



3장. 스택(Stack)과 큐(Queue)

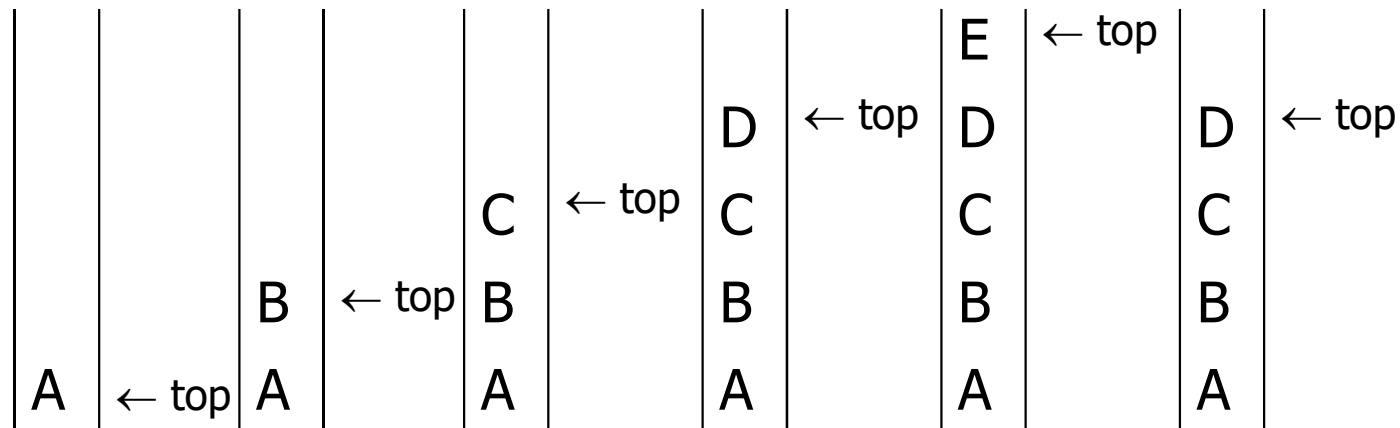


목차

- 스택의 개념
- 큐의 개념
- 미로 찾기 문제(**Mazing Problem**) *미로 찾기?*
-  수식의 계산(**Evaluation of Expression**)
- 다중 스택과 큐

1. 스택(Stack)의 개념

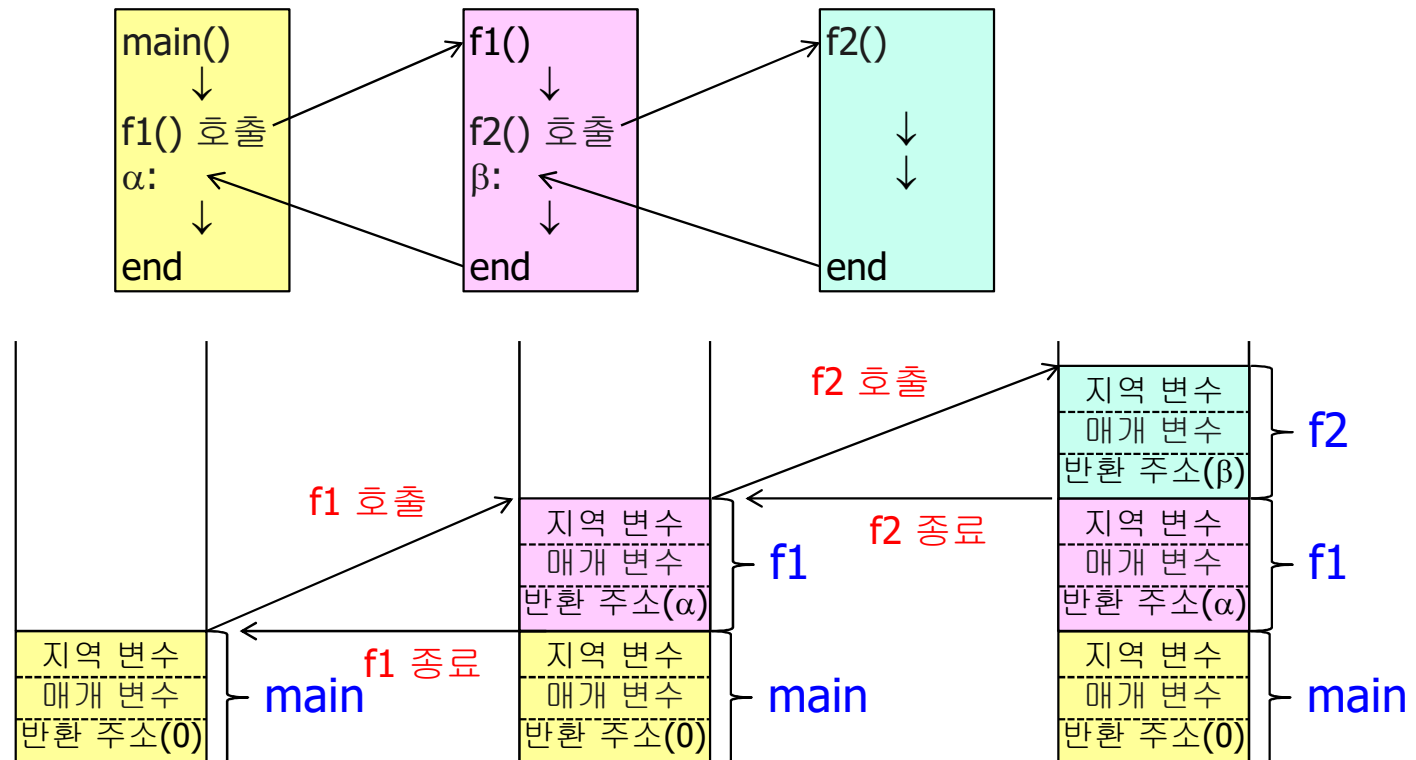
- 스택의 정의
 - 삽입과 삭제가 “**top**”이라 불리는 한쪽 끝 지점에서 발생하는 순서화 리스트
 - Last-In-First-Out (LIFO)



- **Example:** 시스템 스택
- **ADT 3.1:** 스택 ADT

Example: 시스템 스택(Run-time Stack)

- 시스템에서 함수 호출을 처리하기 위해 사용
- 재귀함수 호출의 성능과 관련



ADT 3.1: 스택 ADT

ADT Stack

객체: 0 혹은 그 이상의 유한 개 원소로 구성된 순서화 리스트
함수:

for all $stack \in \text{Stack}$, $item \in \text{element}$, $max_stack_size \in \text{양의 정수}$

Stack **CreateS(max_stack_size)** ::=
create an empty stack whose maximum size is max_stack_size

Boolean **IsFull(stack, max_stack_size)** ::=
if (number of elements in stack == max_stack_size)
return TRUE
else return FALSE

Stack **Push(stack, item)** ::=
if (IsFull(stack)) **stack_full** ← 오류처리
else insert item into **top** of stack and return
return void

Boolean **IsEmpty(stack)** ::=
if (stack == CreateS(max_stack_size)) return TRUE
else return FALSE

Element **Pop(stack)** ::=
if (IsEmpty(stack)) **stack_empty** ← 오류처리
else remove and return the item on the **top** of the stack
return 원소 반환

