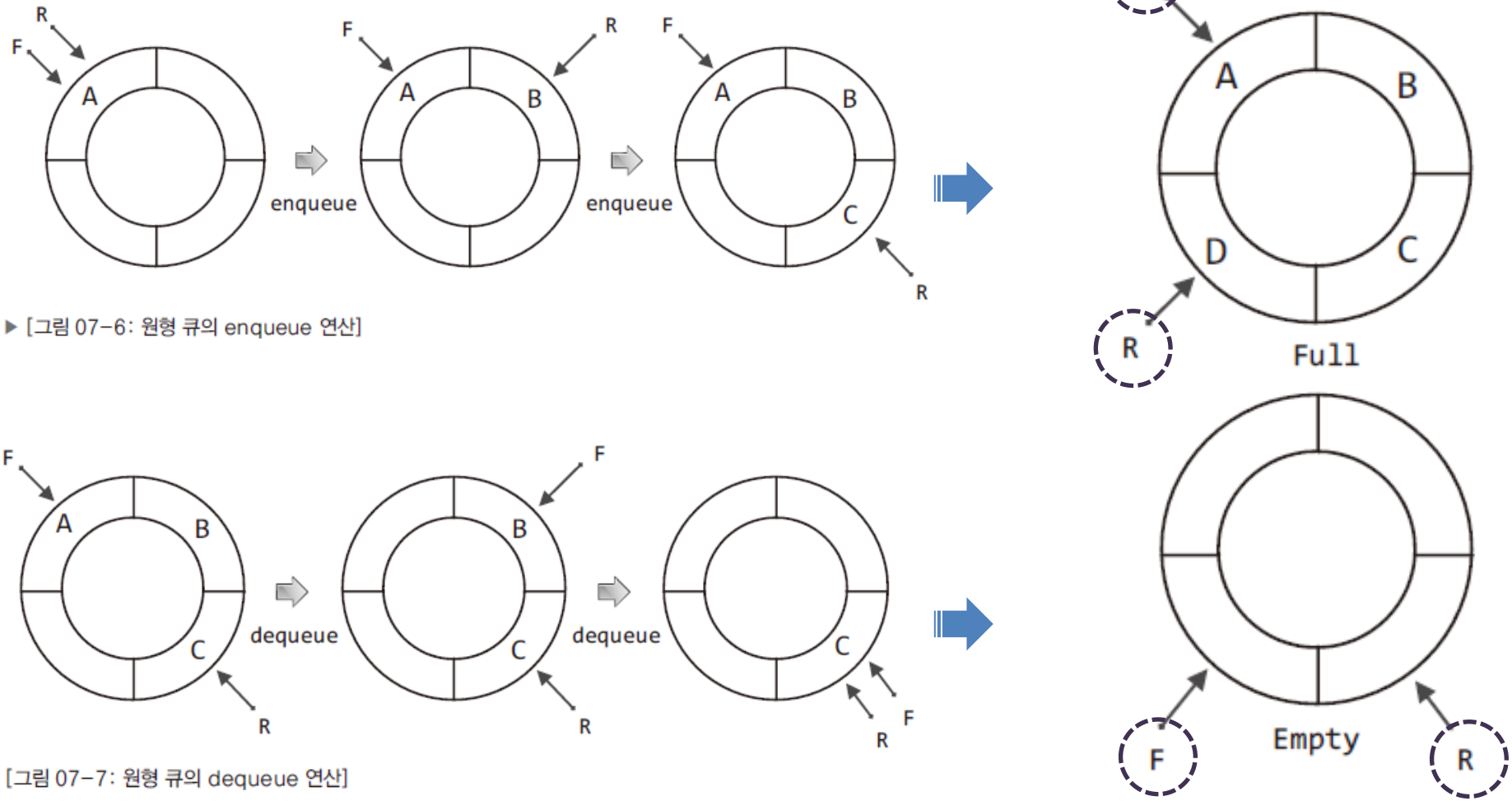




단순 배열 큐와 마찬가지로 R0이 이동하면서 Enqueue, f가 이동하면서 Dequeue.

다른 점은 배열 끝에서 다시 배열 맨 앞으로 계속 전진한다는 것.

