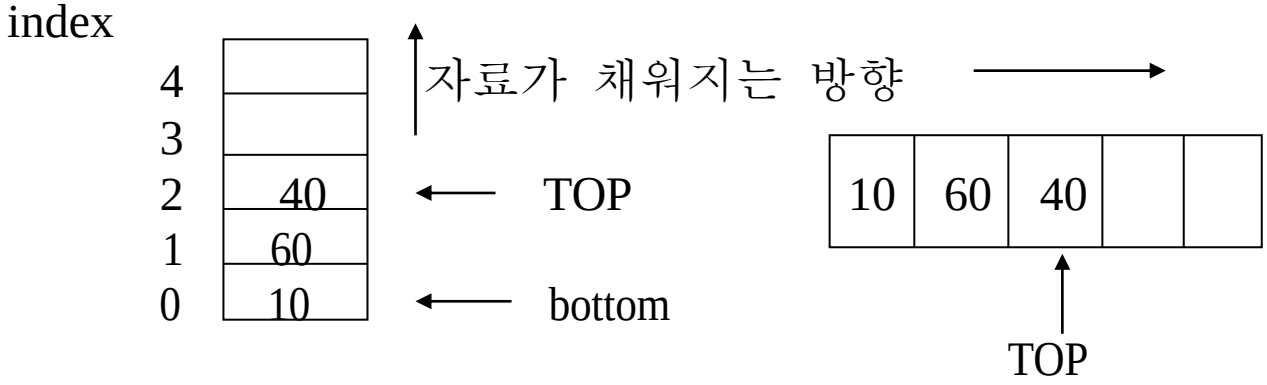
제 3 장 스택(Stack)과 큐(Queue)

1. STACK 의 정의

- 데이터의 삽입/삭제가 한쪽 끝에서만 일어나는 순서리스트. 따라서 stack 내의 임의의 곳에 자료를 삽입 또는 삭제 불가
  . 순서 리스트 A= $a_0, a_1, \ldots a_{n-1}$, $n \geq 0$ (a_0 는 bottom, a_{n-1} 은 top)
- 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, linked-list 로 구현하면 스택의 크기가 가변적이어서 동적 스택 (dynamic stack) 이라 한다.

1.1. 기본 개념

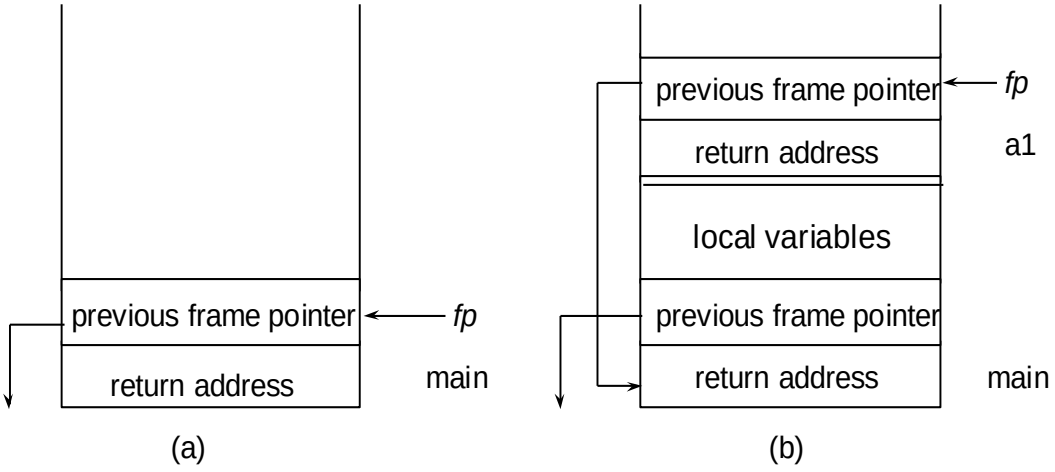
그림은 10, 60, 40 세개 정수가 저장된 stack 의 구조이다. stack 의 맨 위를 top 이라고 하고 맨 아래를 bottom 이라고 하며, stack 의 크기는 5 이다.