스택 ADT의 구현(1)

```
Stack CreateS(max_stack_size) ::=  
    #define MAX_STACK_SIZE 100  
    typedef struct {  
        int key;  
        /* other fields */  
    } element;  
    element stack[MAX_STACK_SIZE];  
    int top = -1; // 전역 변수
```

top = -1, 0 에 따라 push, pop이
달라짐

```
Boolean IsEmpty(Stack) ::= top < 0; top == -1
```

```
Boolean IsFull(Stack) ::= top >= MAX_STACK_SIZE - 1;
```

top = 0 top > MAX_STACK_SIZE



스택 ADT의 구현(2)

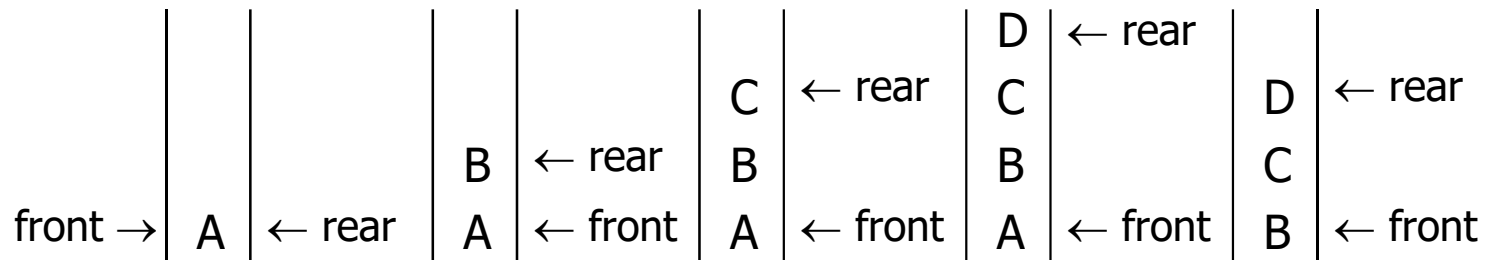
```
void push(element item)
{
    // 스택에 새로운 항목을 추가
    if (top >= MAX_STACK_SIZE - 1) {
        stack_full();
        return;
    }
    stack[top] = item;
    stack[++top] = item;
}
```

stack[top]에는 데이터 저장.
top = 0으로 초기화될 경우?

```
element pop()
{
    // 스택 top의 항목을 return
    if (top == -1)
        return stack_empty();
    return stack[top--];
}
```

2. 큐(Queue)의 개념

- 큐의 정의
 - 삽입과 삭제가 다른 쪽에서 발생하는 순서화 리스트
 - 삽입이 발생하는 위치: **rear**
 - 삭제가 발생하는 위치: **front**
 - First-In-First-Out (FIFO)



■ ADT 3.2: 큐 ADT

ADT 3.2: 큐 ADT

ADT Queue

객체: 0 혹은 그 이상의 유한 개 원소로 구성된 순서화 리스트

함수:

for all queue \in Queue, item \in element, max_queue_size \in 양의 정수

Queue **CreateQ(max_queue_size)** ::=

create an empty queue whose maximum size is max_queue_size

Boolean **IsFullQ(queue, max_queue_size)** ::=

if (number of elements in queue == max_queue_size)

return TRUE

else return FALSE

Queue **AddQ(queue, item)** ::=

if (IsFullQ(queue, max_queue_size)) return queue_full

else insert item at rear of queue and return queue

Boolean **IsEmptyQ(queue)** ::=

if (queue == CreateQ(max_queue_size)) return TRUE

else return FALSE

Element **DeleteQ(queue)** ::=

if (IsEmptyQ(queue)) return queue_empty

else remove and return the item at front of queue

큐 ADT의 구현(1)

Queue **CreateQ(max_queue_size)** ::=

```
#define MAX_Q_SIZE 100
```

```
typedef struct {
```

```
    int key;
```

```
    /* other fields */
```

```
} element;
```

```
element queue[MAX_Q_SIZE];
```

```
int rear = -1, front = -1; // 전역 변수
```

rear = 0
front = 0

Boolean **IsEmptyQ(queue)** ::= front == rear;

Boolean **IsFullQ(queue)** ::= rear == MAX_Q_SIZE - 1;

큐 ADT의 구현(2)

```
void addq(element item)
{ // Queue에 새로운 항목을 추가
  if (rear >= MAX_Q_SIZE - 1) {
    queue_full();
    return;
  }
  queue[++rear] = item;
}
```

Handwritten notes:

- next to `rear`: `rear = 0`
- next to `MAX_Q_SIZE - 1`: `≠`
- next to `queue[++rear]`: `rear++`

```
element deleteq()
{ // Queue의 항목을 return
  if (front == rear) return queue_empty();
  return queue[++front];
}
```

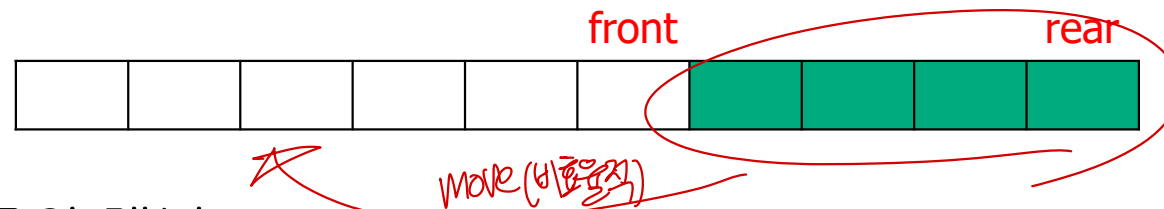
Example: 작업 스케줄링(Job Scheduling)

- 작업 큐(job queue)는 운영체제에 의해 관리
- 작업들간에 우선순위의 차이는 없다고 가정.
 - 우선순위 큐는 "5장 Tree"에서 설명.

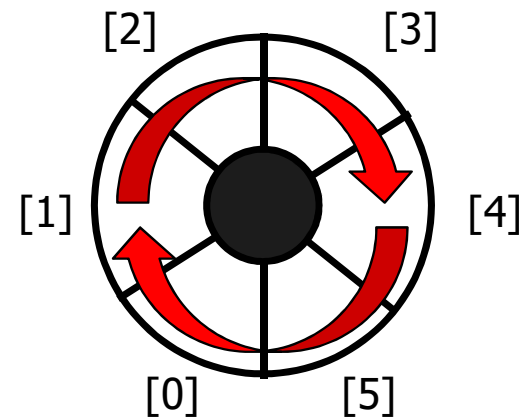
front	rear	Q[0]	Q[1]	Q[2]	Q[3]	Comments
-1	-1					Queue is empty
-1	0	J1				Job 1 is added
-1	1	J1	J2			Job 2 is added
-1	2	J1	J2	J3		Job 3 is added
0	2		J2	J3		Job 1 is deleted
1	2			J3		Job 2 is deleted

원형 큐(Circular Queue)

- 배열을 이용하여 큐를 구현할 때 발생하는 문제점
 - QueueFull의 조건: $\text{rear} == \text{max_Q_SIZE} - 1$
 - 문제점: 큐에 저장된 원소의 수 $< \text{max_Q_SIZE}$
 - 큐의 모든 항목들을 왼쪽으로 이동
 - 최악의 성능: $O(\text{MAX_Q_SIZE})$



