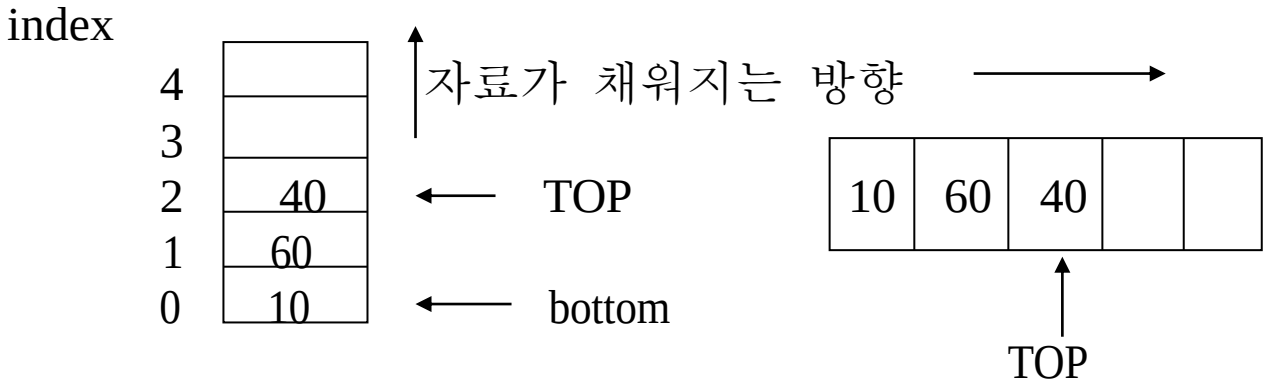
제 3 장 스택과 큐

1. STACK 의 정의

- 모든 데이터의 삽입과 삭제가 한쪽 끝에서만 일어나는 리스트 따라서 stack 내의 임의의 곳에 자료를 삽입 또는 삭제 불가
- stack 에 저장되는 자료형은 배열과 같이 동일 해야 함
- **LIFO(Last in First Out)** 리스트라고 한다.
- stack 은 배열(array)과 링크드 리스트(linked-list)로 구현된다.
- Stack 을 배열로 구현 하면 배열의 크기가 한정되어, 정적 스택 (static stack) 이라고 하고, 링크드 리스트로 구현하면 스택의 크기가 가변적이어서 동적 스택 (dynamic stack) 이라 한다.
- 순서리스트 A= $a_0, a_1, \ldots a_{n-1}$, $n \geq 0$ (a_0 는 bottom, a_{n-1} 은 top)

1.1. 기본 개념

그림은 10, 60, 40 세개 정수가 저장된 stack 의 구조이다. stack 의 맨 위를 top 이라고 하고 맨 아래를 bottom 이라고 하며, stack 의 크기는 5 이다.



10, 60, 40 의 3 개의 데이터가 저장된 스택이 모습, 스택의 크기는 5

1.2. Stack 알고리즘 [ADT]

structure *Stack*

objects: 0개 이상의 원소를 가진 유한 순서 리스트

functions:

Stack CreateS(max_stack_size) ::=

최대 크기가 max_stack_size인 공백 스택을 생성

void push(stack, item) ::=

if (IsFull(stack)) stack_full

else stack의 top에 item을 삽입하고 return

int pop(stack) ::=

if (IsEmpty(stack)) return

else stack top의 item을 제거해서 반환

int IsFull(stack, max_stack_size) ::=

if (stack의 원소수 == max_stack_size) return TRUE

else return FALSE

int IsEmpty(stack) ::=

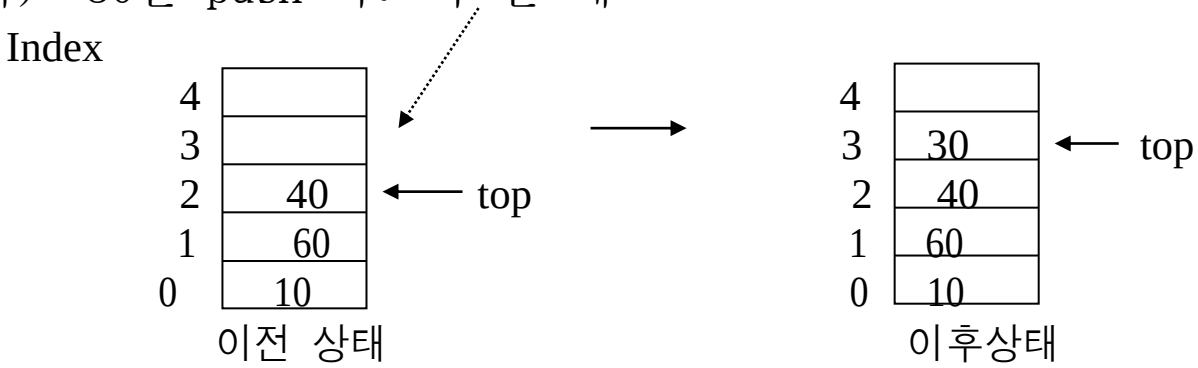
if (stack == CreateS(max_stack_size)) return TRUE

else return FALSE

```
const int stackSize = 4;
class Stack {
  private:
    int stack[stackSize];    int top;
  public:
    Stack() {top = -1;}
    void push(int val) {stack[++top] = val; }
    int pop() {return stack[top--]; }
    int isEmpty() {return top == -1;}
    int isFull() {return top == stackSize - 1;}
```

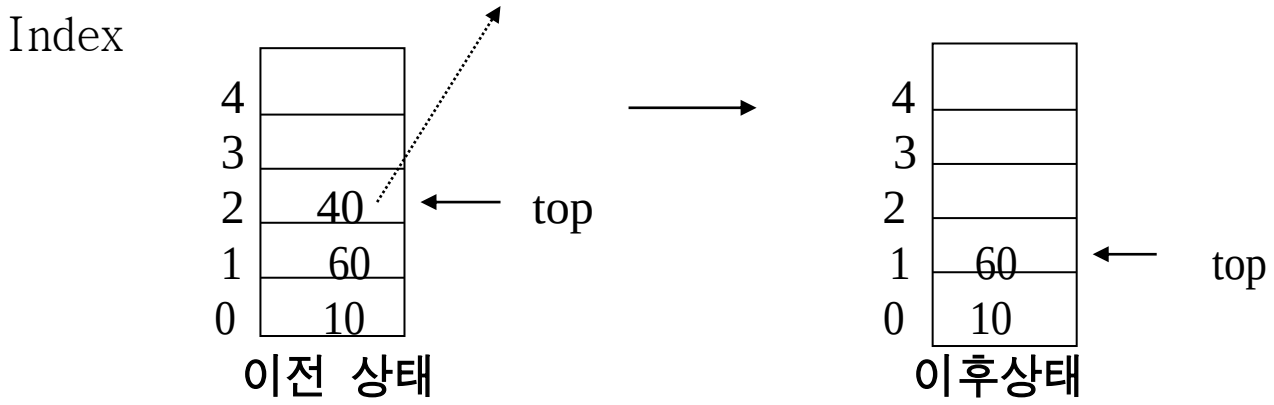
```
void push(int top, int item) {            // (PUSH)
    if (top ≥ stackSize - 1) {
        stack_full();     return;          // stack full message
    }
    stack[++top] = item;   }
```

예) 30을 push 하고자 할 때



```
int pop()                                 // (POP)
{   /* stack의 최상위 원소를 반환 */
    if (top == -1)                        // 공백 스택의미
        return stack_empty();             // 오류 message
    return stack[(top)--];   }
```

