



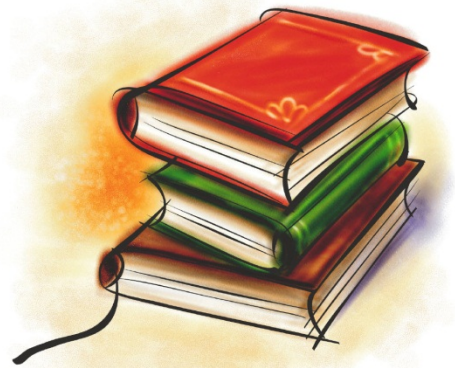
CSE2010 자료구조론

Week 5: Stack

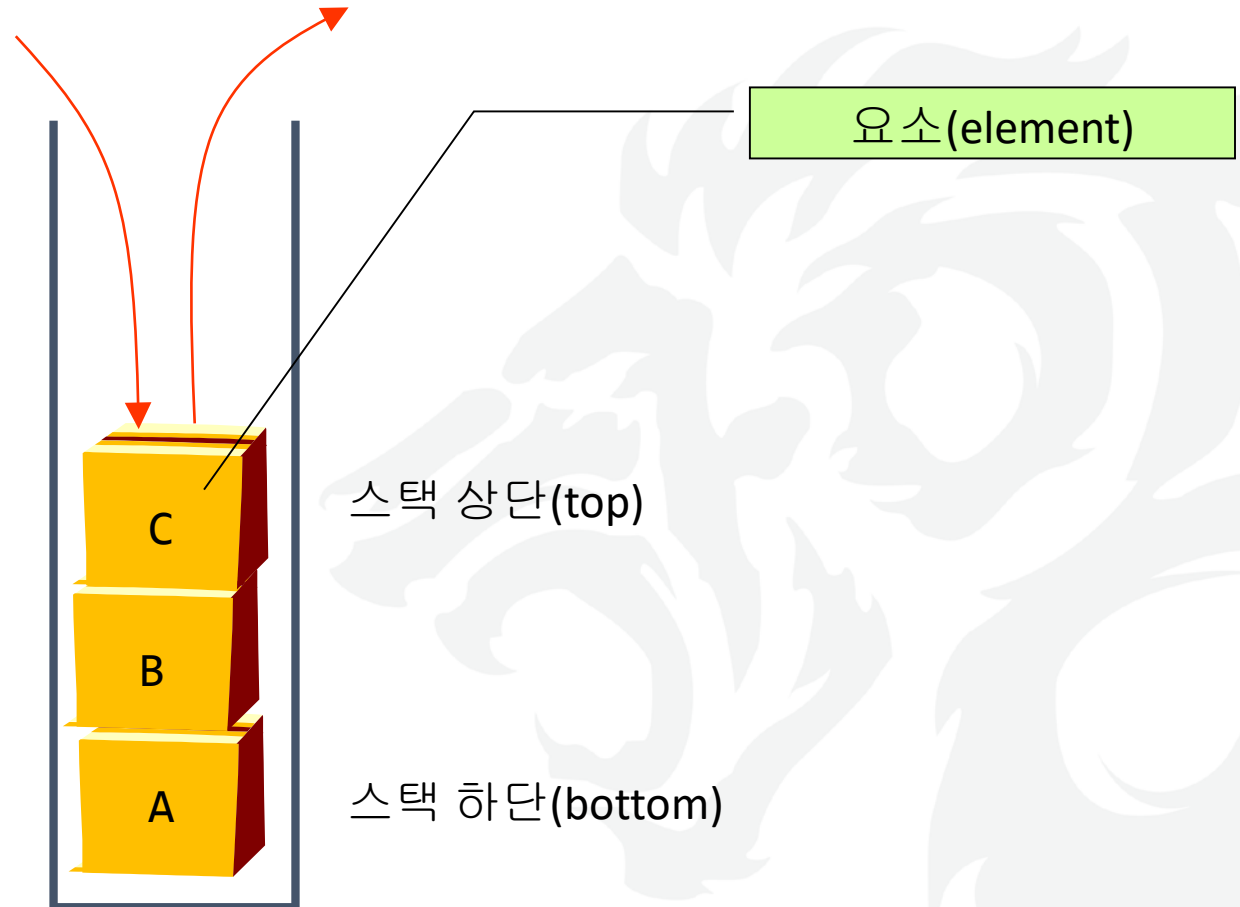
ICT융합학부 조용우

스택(stack)?