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

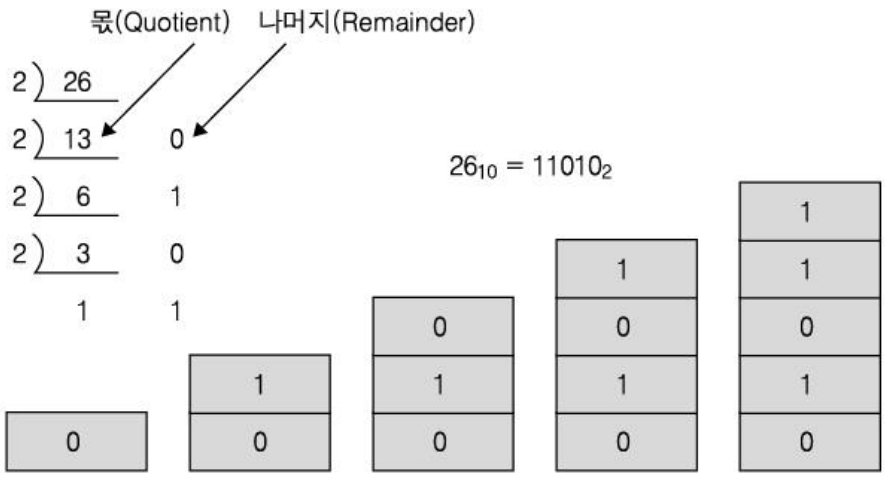
Ex) system stack

- 프로그램 실행 시 함수 호출을 처리 / 순환 호출의 경우
- 함수 호출 시 activation record 또는 스택 프레임(stack frame) 구조 생성
  . 이전 스택 프레임에 대한 포인터, 복귀 주소, 지역 변수, 매개 변수



(함수 a1 호출 전.후 시스템 스택의 예)

Ex) 진법의 변환 (예: 10 진수에서 2 진수로)



1.2 스택 추상데이터 타입 (Stack [ADT])

```
class Stack {
    // objects: 0개 이상의 원소를 가진 유한 순서 리스트
public:
    Stack (int MaxStackSize = DefaultSize);
      // 최대 크기가 MaxStackSize인 공백 스택을 생성
    Boolean IsFull();
     // 스택 내에 있는 원소의 수가 스택의 최대 크기와 같은
     // 경우 TRUE (1), 그렇지 않은 경우 FALSE (0)을 반환
    void Add(int item);     //PUSH
     // IsFull()이 TRUE이면 StackFull()을 호출, else
     // 그렇지 않으면 스택의 top에 item을 삽입
    Boolean IsEmpty();
     // 스택의 원소 수가 0인 경우 TRUE (1), else FALSE (0)을 반환
    int Delete( );     //POP
     // IsEmpty()가 TRUE이면 StackEmpty()을 호출하고 0을 반환,
     // FALSE 이면 스택의 top 원소를 제거하고 그 포인터를 반환
};
```

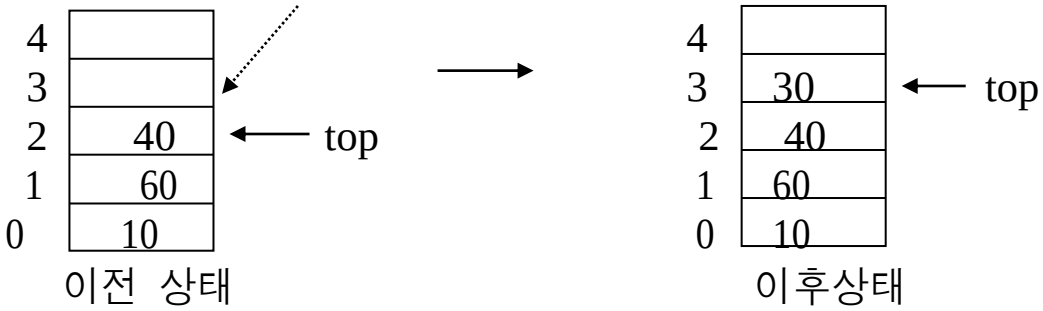
- **Stack 구현**
  - 1차원 배열 stack[Maxsize] 사용,
  - i번째 원소는 stack[i-1]에저장
  - 변수 top은 스택의 최상위 원소를 가르킴 (초기: top = -1)

```
const int MaxSize = 4;

class Stack {
   private:
      int stack[MaxSize];    int top;
   public:
      Stack() {top = -1;}
```

```
    void push(int val) {stack[++top] = val; }
    int pop() {return stack[top--]; }
    boolean isEmpty() {return top == -1;}
    boolean isFull() {return top == MaxSize - 1;}
}
```