- 원형 큐의 개념
 - 큐의 처음과 마지막을 연결
 - 나머지(%) 연산자 이용

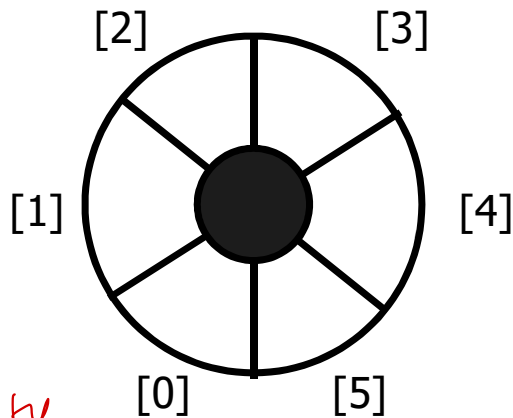


원형 큐(Circular Queue)

$\text{rear} + 1 == \text{front}$

or

$\text{rear} == (\text{front} - 1 + \text{max_queue_size}) \% \text{max_queue_size}$



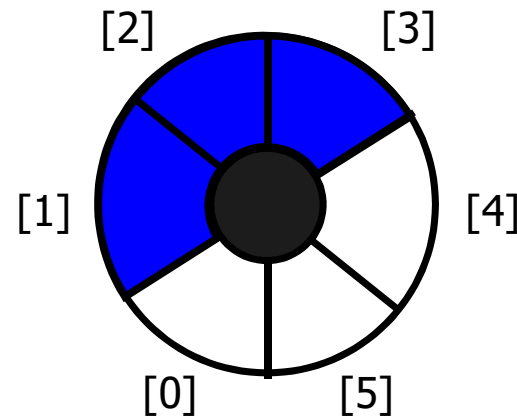
Q. Empty
 $f == r$

front = 0, rear = 0

~~rear++~~
 $(\text{rear} + 1) \% 6$

다음 칸

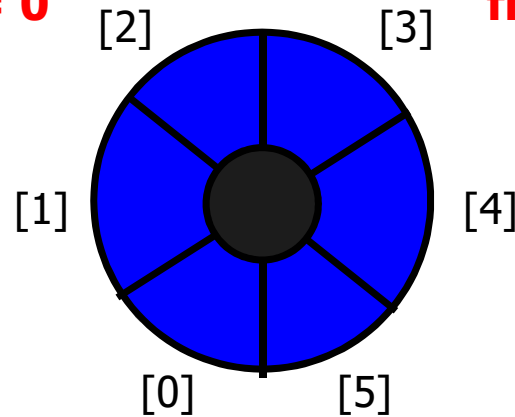
$(\text{rear} + 1) \% \text{max_queue_size} == \text{front}$



front = 0, rear = 3

Q. Full
 $\text{rear} + 1 == \text{front}$

최대 큐 이용률 =
 $\text{MAX_Q_SIZE} - 1$



front = 0, rear = 0

원형 큐의 구현

```
void addq(element item)
{
    // 원형 큐에 새로운 항목을 추가
    rear = (rear + 1) % MAX_Q_SIZE;
    if (rear == front) {
        queue_full(); return;
    }
    queue[rear] = item;
}
```

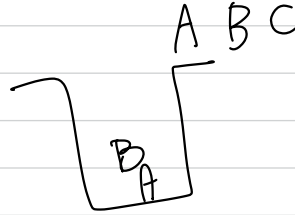
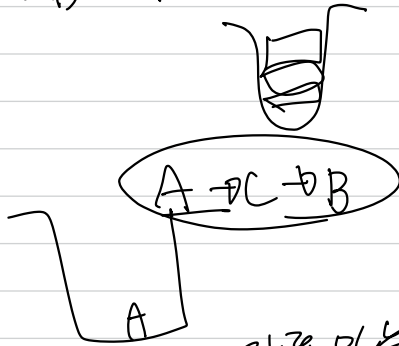
array size

```
element deleteq()
{
    // 원형 큐의 항목을 return
    if (front == rear)
        return queue_empty();
    front = (front + 1) % MAX_Q_SIZE;
    return queue[front];
}
```

C → B → A

(2, 2, 2)

C
B
A



2nd
front

2nd element

2nd
rear - 1

max index

if (rear == max + 1)

full

Queue[rear] = data

rear++

C R A
A B C //

~~X~~ B A C
C A B //



3. 미로 찾기(Mazing Problem)

- 2차원 배열을 이용한 미로의 구현
 - **maze[row][column]**: 0 - 길, 1 - 벽
 - 그림 3.8 참조
- 이동 방향
 - 8방향(N, NE, E, SE, S, SW, W, NW)
 - 각 방향에 대한 maze 배열의 첨자 변환: 그림 3.9
- 경계 지역: **8방향이 아님. (모서리: 3방향, 변: 5방향)**
 - $m \times p$ 미로를 $(m + 2) \times (p + 2)$ 미로로 변환
 - 각 경계 영역의 maze 배열값은 1로 설정
 - 입구: maze[1][1], 출구: maze[m][p]