예) 40을 pop 하고자 할 때



스택 프로그램 example (Procedural, not Class)

```
/*************************************************************/
// File Name:          ARRAYSTK1.CPP                          //
// Description :    <Array Implementation of a Stack>         //
//                     Class 사용하지 않은 code                //
/*************************************************************/
#include <iostream>     // add string, stdlib.h
const int stackSize = 3;
int stack[stackSize];     int top;

void main()
{   void create_stack(), push(int), traverse_stack();
    int pop(); int isFull(), isEmpty();
    int num; char input[10];

    create_stack();

    while (1) {
        cout<<"Enter command(push, pop, traverse, exit):";
        cin >> input;

        if (strcmp(input, "push") == 0) {
            if (!isFull()) {
                cout << "Enter an integer to push => ";
                cin >> num;
                push(num);       }
            else     cout << "Stack is full!\n";
        }
        else if (strcmp(input, "pop") == 0) {
             if (!isEmpty()) {
                  num = pop();
                  cout << num << " is popped.\n";
             }
             else       cout << "Stack is empty!\n";
       }
```

```
            else if (strcmp(input, "traverse") == 0) displayStack();
            else if (strcmp(input, "exit") == 0) exit(0);
            else cout << "Bad Command!\n";
    }
}

void create_stack() {    top = -1; }      //stack create

int isFull() {
          if (top == stackSize - 1)      return 1;
           else        return 0;    }

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

void push(int num) {
          ++top;
          stack[top] = num;
 }

int pop() {
       return (stack[top--]);
 }

void displayStack()
{     int sp;
       if (isEmpty())    cout << "Stack is empty!" << endl;
       else {
                sp = top;           // sp = temporary pointer
                while (sp != -1) {
                          cout <<      stack[sp];      --sp;
                }
                cout << endl;
    }
 }
```

```
// classarraystack1.cpp      (Class)
// Stack implementation with arrays example

const int Stack_Size = 4;

class Stack {
   private:
         int stack[Stack_Size];      int top;
   public:
         Stack() {top = -1;}
         void push(int val) {stack[++top] = val;}
         int pop()      {return stack[top--];}
         int isEmpty() {return top == -1;}
         int isFull()    {return top == Stack_Size - 1;}
         void displayStack();
   };

void Stack::displayStack()
{
         int sp;        sp = top;
         while (sp != -1) { cout <<      stack[sp--]; }
         cout << endl; };

void main()
{        Stack s1;
         s1.push(10); s1.push(20); s1.push(30); s1.push(40);
         s1.displayStack();

         if (s1.isFull())      cout << "Stack is full\n";

         cout << "Pop: " << s1.pop() << endl;
         cout << "Pop: " << s1.pop() << endl;
         cout << "Pop: " << s1.pop() << endl;
         cout << "Pop: " << s1.pop() << endl;
         if (s1.isEmpty())
               cout << "Stack is   empty\n"; }
```

2. 수식의 계산 (Evaluation of Expression)

수식의 표현

- 중위 표기(*infix notation*) : a * b
- 후위 표기(*postfix notation*) : a b *
- 전위 표기(*prefix notation*) : * a b

1) Infix to Postfix conversion

1. Initialize stack

2. While NOT end-of-expression
   . Get next token

   . If token is
   "(" : then PUSH

   ")" : then POP and display elements in stack until
   left parenthesis("(") is encountered
   Pop left parenthesis

   Operator: if "**token** (**higher priority)** ≥ **top element"** then
   PUSH token onto stack

   else {**POP and Display** top element
   PUSH token onto Stack}

   Operand: Display

3. End-of-expression, then POP and Display until stack is empty

ex) 7 * 8 - (2 + 3) $ //infix to postfix conversion

Input	stack	output
7		7
*	*	7
8	*	7 8
-	-	7 8 *
(	(-	7 8 *
2	(-	7 8 * 2
+	+ (-	7 8 * 2
3	same	7 8 * 2 3
)	-	7 8 * 2 3 +
$		7 8 * 2 3 + -

Priority:

)	3
*, /	2

+, -	1
(	0

ex) A*(B+C)*D$ => ABC+*D* 연습할것

---

isp(in-stack precedence)와 icp(incoming precedence)

precedence stack[MAX_STACK_SIZE];