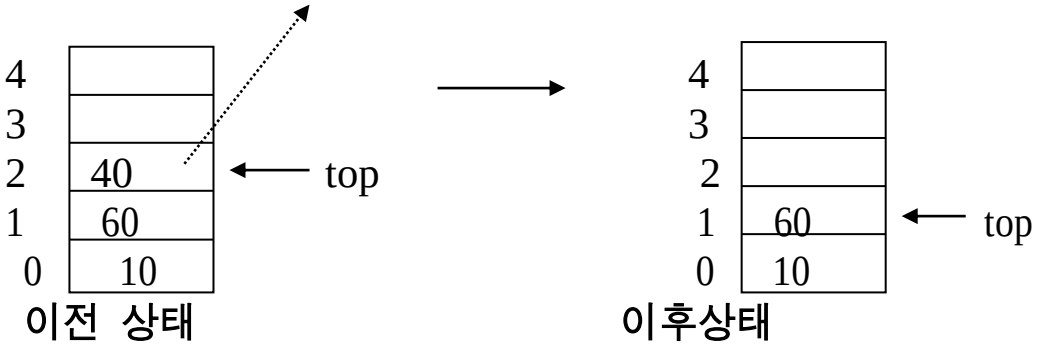
```
void push(int item) {              // (PUSH)
    if (top ≥ MaxSize - 1) {
        stack_full();     return;      // stack full message
    }
    stack[++top] = item;   }
```

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 **Sample Code** #1 (Procedural, not Class)

```
/*****************************************************/
// File Name:          ARRAYSTK.CPP                   //
// Description :   <Array Implementation of a Stack>  //
//                  Class 사용하지 않은 code          //
/*****************************************************/
#include <iostream>
const int stackSize = 3; using namespace std;
int stack[stackSize];    int top;

void create_stack(); void push(int num); int pop();
int isFull(); int isEmpty(); void displayStack();

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

int isFull() {
        if (top == MaxSize - 1)
                    return TRUE;
         else       return FALSE;
 }

int isEmpty() {
        if (top == -1)      return TRUE;
        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;
     }
  }
```

```
// sample Code #2        classarraystack1.cpp      (Class)
// Stack implementation with arrays example.
#include <iostream>
using namespace std;
const int StackSize = 4;

class Stack {
   private: int stack[StackSize];      int top;
   public:
         Stack() {top = -1;} //top 을  -1 로  초기화  하여  스택생성
         void push(int val) {stack[++top] = val;}
         int pop()      {return stack[top--];}
         int isEmpty() {return top == -1;}
         int isFull()    {return top == StackSize - 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"; }
```

```
// sample Code #3
// default constructor
// typedef int Type

#include <iostream>
#include <iomanip>        //setw
#include <assert.h>       // assert
#include <stdlib.h>       // for exit
using namespace std;

const int Stack_Size = 4;
typedef int Type;

class Stack {
private:
   int top;
   int size;
   Type *stack;
public:
   Stack(int sz = Stack_Size);
    ~Stack();
    void push(Type val);
   Type pop();
   int isEmpty() {return top == -1;}
   int isFull() {return top == size - 1;}
   void displayStack();
};


Stack::Stack(int sz) {
   stack = new Type[sz];
   assert(stack != 0);
   top = -1;
   size = sz;
}

Stack::~Stack()
{delete stack;}
void Stack::push(Type val) {
   if (isFull()) {
      cerr << "Error: push on full stack" << endl;
     exit(-1);
   }
    stack[++top] = val;
 }


Type Stack::pop() {
 if (isEmpty()) {
    cerr << "Error: pop on empty stack" << endl;
     exit(-1);
 }
   return stack[top--];
}

void Stack::displayStack() {
   int sp;
   for (sp = top; sp != -1; --sp)
   cout << setw(10) << stack[sp];
   cout << endl;
};


void main(){
 Stack s1;

s1.push(10);
s1.push(20);
s1.push(30);
s1.push(40);
s1.displayStack();

cout << "Pop: " << s1.pop() << endl;
cout << "Pop: " << s1.pop() << endl;
cout << "Pop: " << s1.pop() << endl;
cout << "Pop: " << s1.pop() << endl;
// empty stack test
cout << "Pop: " << s1.pop() << endl;
}
```