그림 3.8: 예제 미로

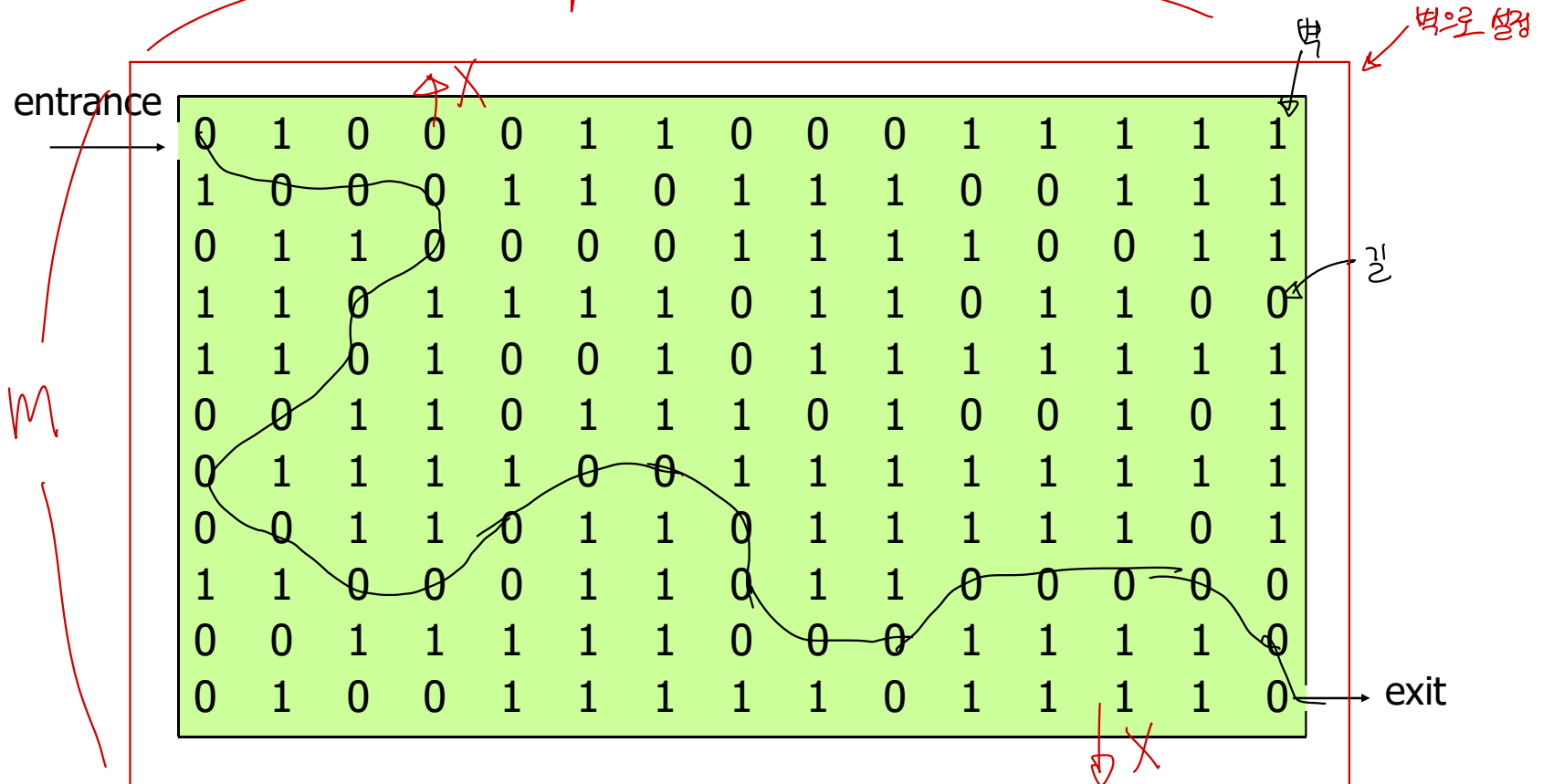
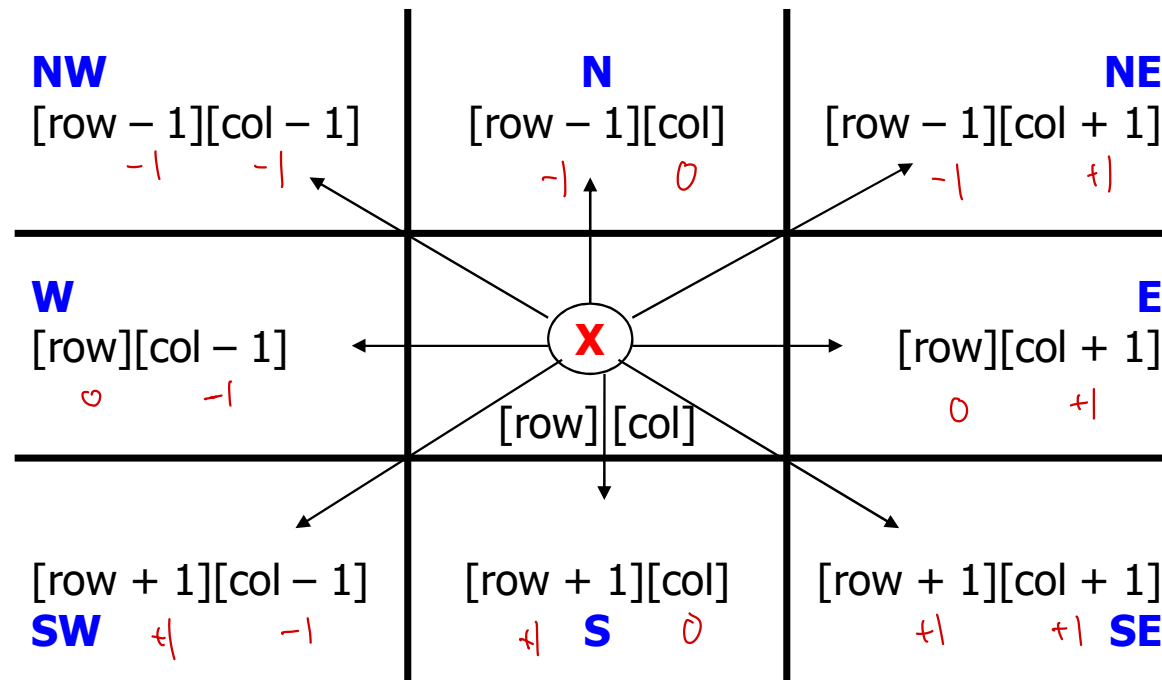


그림 3.9: 이동가능 지점



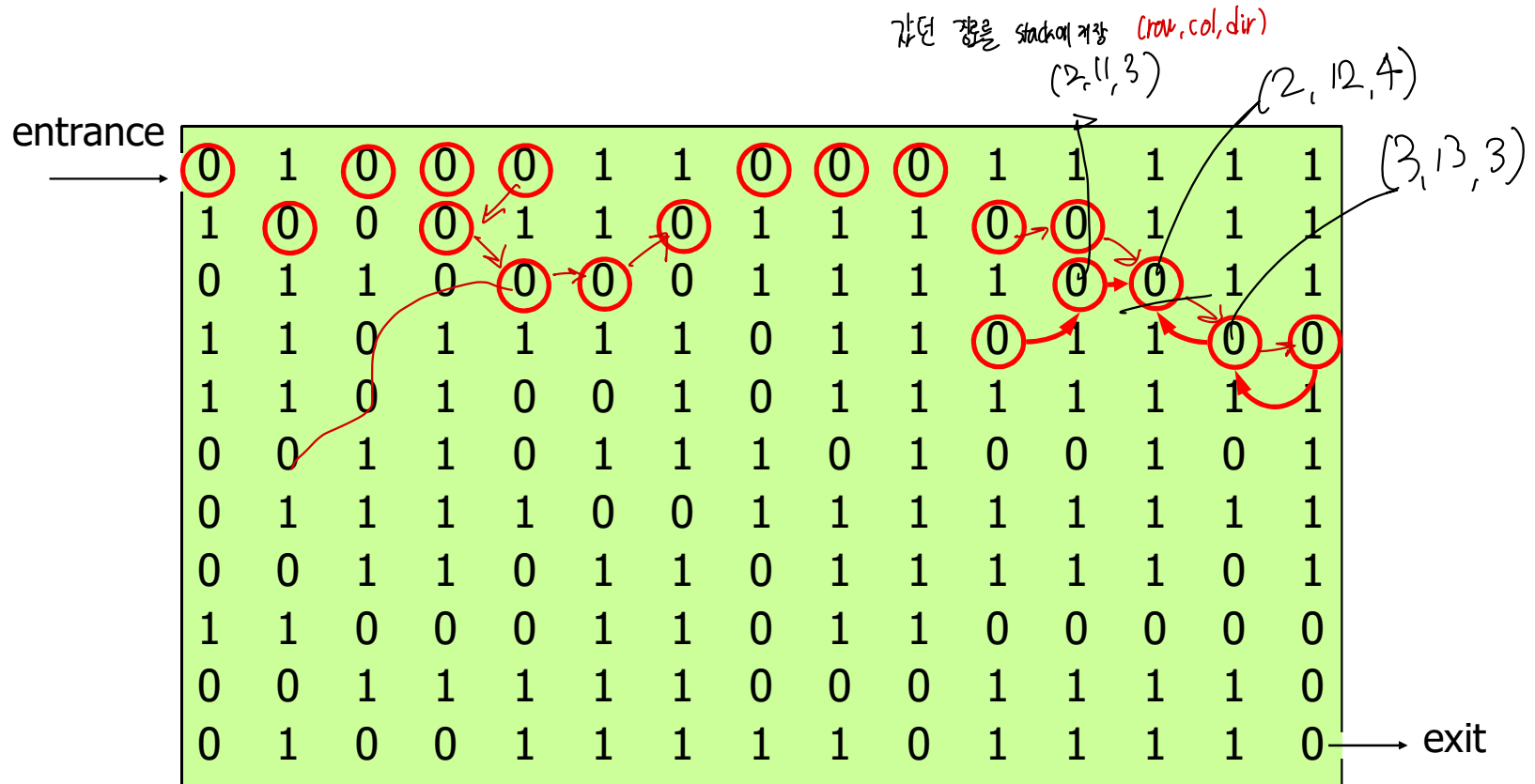
미로찾기 프로그램의 구현(1)