/* isp 와 icp 배열 -- 인덱스는 연산자의 우선순위 값 */

static int **isp[ ]** = {……};

static int **icp[ ]** = { …… …….. };

```
void postfix (void) {
    int top, i = 0;    stack[0] = eos;    //end-of-string

    get_expression(); //get a token from input

    while ((token = line[i]) != '\0'; )    {        // do until end-of-string
       if (token == operand)      print (token);    // print symbol

        else if (token == lparen)      Push(top, token)    // if "("

       else if (token == rparen){            //    if ")"
              while (stack[top] != lparen)      // do until "(" appears
                 print (POP(top));    /*    POP & print
               POP(top);           /* remove "(" */
       }
       else { // if operator
          if (isp[stack[top]]  ≤  icp[token]) Push(top, token); //Push token
          else { print (POP(top))
                    Push(top, token); }    // push token
        }
       }
      while ((token=POP(top)) != eos)      print (token);
```

---

2) **Postfix Evaluation (후위표기)**

. Infix 표기는 가장 보편적인 수식의 표기법,

. 대부분의 compiler 는 후위 표기법을 사용한다.

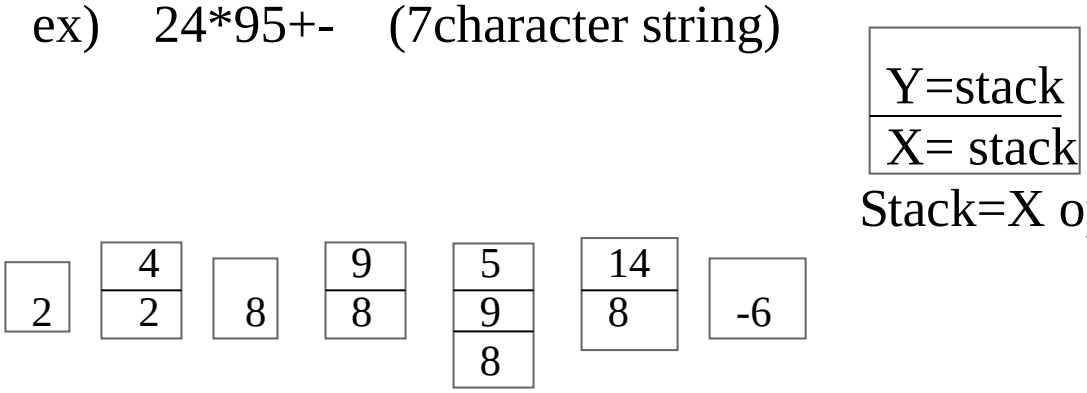
- 괄호(parenthesis) 사용 안 함. – 계산이 간편
- 연산자의 우선순위 없음 (L -> R 순서의 계산)

● Algorithm:

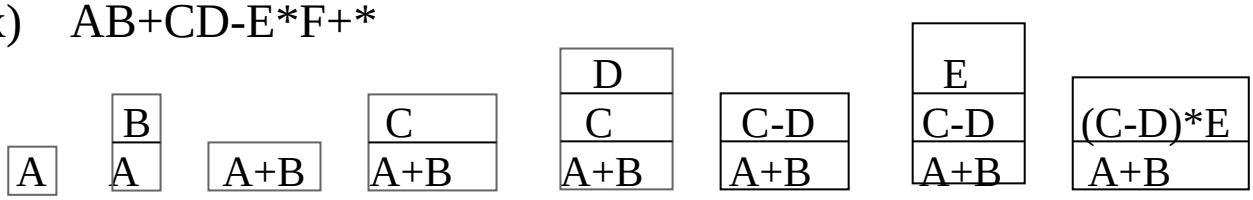
1. Initialize stack
2. Repeat until end-of-expression
   - . Get next token
   - . If **“token = operand”** then PUSH onto Stack
     else “**token=operator”**
     - . POP two operands from stack
     - . Apply the operator to these
     - . Push the results onto stack