// **sample Code #4**

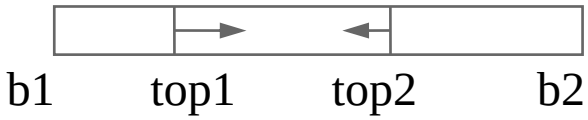
classarraystack4.cpp

// template classes

```
#include <iostream>
#include <iomanip>
#include <assert.h>
#include <stdlib.h>
using namespace std;

const int Stack_Size = 4;

template <class Type>
class Stack {
  private:
     int top;
     int size;
     Type *stack;
public:
   Stack(int sz = Stack_Size);
   ~Stack();
   void push(Type val);
   Type pop();
   int isEmpty() {return top == -1;}
   int isFull() {return top == size - 1;}
   void displayStack();
};


template <class Type>
Stack<Type>::Stack(int sz)    {
   stack = new Type[sz];
   assert(stack != 0);
   top = -1;
   size = sz;
}

template <class Type>
Stack<Type>::~Stack() {
   delete stack;
}

template <class Type>
void Stack<Type>::push(Type val){

   if (isFull()) {
       cerr << "Error: push on full stack" << endl;
   exit(-1);
   }
   stack[++top] = val;
}

template <class Type>
Type Stack<Type>::pop() {
   if (isEmpty()) {
    cerr << "Error: pop on empty stack" << endl;
   exit(-1);
   }
     return stack[top--];
}

template <class Type>
void Stack<Type>::displayStack() {
int sp;
  for (sp = top; sp != -1; --sp)
      cout << setw(10) << stack[sp];
  cout << endl;
};

void main() {
Stack<int> s1;

s1.push(10); s1.push(20);
s1.push(30); s1.push(40);
s1.displayStack();

cout << "Pop: " << s1.pop() << endl;
cout << "Pop: " << s1.pop() << endl;
cout << "Pop: " << s1.pop() << endl;
cout << "Pop: " << s1.pop() << endl;
cout << "Pop: " << s1.pop() << endl;
//test empty stack
cout << "Pop: " << s1.pop() << endl;
}
```