- 8가지 이동방향을 구현하기 위해 **move[8]** 배열 사용

```
typedef struct {  
    short int    x;  
    short int    y;  
} offsets;  
offsets move[8];
```

Name	Dir	move[dir].x	move[dir].y
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

그림 3.8: 예제 미로



미로찾기 프로그램의 구현(2)

- 현재 좌표가 (row, col)일 경우, 다음 이동좌표의 계산
 - $\text{next_row} = \text{row} + \text{move}[\text{dir}].x$
 - $\text{next_col} = \text{col} + \text{move}[\text{dir}].y$
- 북쪽(N)부터 다음 이동좌표를 차례대로 계산하여 이동 가능한 지(즉, 길 or 벽) 확인한 후, 길이면 이동.
- 한번 갔었던 길을 다시 갈 필요는 없으므로, 기존에 다녀온 길들을 **mark[m+2][p+2]** 배열에 저장
 - 모든 $\text{mark}[\text{row}][\text{col}]$ 의 값은 0으로 초기화
 - $\text{maze}[\text{i}][\text{j}]$ 를 방문한 후, $\text{mark}[\text{i}][\text{j}]$ 를 1로 설정
- 갔다가 길이 없을 경우 돌아와야지? **stack[]** 사용


maze : 미로
mark : 가본길 1
안가본길 0

$\text{maze}(\text{nr})(\text{nc}) == 0$ &&
 $\text{mark}(\text{nr})(\text{nc}) == 0$

```
#define MAX_STACK_SIZE 100 // = m x p
typedef struct {
    short int row;
    short int col;
    short int dir;
} element;
element stack[MAX_STACK_SIZE];
```

가장 최근 정보 가려내기

Program 3.11: 미로찾기 초기버전(1)

 (1,1) (1,1) E
미로의 입구좌표와 ~~N~~ 방향으로 stack 초기화 //최적화?
while (stack is not empty) {
 // stack top의 위치로 이동
 <row, col, dir> = delete from top of stack;
 while (there are more moves from current position) {
 <next_row, next_col> = coordinate of next move;
 dir = direction of move;
 if ((next_row == EXIT_ROW) &&
 (next_col == EXIT_COL))
 success;



Program 3.11: 미로찾기 초기버전(2)

```
if (maze[next_row][next_col] == 0 &&
    mark[next_row][next_col] == 0) {
    // 정상적인 길이며 아직 방문한 적이 없음
    mark[next_row][next_col] = 1; // 이제 방문
                                // 현재 좌표와 방향을 stack에 저장
    add <row, col, dir> to the top of the stack;
    row = next_row;
    col = next_col;
    dir = north;
}
}
}
printf("No path found \n");
```


Program 3.12: 미로찾기 최종버전(1)

```
void path(void)
{ // 미로를 통과하는 경로가 존재할 경우, 이를 출력
  int i, row, col, next_row, next_col, dir;
  int found = FALSE;
  element position;

  // 미로의 입구좌표와 E 방향으로 stack 초기화
  mark[1][1] = 1; top = 0;
  stack[0].row = 1; stack[0].col = 1; stack[0].dir = 2;
  while ( top > -1 && !found ) { // stack이 empty가 아니고, 아직
                                // 경로를 발견 못할 때까지 실행
    position = pop();           // top의 위치로 이동
    row = position.row;        col = position.col;
    dir = position.dir;
```

Program 3.12: 미로찾기 최종버전(2)

```
while ( dir < 8 && !found ) { // 8방향을 차례대로 검사
    next_row = row + move[dir].x; // move in direction dir
    next_col = col + move[dir].y;
    if ( next_row == EXIT_ROW && next_col == EXIT_COL )
        found = TRUE; // 출구 발견. 경로는 어디에?
    else if ( !maze[next_row][next_col] &&
        !mark[next_row][next_col] ) { // 아직 안 가본 길
        mark[next_row][next_col] = 1;
        position.row = row;
        position.col = col;
        position.dir = ++dir;
        push(position); // 현재 좌표와 방향을 stack 저장
        row = next_row; // 안 가본 길로 전진. 방향은 북쪽
        col = next_col;
        dir = 0;
    }
    else ++dir;
} }
```



Program 3.12: 미로찾기 최종버전(3)

```
if (found) {           // stack에 저장된 경로 출력
    printf( " The path is: \n " );
    printf ( "row  col \n" );
    for ( i=0; i <= top; i++ )
        printf( " %2d %5d ", stack[i].row, stack[i].col );
    printf( " %2d %5d \n ", row, col );
    printf( " %2d %5d \n ", EXIT_ROW, EXIT_COL );
}
else printf( " The maze does not have a path \n " );
}
```

$$(a = (b = (c = 4))) \quad (3+4)+5$$

l → r r → l

4. 수식 계산(Evaluation of Expressions)

- 수식에서 **Precedence**와 **Associativity**
 - **Precedence**: 연산자들간의 우선 순위
 - **Associativity**: 동일한 우선순위를 갖는 연산자들간의 실행 순서

- **Example**

- $x = ((a / b) - c) + (d * e) - (a * c)$
- $x = ((a / (b - c + d)) * (e - a)) * c$

C 언어에서 Precedence/Associativity(1)

Token	Operator	Precedence	Associativity
() [] -> .	function call array element struct or union member	17	left – to – right
-- ++	increment, decrement ²	16	left – to – right
-- ++ ! ~ - + & * sizeof	decrement, increment ³ logical not one's complement unary minus or plus address or indirection size (in bytes)	15	right – to – left
(type)	type cast	14	right – to – left
* / %	multiplicative	13	left – to – right
+ -	binary add or subtract	12	left – to – right

C 언어에서 Precedence/Associativity(2)

<< >>	shift	11	left – to – right
> >= < <=	relational	10	left – to – right
== !=	equality	9	left – to – right
&	bitwise and	8	left – to – right
^	bitwise exclusive or	7	left – to – right
	bitwise or	6	left – to – right
&&	logical and	5	left – to – right
	logical or	4	left – to – right
?:	conditional	3	right – to – left
= += -= /= *= % <<= >>= &= ^=	assignment	2	right – to – left
,	comma	1	left – to – right