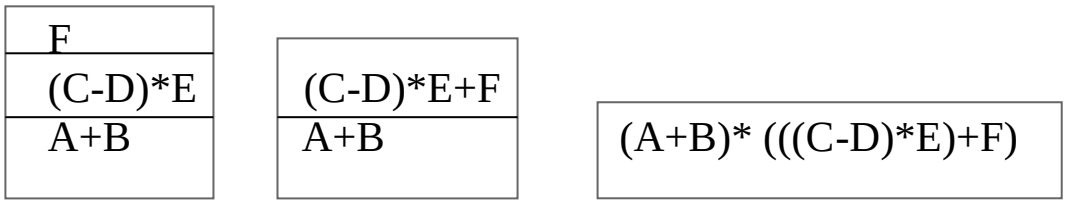
3. When end-of-expression, its value(result) is on top of Stack

ex) 24*95+- (7character string)



ex) AB+CD-E*F+*





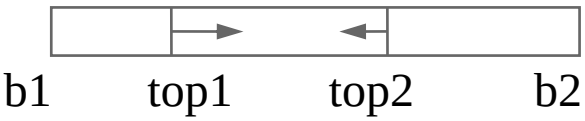
* Postfix Evaluation

```
int stack[MAX_STACK_SIZE];

int eval(void)   {
    int op1,op2;
    int n = 0; /* 수식 스트링을 위한 카운터 */
    int top = -1;
    token = get_token (&symbol,    &n);

    while (token != eos) {    /* not end of string */

       if ("token == operand")    Push(&top, symbol-'0'); /*convert to num.

       else { /* if operator, then, 연산수행 후, 결과를 stack에 push
              op2 = POP(&top);       /* 스택 삭제 (POP)*/
             op1 = POP(&top);

             switch(token) {
                  case '+':    PUSH(&top, op1+op2);      break;
                  case '-':    PUSH(&top, op1-op2);      break;
                  case '*':    PUSH(&top, op1*op2);    break;
                  case '/':    PUSH(&top, op1/op2);      break;
             }
        }
        token = get_token(&symbol, &n);
    }
    return POP(&top); /* 결과를 반환 */
}
```

---

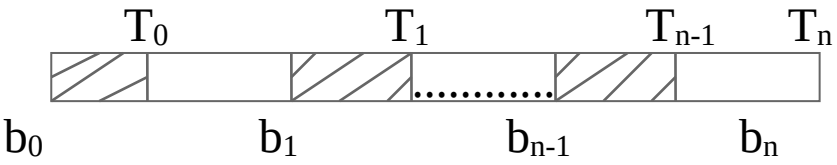
3. 다중 스택

ex) 하나의 array 에 두개의 stack 사용할 경우



. PUSH 때 top 증가
. Stackfull check 는 top1=top2 를 check 하면 된다

ex) n 개의 stack



. $Stack_k$ is empty?: top[k] = bottom[k]
. $Stack_k$ is Full?: top[k] = bottom[k+1]

- Add

```
. if (top[k] = bottom[k+1]) then
                stack-full(k);
  else    stack[top[k]] = item;
```

- Delete

```
. if top[k] = bottom[k]
          return stack-empty(k);
  return    stack[top[k]-1];
```

4. QUEUE

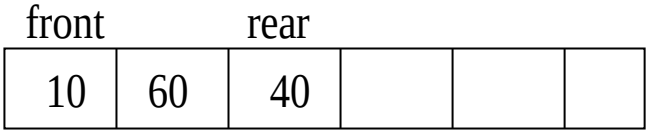
* 데이터의 삽입과 삭제는 한쪽 끝(rear)과 다른 한쪽 끝(front)에서 발생한다. (임의의 곳에 자료를 삽입/삭제 불가능)

- Stack: 1 pointer, Queue: 2 pointer

$Q = (a_1, a_2, \ldots\ldots a_n)$

Front rear

(먼저 add 된 노드) (나중 add된 노드)



● 스택 과 큐에서의 삽입/삭제 연산

항목 / 자료구조	삽입 연산		삭제 연산	
	연산자	삽입 위치	연산자	삭제 위치
스택	Push	top	Pop	top
큐	enQueue	rear	deQueue	front

1) 자료형은 배열처럼 동일 해야 한다.
2) FIFO(First-In-First-Out) 리스트라고 한다.(제일 먼저 입력된 것이 제일먼저 제거됨)

3) 큐는 배열과 링크드 리스트로 구현된다
4) 배열로 구현시는 정적큐(static queue) 라고 하고, 링크드 리스트로 구현시에는 동적 큐(dynamic queue) 라고 한다.