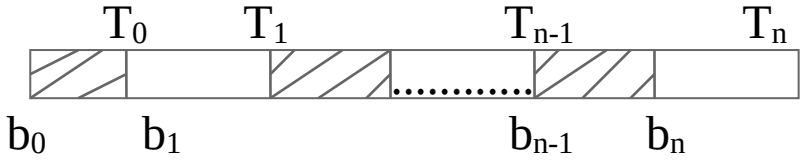
1.3 다중 스택

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];
```

2. QUEUE

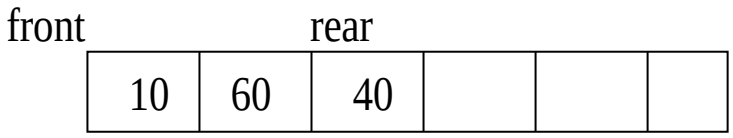
* 데이터의 삽입과 삭제는 한쪽 끝(rear)과 다른 한쪽 끝(front)에서 발생한다. (임의의 곳에 자료를 삽입/삭제 불가능)

- Stack: 1 pointer, Queue: 2 pointer

$Q = (a_0, a_2, \ldots\ldots a_{n-1})$

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 리스트(Linked List)로 구현된다
4) 배열로 구현시는 정적큐(static queue) 라고 하고, 링크드 리스트로 구현시에는 동적 큐(dynamic queue) 라고 한다.

[Queue(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

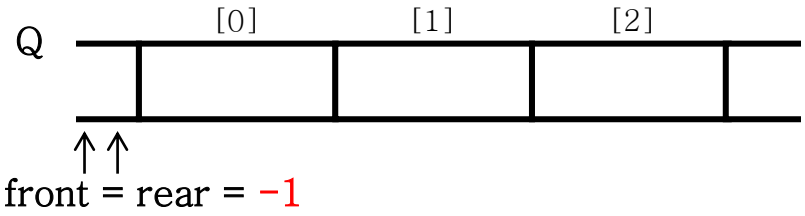
int Dequeue(queue) ::=
if (IsEmpty(queue)) return
else queue의 앞에 있는 item을 제거해서 반환

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

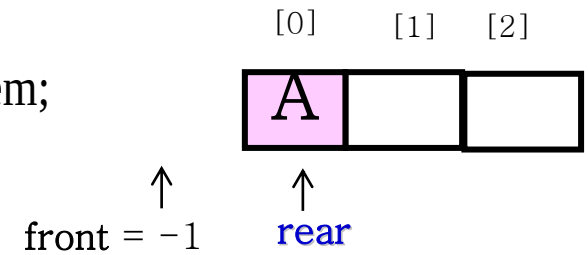
Boolean IsEmptyQ(queue) ::=
if (queue == CreateQ(max_queue_size)) return TRUE
else return FALSE

```
 class Queue {
 private:
        int front;    int rear; …//
        const int QueueSize;
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;      //0 or -1로  초기화
  }
```



```
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];     //먼저  인덱스증가  후  삭제,반환
}
```

```
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() {

          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;
              }
          }
  }
```

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

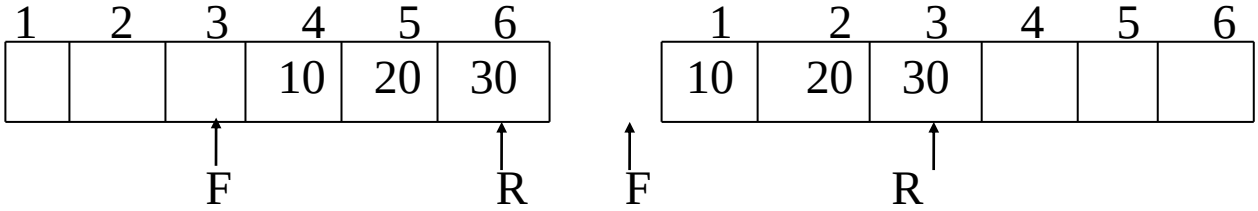
front	rear	Q[0]	Q[1]	Q[2]	Q[3]	설 명
-1	-1					공백큐
-1	0	J1				Job 1의 삽입
-1	1	J1	J2			Job 2의 삽입
-1	2	J1	J2	J3		Job 3의 삽입
0	2		J2	J3		Job 1의 삭제
1	2			J3		Job 2의 삭제

* Problems with Queue(문제점)

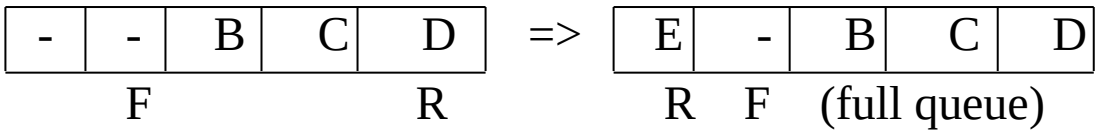
위의 (예)에서 보듯이 작업이 큐에 들어오고 나감에 따라 큐는 전체적으로 오른쪽으로 shift 된다. 즉, rear index 가 큐의 maxsize-1 과 동일하게 되어 큐는 full 이 된다.

■ **해결책:**

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



2) 환상 queue (Circular Queue) 이용

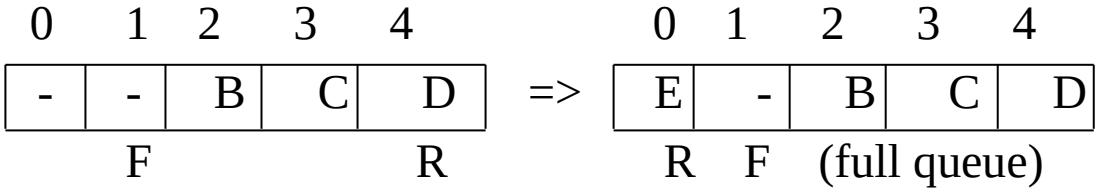


3. Circular Queue (원형 큐)

- 배열 queue[maxsize]을 원형으로 취급
- 동작은 front 와 rear 인덱스를 시계방향으로 이동
- 초기: front==rear==0, 공백: front==rear