중위 표기법과 후위 표기법

- 중위 표기법(**Infix notation**) 바꾸는 문제
 - 피연산자들 사이에 연산자가 위치
 - 괄호를 포함할 수 있고, 계산 과정이 복잡
- 후위 표기법(**Postfix notation**) (2+(3*4))
2 3 4 * +
 - 피연산자들 다음에 연산자가 위치
 - 괄호가 필요없고, 한번의 스캔으로 수식 계산 가능

Infix	Postfix
2 + 3	2 3 +
(a * b) + 5 (a b *) + 5	a b * 5 +
a + (b * 5) a b 5 * +	a b 5 * +
(1 + 2) * 7	1 2 + 7 *
(a / (b - c + d)) * (e - a) * c	a b c - d + / e a - * c *

$a b c - d + / e a - * c *$

Postfix 수식의 계산

- **Stack**을 이용: **6 2 / 3 - 4 2 * +**
 - 피연산자는 스택에 저장
 - 연산자의 경우, 스택에서 피연산자 pop & 결과를 push

Token	Stack[0]	Stack[1]	Stack[2]	Top
6	6			0
2	6	2		1
/	3			0
3	3	3		1
-	0			0
4	0	4		1
2	0	4	2	2
*	0	8		1
+	8			0

Postfix 수식 계산을 위한 자료구조

```
#define MAX_STACK_SIZE 100
#define MAX_EXPR_SIZE 100
typedef enum { lparen, rparen, +plus, -minus, *times,
              /divide, %mod, eos, operand } precedence;
int stack [ MAX_STACK_SIZE ]; // global stack
char expr [ MAX_EXPR_SIZE ]; // input string
```

[↑]
postfix '3' '4' '5' '*' '+'

Program 3.13: Postfix 수식 계산(1)

```
int eval (void)
{
    // expr[] 배열에 문자열로 저장된 postfix 수식 계산.
    // expr[]과 stack[], 그리고 top은 전역변수임.
    // get_token() 함수는 수식의 각 문자의 precedence를 return
    // 수식에서 피연산자는 한 문자로 구성된다고 가정.

    precedence token;
    char symbol;
    int op1, op2;
    int n = 0;          // 수식 문자열의 현재 판독 위치 expr의 현재위치
    top = -1;           // stack 초기화
    token = get_token(&symbol, &n);
    while (token != eos) {
        if (token == operand) {
            push(symbol - '0'); // 피연산자를 만나면 스택에 저장
        }
    }
}
```

Handwritten notes:

- pointer* (with arrow pointing to `&symbol`)
- 피연산자* (with arrow pointing to `operand`)
- 정수로 변환* (with arrow pointing to `symbol - '0'`)

Program 3.13: Postfix 수식 계산(2)

```
연산자 else { // stack에서 피연산자 2개를 제거한 후 이를 이용하여
              // 수식을 계산한 후 결과를 다시 stack에 저장
이항연산자( op2 = pop(); // stack delete
             op1 = pop();
             switch ( token ) {
                 case plus : push(op1 + op2); break;
                 case minus : push(op1 - op2); break;
                 case times : push(op1 * op2); break;
                 case divide : push(op1 / op2); break;
                 case mod : push(op1 % op2);
             }
             }
            token = get_token ( &symbol, &n );
        }
        return pop(); // return result
    }
```

Program 3.13: Postfix 수식 계산(3)

```
precedence get_token (char *symbol, int * n)
{ // 수식 문자열에서 다음 문자를 검사하여 해당 token을 반환

    *symbol = expr[(*n)++];
    switch (*symbol)
    {
        case '(': return lparen;
        case ')': return rparen;
        case '+': return plus;
        case '-': return minus;
        case '/': return divide;
        case '*': return times;
        case '%': return mod;
        case '—': return eos;
        default : return operand; // 오류 검사 없음. 피연산자
    }
}
```

enum token

eos 끝

Infix 수식을 Postfix 수식으로 변환

- 방법 1: 괄호를 이용하여 변환

- 예: $a / b - c + d * e - a * c //$

- $(((((a / b) - c) + (d * e)) - (a * c)))$

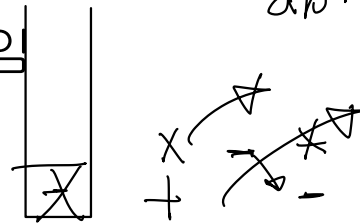
- $a b / c - d e * + a c * -$

- 두 단계 처리로 인해 비효율적임

먼저 계산해야 할 부분 괄호

$ab/c - de * + ac * -$

- 방법 2: **Stack**을 이용하여 변환



- 연산자들의 우선순위를 이용하여 변환

- 우선순위(top) < 우선순위(incoming): 입력 연산자를 스택에 저장

- 우선순위(top) > 우선순위(incoming): 스택 top에 있는 연산자를 출력

- 우선순위(top) = 우선순위(incoming): 결합성에 따라 처리

- 괄호가 있는 수식 처리에 주의

그림 3.15: 수식 변환의 예(1) 연산자가 stack에

Token	Stack [0] [1] [2]	Top	Output
a		-1	a
+	+	0	a
b	+	0	ab
*	+	1	ab
c	+	1	abc
eos		-1	abc*+

Token	Stack [0] [1] [2]	Top	Output
a		-1	a
*	*	0	a
b	*	0	ab
+	+	1	ab*
c	+	1	ab*c
eos		-1	ab*c+

$a + b \text{ (X) } c$
 incoming
 $abc * +$
 1) $in > st$
 2) $in < st$
 3) $in == st$

$a \times b \text{ (X) } c$
 $ab * c +$
 stack

$a + b + c$
 $ab + c +$
 stack
 ① left \rightarrow right
 ② right \rightarrow left

$a + b \times c$
 $abc * +$
 incoming
 $a \times b + c$
 abc
 $ab + c$

그림 3.16: 수식 변환의 예(2)

$$a * (b + c) * d$$

$a * b * c + * d *$
 $a * b * c + * d *$

Token	Stack [0] [1] [2]	Top	Output
a		-1	a
*	*	0	a
((1	a
b	(1	ab
+	(+	2	ab
c	(+	2	abc
)	*	0	abc +
*	*	0	abc + *
d	*	0	abc + * d
eos	*	0	abc + * d *

입력 연산자 (: 우선 순위가 가장 높다.

스택 **top**에서의 (: 우선 순위가 가장 낮다.