[큐 추상 데이타 타입]

---

structure Queue

objects: 0개 이상의 원소를 가진 유한 순서 리스트

functions: max_queue_size∈positive integer

Queue CreateQ(max_queue_size) ::=
최대 크기가 max_queue_size인 공백 큐를 생성

Queue Enqueue(queue, item) ::=
if (IsFull(queue)) queue_full
else queue의 뒤에 item을 삽입하고 queue를 반환

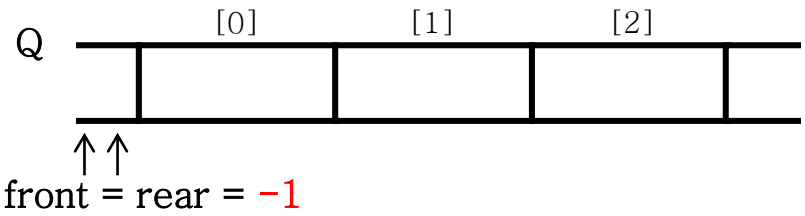
int Dequeue(queue) ::=
if (IsEmpty(queue)) return
else queue의 앞에 있는 item을 제거해서 반환

Boolean IsFullQ(queue, max_queue_size) ::=
if (queue의 원소수 == max_queue_size) return TRUE
else return FALSE

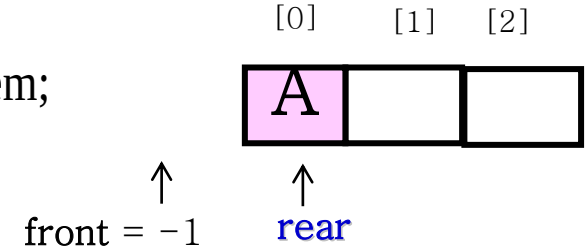
Boolean IsEmptyQ(queue) ::=
if (queue == CreateQ(max_queue_size)) return TRUE
else return FALSE

```
 class Queue {
 private:
        int* arrayOfData;    int front;    int rear;
        const int sizeQueue;
public:
        Queue(int size);
        virtual ~Queue();
        void enQueue(int value);
        int deQueue();
        bool isFull();
        bool isEmpty();
        void print();
};
```

```
void create_queue()    {    // if front==rear, Queue is empty
     front = -1;      rear = -1;
 }
```



```
void Enqueue (int item)              /* queue에  item을  삽입  */
{
        if (rear == MAX_QUEUE_SIZE-1) {
                queue_full();
                return;}
          queue[++rear] = item;
        }
```



```
int dequeue ()       /* queue의  앞에서  원소를  삭제  */
{
        if (front == rear)
                return queue_empty(); /*  에러  key를  반환  */
        return queue[++front];
}
```

```
int queue_full() {
   if (rear == queue_size –1)    return true;
   else    return false;
 }
int queue_empty()    {
  if (front ==rear) return true;
    else    return false;
}


void print_queue() {
          int i;

          if (queue_empty())
                cout << "Queue is Empty!\n";
          else {
              i = front + 1;
              cout << "-- Print Queue --\n";
              while (i <= rear) {
                        cout <<    queue[i];
                        i = i + 1;
              }
          }
}
```