```
void enqueue (int item)
{
     rear = (rear+1) % QUEUESIZE;
    if (front == rear) {cout << "Queue is full"; exit(-1)}
    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 데이터 enque/dequeue 할때 flag 사용,

- Enqueue 때마다 flag 를 1 로 set.
- Dequeue 후 reset flag to 0.
- Front==Rear==0

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

```
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.

Queue sample Code #1

```
include <iostream>
  …….
const int QueueSize = 10;

template <class Type>
class Queue {
private:
      int front; int rear; int flag;
      Type queue[QueueSize];
public:
      Queue() {front = 0; rear = 0;
            flag = 0;}
      void enqueue(Type item);
      Type dequeue();
};
template <class Type>
void  Queue<Type>::enqueue(Type
item)   {
if ((front == rear) && (flag == 1)) {
      cout << "Queue is full";
      exit(1);
}
else {
      queue[rear] = item;
      rear = (rear + 1) % QueueSize;
      flag = 1;
      }
};

template <class Type>
Type Queue<Type>::dequeue() {
Type item;
if ((front == rear) && (flag == 0)) {
cout << "Queue is empty";exit(1);
}
else {
   item = queue[front];
   front = (front + 1) % QueueSize;
   flag = 0;
   }
   return item;
};

void main()
{
Queue<int> q1;

q1.enqueue(10);cout << "Eneueue : 10"
q1.enqueue(20);cout << "Eneueue : 20"
q1.enqueue(30);cout << "Eneueue : 30"
q1.enqueue(40);cout << "Eneueue : 40"

cout << endl;
cout << "Dequeue : "<<q1.dequeue()<<endl;
cout << "Dequeue : "<<q1.dequeue()<<endl;
cout << "Dequeue : "<<q1.dequeue()<<endl;
cout << "Dequeue : "<<q1.dequeue()<<endl;
}

/*
Eneueue : 10
Eneueue : 20
Eneueue : 30
Eneueue : 40

Dequeue : 10
Dequeue : 20
Dequeue : 30
Dequeue : 40
Press any key to continue
*/
```

4. 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

4.1 Infix to Postfix conversion

1. Initialize stack

2. While NOT end-of-expression
   . Get next token
   . If token is
   "(" : then PUSH stack

   ")" : 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

● Postfix notation 의 마지막에 '\0'넣기

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

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

ex) A*(B+C)*D$ => ABC+D** 연습할 것

ex) 3*4+5*6

* 연산자 우선순위:

연산자	우선순위		과제용 연산자	순위
Not, ~	6		)	3
*, /	5		*, /	2
+, -	4		+, -	1
<, >, ≥ , ≤ , ≠	3		(	0
AND, &&	2			
OR, ‖	1			

*** 수식 계산 알고리즘**

(1) 1단계 : infix 수식을 postfix 수식으로 바꾼다.
(2) 2단계 : postfix 수식을 stack을 이용하여 계산한다.