입력 연산자) : (를 만날 때까지 스택에 있는 연산자들을 **pop**

수식 변환을 위한 자료구조

- 연산자를 위한 **stack** 필요
- 괄호 연산자를 처리하기 위한 두 종류의 우선순위
 - int isp[]: stack에 저장된 연산자의 우선순위 in-stack precedence
 - int icp[]: 입력 연산자의 우선순위 incoming precedence

```
precedence stack[ MAX_STACK_SIZE ];
```

```
/* isp and icp arrays - index is value of precedence
```

```
lparen, rparen, plus, minus, times, divide, mod, eos */
```

```
[ int    isp[ ] = { 0, 19, 12, 12, 13, 13, 13, 0 };  
  int    icp[ ] = { 20, 19, 12, 12, 13, 13, 13, 0 };
```



3+(2 X 4)
'('=20 stack + (X
3 24 X +
'('=0 stack (X
3+24 X

Program 3.15: 수식 변환(1)

```
void postfix ( void )  
{ // 수식 변환 프로그램. (infix → postfix)  
  // expr[(infix 저장)과 stack[], 그리고 top은 전역변수
```

```
  char symbol;  
  precedence token;  
  int n = 0;  
  top = -1;
```

3 + (4 * 5)
↓ ↓ ↓ ↓ ↓

```
  push(eos);           // place eos on stack  for문 시작할 때 한 번만 실행  
  for (token = get_token(&symbol, &n); token != eos;  
        token = get_token(&symbol, &n)) {  
    if (token == operand)    printf("%c", symbol);  
                             피연산자
```

Program 3.15: 수식 변환(2)

```
else if (token == rparen) {  
    // 왼쪽 괄호가 나올 때까지 stack pop  
    while (stack[top] != lparen)   
        print_token(pop());  
    pop(); // 왼쪽 괄호 제거  
}  
else { // 우선순위가 높은 연산자 pop. associativity?  
    while (isp[stack[top]] >= icp[token])  
        print_token(pop());  
    push(token);  
}  
}  
while ( (token = pop()) != eos)  
    print_token(token); // stack의 모든 연산자 출력  
printf("\n");
```

$3 + (4 \times 5)$
stack + (*)
3 4 5 * +

$3 \times 5 + 7$
stack X +
3 5 7 X +

eos
✓
 $3 + 4 \times 2$
stack + X
3 4 2 X +

연습 문제

1
4
1
3
5
2
X
2

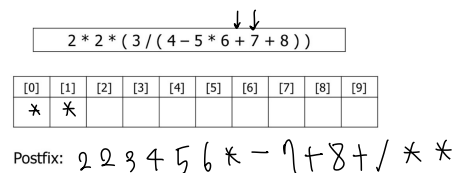
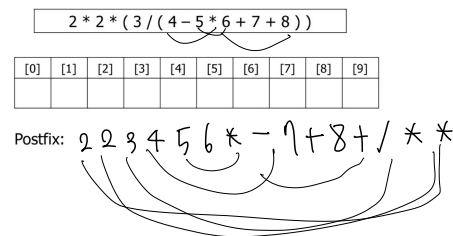
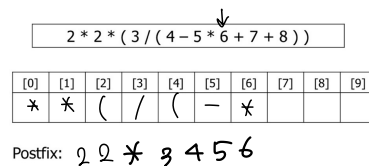
5x6

- 아래 수식을 **postfix** 수식으로 변환하고자 한다. 저장된 데이터가 가장 많을 때의 스택 내용을 표시하라.

$$2 * 2 * (3 / (4 - 5 * 6 + 7 + 8))$$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
*	(/	(-	X				

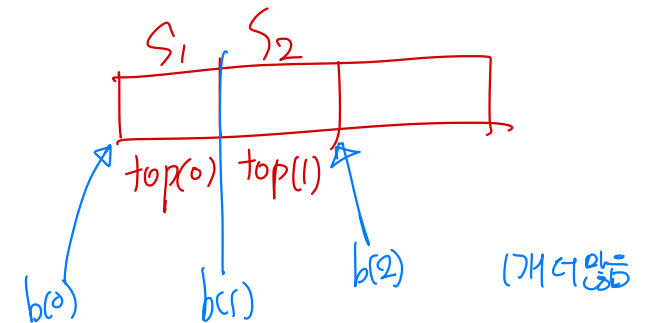
Postfix: 2 2 * 3 4 5 6



5. 다중 스택과 큐 (Multiple Stacks and Queue)

- 스택이나 큐를 배열로 구현할 경우 문제점
 - 정적 메모리 할당.
 - 스택/큐 오버플로우 혹은 메모리 낭비
- 다중 스택과 큐의 기본 개념
 - memory[MSIZE]에 n 개의 스택/큐를 저장하자
- C 언어에서 다중 스택의 구현

```
#define MSIZE 100 // 메모리 크기
#define MAX_STACKS 10 // 최대 스택 수
// 전역 메모리 선언
element memory[MSIZE];
int top[MAX_STACKS];
int boundary[MAX_STACKS+1];
int n; // 사용자가 입력한 스택의 수
```



C 언어에서 다중스택의 구현(1)

- **memory[]** 배열을 **n**개의 스택으로 균등하게 분할

$\text{top}[0] = \text{boundary}[0] = -1;$

$\text{for} (i = 1; i < n; i++)$

$\text{top}[i] = \text{boundary}[i] = (\text{MSIZE} / n) * i - 1;$

$\text{boundary}[n] = \text{MSIZE} - 1;$

$\begin{matrix} =100 & \div 5 \\ 100/5=20 \times 1 - 1 = 19 \end{matrix}$

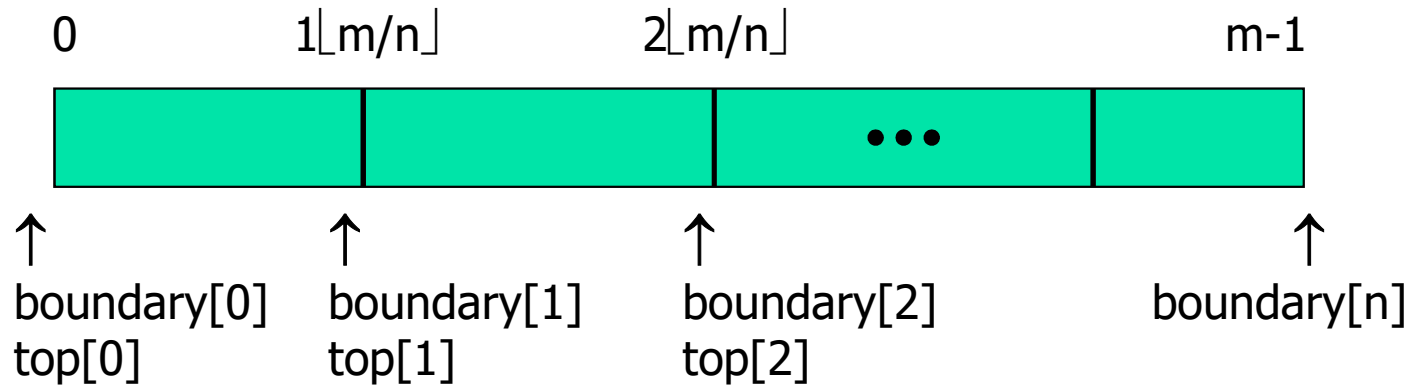
- **Stack_Full**과 **Stack_Empty** 조건

- **Stack_Full** $\text{top}(i) = b(i+1)$
 - $\text{top}[\text{stack_no}] == \text{boundary}[\text{stack_no} + 1]$

- **Stack_Empty** $\text{top}(i) = b(i)$
 - $\text{top}[\text{stack_no}] == \text{boundary}[\text{stack_no}]$

$\text{top}(i) = b(i) + 1 \cdots b(i+1)$

다중 스택의 초기 구성



모든 스택은 초기에 **empty**이며, **memory[]** 배열을 균등하게 배분.