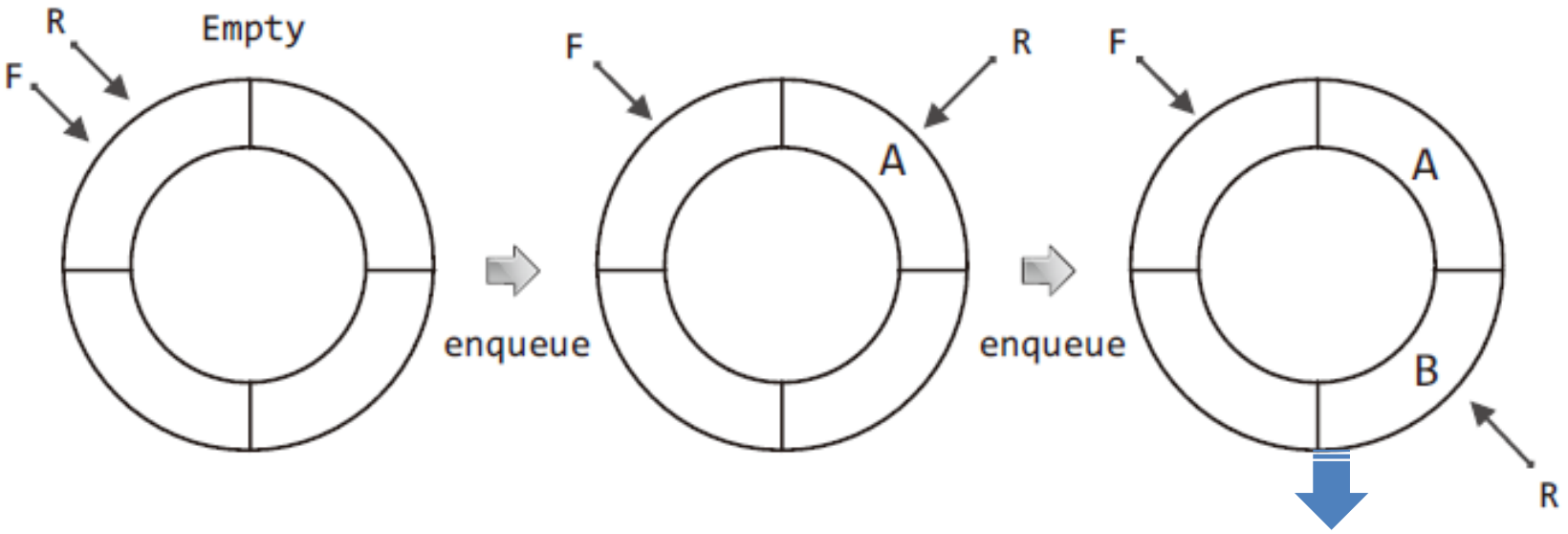
원형 큐의 문제점
(F와 R의 위치가 empty, full일 때 똑같다)



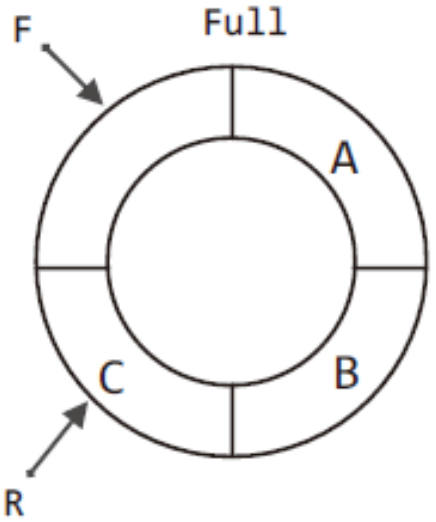
▶ [그림 07-6: 원형 큐의 enqueue 연산]

▶ [그림 07-7: 원형 큐의 dequeue 연산]

개선된 원형 큐
(배열 길이 N보다 1개 덜 채운 상태를 full로 간주!)



저장 공간 하나를 잃고 문제점을 해결!
(N=배열 크기)
front : 최종 삭제한 위치
rear : 최종 삽입한 위치
Empty : R == F
Full : (R+1)%N == F



원형큐의 배열기반 구현
(구조체 정의)

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

#define TRUE      1
#define FALSE     0

#define QUE_LEN100
typedef int Data;

typedef struct _cQueue
{
        int front;
        int rear;
        Data queArr[QUE_LEN];
} Queue;
```

원형큐의 배열기반 구현
(큐의 초기화, empty체크, index전진함수)

```
void QueueInit(Queue * pq) {
        pq->front = 0;
        pq->rear = 0;
}

int QIsEmpty(Queue * pq) {
        if (pq->front == pq->rear)
                return TRUE;
        else
                return FALSE;
}
```

```
int NextPosIdx(int pos) {
        if (pos == QUE_LEN - 1)
                return 0;
        else
                return pos + 1;
}
```

front, rear 를 전진시킬 때는 매번 원형으로 움직이게 해야 하므로 함수 사용.

원형큐의 배열기반 구현
(enqueue, dequeue)

```
void Enqueue(Queue * pq, Data data) {
	if (NextPosIdx(pq->rear) == pq->front) {
		printf("Queue Memory Error!");
		exit(-1);
	}

	pq->rear = NextPosIdx(pq->rear);
	pq->queArr[pq->rear] = data;
}

Data Dequeue(Queue * pq) {
	if (QIsEmpty(pq)) {
		printf("Queue Memory Error!");
		exit(-1);
	}

	pq->front = NextPosIdx(pq->front);
	return pq->queArr[pq->front];
}
```

원형큐의 배열기반 구현
(QPeek)

```
Data QPeek(Queue * pq)
{
	if (QIsEmpty(pq))
	{
		printf("Queue Memory Error!");
		exit(-1);
	}

	return pq->queArr[NextPosIdx(pq->front)];
}
```

원형큐의 배열기반 구현
(main 함수)

```
int main(void) {
        Queue q;
        QueueInit(&q);

        Enqueue(&q, 1);   Enqueue(&q, 2);
        Enqueue(&q, 3);   Enqueue(&q, 4);
        Enqueue(&q, 5);

        while (!QIsEmpty(&q))
                printf("%d ", Dequeue(&q));

        return 0;
}
```

Queue의 생성 및 초기화

데이터 넣기

데이터 꺼내기