```
void postfix (void) { // 1 단계  infix-> postfix conversion
    int top = 0;    stack[0] = eos;    //end-of-string 마커표시
    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)**

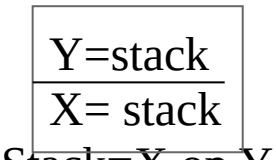
. Infix 표기는 가장 보편적인 수식의 표기법,
. 대부분의 compiler 는 후위 표기법을 사용한다.
- 괄호(parenthesis) 사용 안 함. - 계산이 간편
- 연산자의 우선순위 없음 (L -> R 순서의 계산)

● Algorithm:

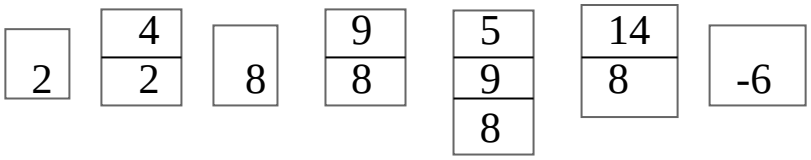
1. Initialize stack
2. Repeat until end-of-expression
    . Get next token
    . If **"token = operand"** then PUSH onto Stack
      else "**token=operator"**
        . POP two operands from stack (OP2 & OP1)
        . Apply the operator to these (OP1 OP OP2)
        . Push the results onto stack

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

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

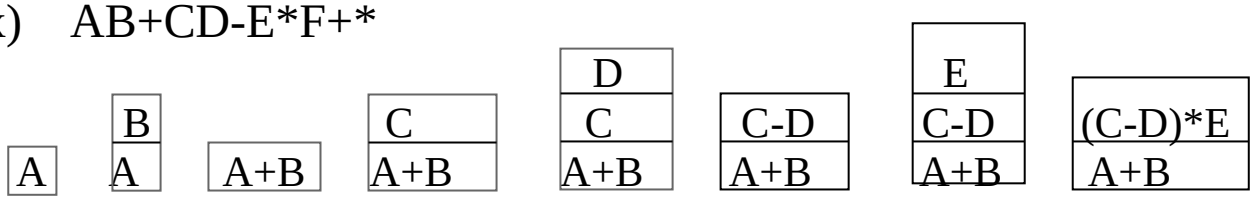


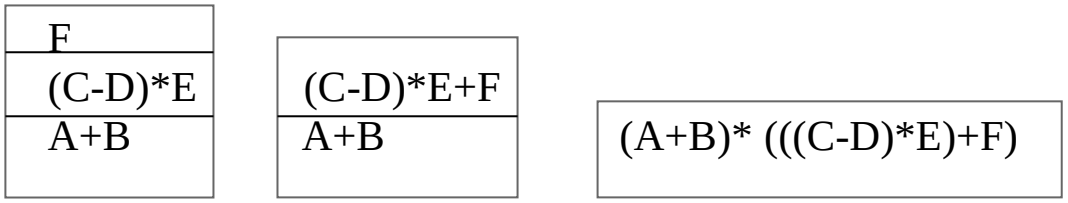
Stack=X op Y



2*4 9+5 8-14 = -6

ex) AB+CD-E*F+*





* Postfix Evaluation

```
int stack[MAX_STACK_SIZE];

int eval(void)   {
    int op1,op2;
    int n = 0; /*  수식  string을  위한  counter */
    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  */
}
```

<< 미로 문제 >>

● 미로는 1≤i≤m이고 1≤j≤p인 이차원 배열 maze[m][p]로 표현
1 : 통로가 막혀 있음　　0 : 통과할 수 있음

입구

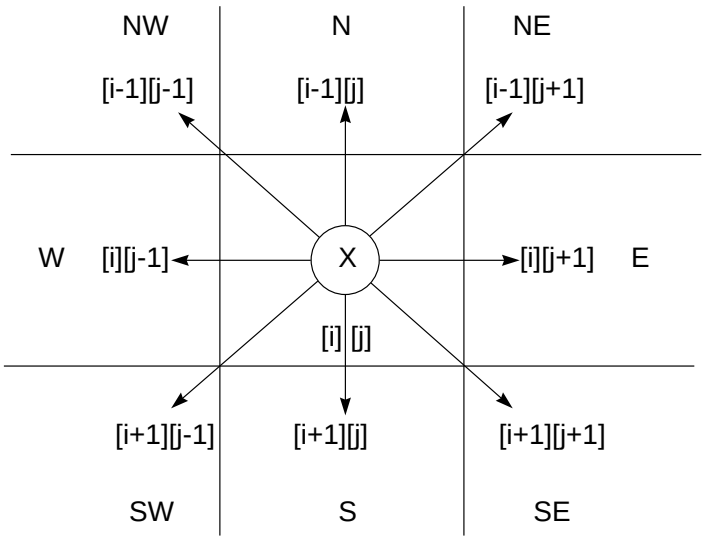
```
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 0 0 1 1 0 1 1 1 0 0 1 1 1
1 1 1 0 0 0 0 1 1 1 1 0 0 1 1
1 1 0 1 1 1 1 0 1 1 0 1 1 0 0
1 1 0 1 0 0 1 0 1 1 1 1 1 1 1
1 0 1 1 0 1 1 1 0 1 0 0 1 0 1
1 0 1 1 0 1 1 1 0 1 0 0 1 0 1
1 1 1 1 1 0 0 1 1 1 1 1 1 1 1
1 1 0 0 0 1 1 0 1 1 0 0 0 0 1
1 0 1 1 1 1 1 0 0 0 1 1 0 0 1
1 1 0 0 1 1 1 1 1 1 1 1 1 1 0
```