예제 [작업 스케쥴링]: 운영체제에 의한 작업 큐(job queue)의 생성

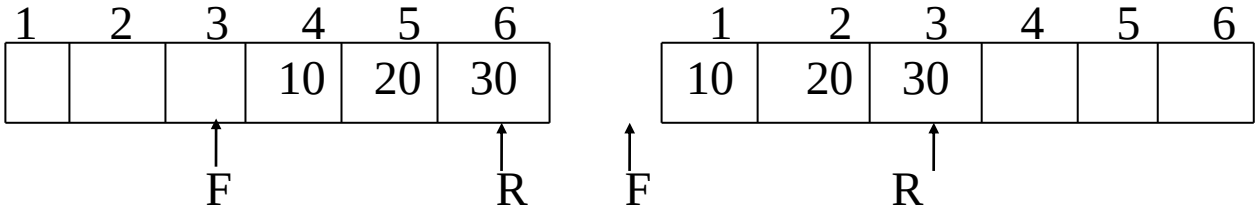
front	rear	Q[0] Q[1] Q[2] Q[3]	설 명
-1	-1		공백큐
-1	0	J1	Job 1의 삽입
-1	1	J1 J2	Job 2의 삽입
-1	2	J1 J2 J3	Job 3의 삽입
0	2	J2 J3	Job 1의 삭제
1	2	J3	Job 2의 삭제

* Problems with Queue

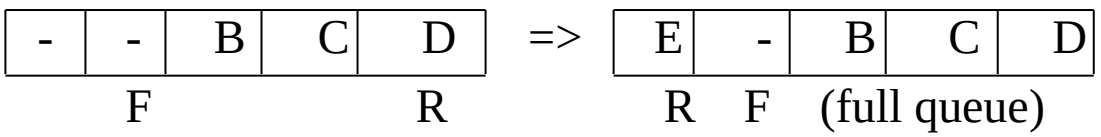
위의 예에서 보듯이 작업이 큐에 들어오고 나감에 따라 큐는 전체적으로 오른쪽으로 shift 된다. 즉, rear index 가 큐의 maxsize-1 과 동일하게 되어 큐는 full 이 된다.

■ **해결책:**

1) Front = 0 이 되도록, 전체 Q 를 왼쪽으로 이동
(Q 에 많은 원소 있을 때는 상당한 처리시간 필요)



2) 환상 Q (Circular Queue) 이용

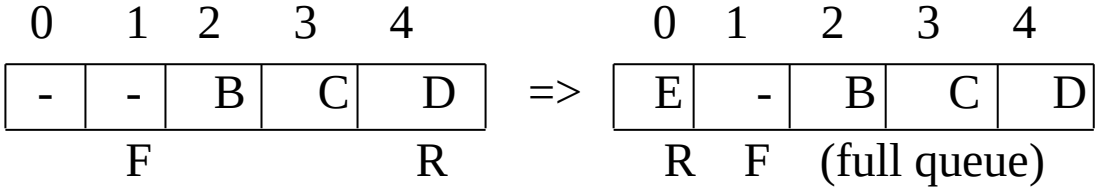


5. **Circular Queue** (원형 큐)

- 동작은 front 와 rear 인덱스를 시계방향으로 이동

```
void enqueue (int item)
{
     rear = (rear+1) % QUEUESIZE;
    if (front == rear) cout << "Queue is full";
    else       queue[rear] = item;
}

void dequeue ()
{
   if (front == rear)    cout<< “Queue is empty”;
   else {
        front = (front+1) %queuesize;
         return queue[front];
}
```

● Circular Queue Issue : 아래 예제에서, Enqueue ‘E’ 다음에는 rear=front 가 되어 Queuefull 이 되기 때문에 큐 의 one space 가 항상 비어있게 된다.



⇨ Insert ‘G’: R=(0+1) mod n = 1 => (F=R), Queuefull 발생, Enqueue 불가능

- Alternative Method#1 - Flag 사용

Queue 에 데이터가 하나라도 있으면 flag=1, 즉, enqueue 때마다 flag 를 1 로 set. Dequeue 시에는 dequeue 후 reset flag to 0.

```
void enqueue(int item)
{
if (front == rear)&& (flag==1)
   cout << "Queue is full";
else {
   Queue[rear] = item;
   rear=(rear+1) % QueueSize;
  }
}
```

```
int dequeue()
{
   int item;
  if (front == rear)&&(flag==0){
       cout << "Queue is empty";
        exit(-1);
   }
  else {
     item = Queue[front];
  front= (front +1) % QueueSize;
    flag=0;
    return item;
  }   }
```

Alternative Method#2 - Count 사용

enqueue 때마다 ++count, Dequeue 때마다 –count

If count ==0 then empty, if count==Queuesize then Full.