C 언어에서 다중스택의 구현(2)

```
void push( int i, element item )
{ // i-번째 스택에 새로운 item을 추가
  if (top[i] == boundary[i+1])
    return ( stack_full ( i ) );
  memory[++top[i]] = item;
}
```

```
element pop( int i )
{ // i-번째 스택의 top에 저장된 항목을 삭제
  if ( top[i] == boundary[i] )
    return ( stack_empty ( i ) );
  return ( memory[top[i]--] );
}
```

Stack_Full의 구현

■ 배경

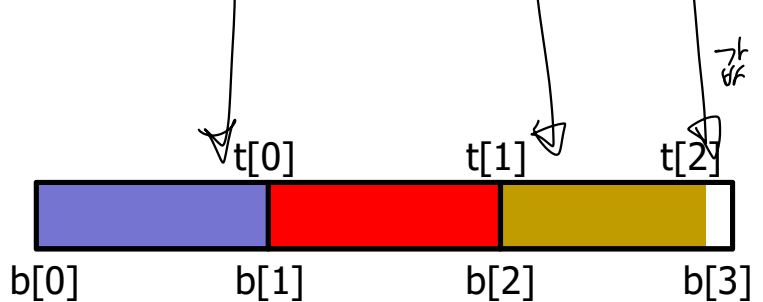
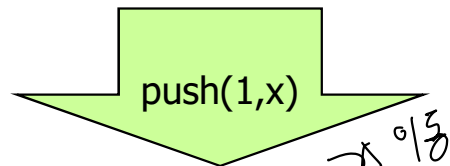
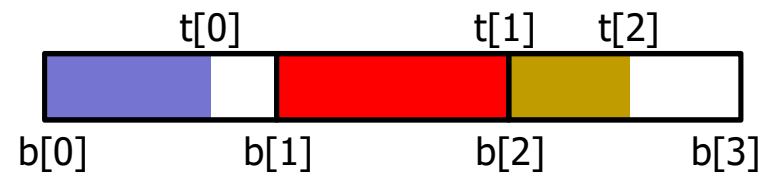
- i-번째 스택이 $\text{full}(\text{top}[i] == \text{boundary}[i+1])$ 이라도 $\text{memory}[]$ 배열에는 여유 공간이 있을 수 있다.
- $\text{boundary}[]$ 와 $\text{top}[]$ 을 변경하여 i-번째 스택에 추가적인 공간을 제공하자...

■ 구현 방법

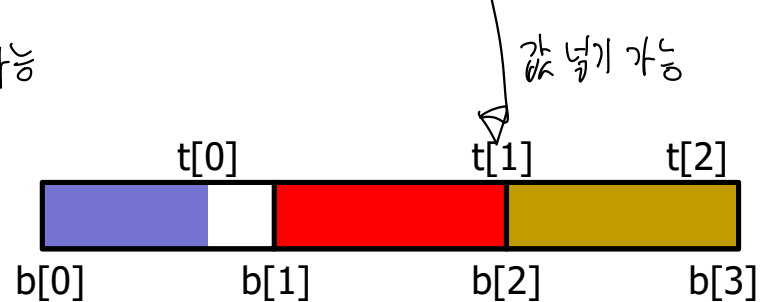
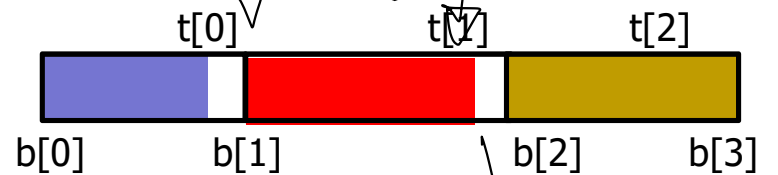
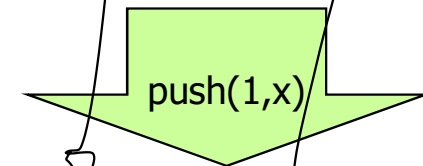
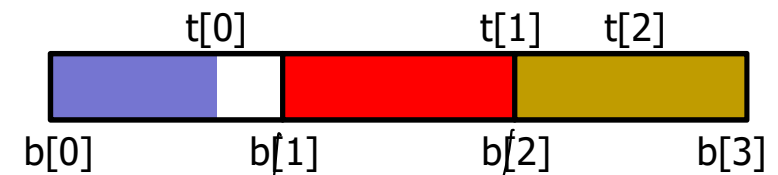
- 가정: $\text{top}[\text{stack_no}] == \text{boundary}[\text{stack_no}+1]$
- $\exists j (\text{stack_no} < j < n \ \&\& \ \text{top}[j] < \text{boundary}[j+1])$
 - 오른쪽 스택에 여유 공간 있음. 한칸씩 Shift Right.
- $\exists k (0 < k < \text{stack_no} \ \&\& \ \text{top}[k] < \text{boundary}[k+1])$
 - 왼쪽 스택에 여유 공간 있음. 한칸씩 Shift Left.
- Otherwise, Memory_FULL
- 실제로 구현해 볼 것!

오른쪽, 왼쪽 스택 검사

Stack_Full의 동작 예



$\text{push}(0, y) ?$



$\text{push}(2, y) ?$

$$(a \times b) + (c \div e)$$

$$ab \times cd$$

$$A \times B \div C/D$$

$$A \div (B \div C) / D - E$$

$$A \times B / (C - D) + E$$

$$A / B - C + D \times E$$

$$A B \times (D) / + \quad \checkmark$$

$$A B C + D / + E - \quad \checkmark$$

$$A B \times (D) / E \quad \times$$

$$A B / C - D E \times +$$

$$354 \times + 62 / 35 + - +$$

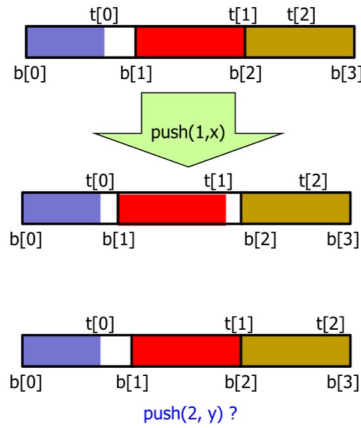
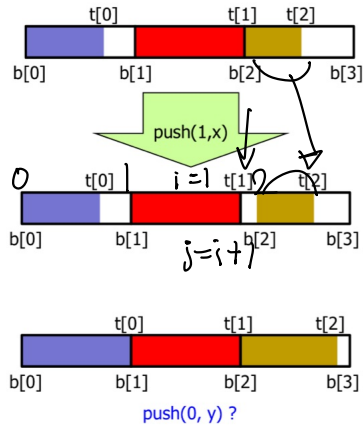
3	5	4		
---	---	---	--	--

$$\begin{array}{r}
 23 \quad 62 \\
 \times \quad 35 \\
 \hline
 115 \\
 690 \\
 \hline
 805
 \end{array}$$

$$354 \times + 62 / 35 + - +$$

$$28$$

$$1$$



$n = 3$

```
for(int j=i+1; j<MSIZE; j++)
{
    if( boundary[j] != top[j] )
    {
        // ...
    }
}
```