출구

(예제 maze)

maze[m+2][p+2]: 미로 주변은 1로 벽을 표시, 미로탐색위한 map
m x p의 크기
mark[m+2][p+2]: 시도한 위치 표시

현재의 위치 x: maze[i][j]



(가능한 이동)

```
struct offsets {
        int vert;
        int horiz;
};
offsets move[8]; /* 여덟 방향 이동에 대한 배열 */
```

<이동 테이블>

Name	dir	move[dir].vert	move[dir].horiz
N	0	-1	0
NE	1	-1	1
E	2	0	1
SE	3	1	1
S	4	1	0
SW	5	1	-1
W	6	0	-1
NW	7	-1	-1

[i][j]에서 sw로 이동한 후 [g][h]위치가 되었다면,

g= i+ move[sw].vert; h= j+ move[sw].horiz;

⇨다음 이동 위치 : *maze[next_row][next_col]*

next_row = row + move[dir].vert;

next_col = col + move[dir].horiz;

경로의 탐색

- *mark[m+2][n+2]* : 초기치 0, 바깥쪽은 모두 벽(1)
- 현위치 저장후 방향 선택 : 북쪽방향(N)에서부터 시계방향
- 잘못된 경로의 선택시는 backtracking후 다른 방향 시도
- 경로의 재시도 방지: *mark[row][col]* = 1 /* 한번 방문한 경우 */

stack[row][col]: 방문된 좌표 저장, 종료시 출력

```
struct element{    int x, y, dir;      }
element stack[maxSize];
```

[미로 탐색 알고리즘]

```
Procedure MIRO-PATH() {
    /* 미로를 통과하는 경로가 있으면 그 경로를 출력한다. */
    // initialize variables, found= FALSE;    element position;
     // 시작 위치인 진입좌표 (1,1), 방향은 북쪽으로 초기화
    mark[1][1]=1;    top=0;
    stack[0].row=1;    stack[0].col=1;    stack[0].dir=1; //북쪽

    while (stack is Not Empty && !found) {
        position = delete(&top); // stack의 top element
        row = position.row;        col = position.col;    dir = position.dir;

      while (dir < 8 && !found) { // 이동이 계속 가능
            //다음 이동좌표 계산
            next_row = row + move[dir].vert;      /* dir 방향으로 이동 */
            next_col = col + move[dir].horiz;

            // 출구 발견시, 종료 found=TRUE
            if (next_row==EXIT_ROW   &&   next_col==EXIT_COL)
                found = TRUE;          //EXIT_ROW, EXIT_COL값은 미리정의됨

            // 이동 가능하고, 이전에 방문하지 않은곳이면
            else if (!maze[next_row][next_col]   &&   !mark[next_row][next_col]) {
                    mark[next_row][next_col]) = 1;       // 현재위치 저장.
                    position.row = row;   position.col = col;   position.dir = ++dir;

                    Push (&top, position);    //현재위치 스택에 저장.

                    row = next.row;   col = next.col;   dir = 0;
            }
            else ++dir;
    }
}
if (found) {
   - Print miro-path 출력.
   - Print Marked matrix // 방문된 matrix 출력
 else print ("The maze does not have a path\n");}
```