- 스택 = 쌓아 놓은 더미



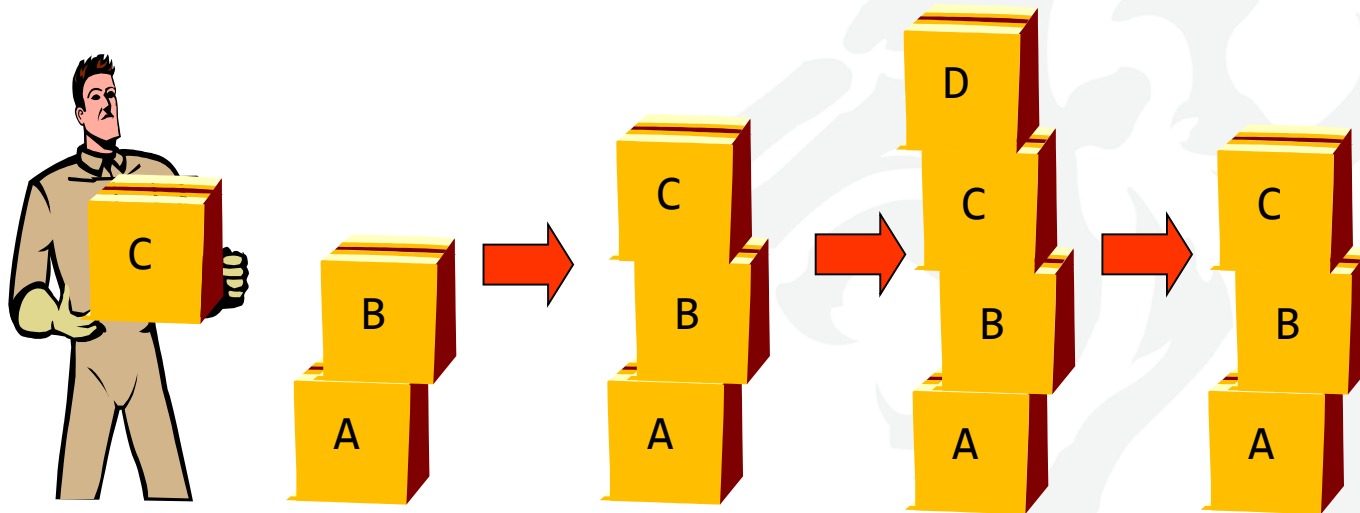
스택의 구조



스택의 특징

- 후입선출(LIFO: Last-In First-Out)

- 가장 최근에 들어온 데이터가 가장 먼저 나감



스택 ADT

■ 스택 ADT

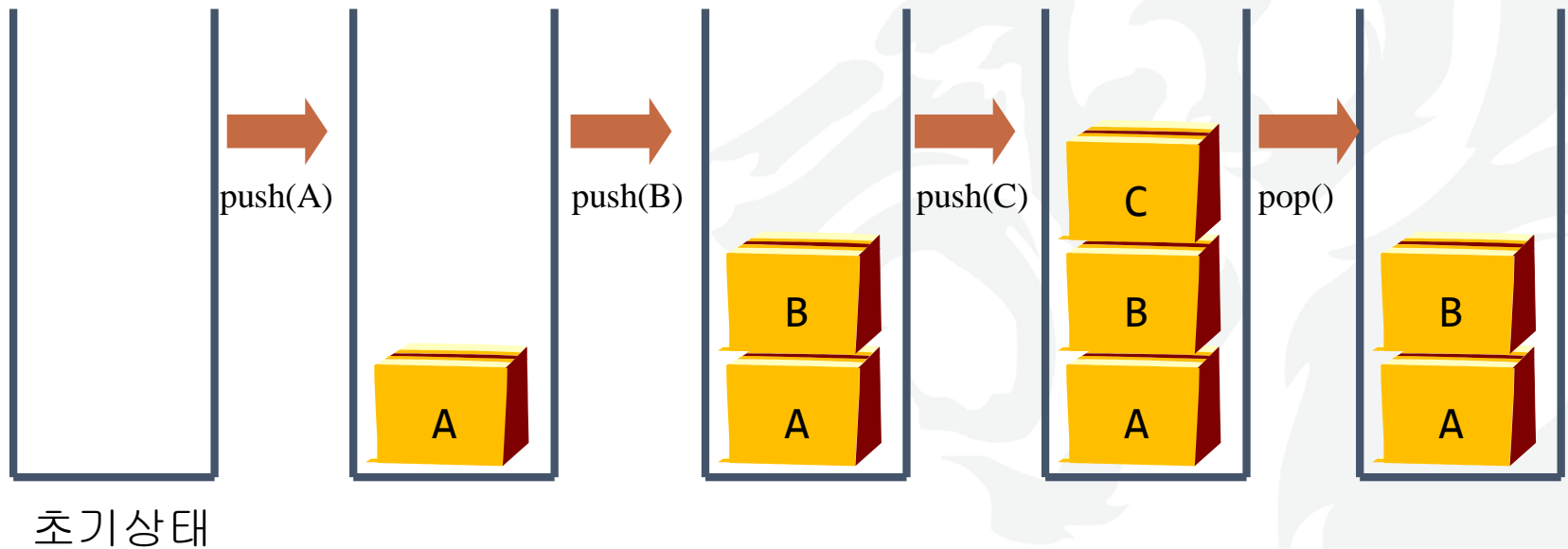
.객체: n 개의 `element`형의 요소들의 선형 리스트

.연산:

- `create()` ::= 스택을 생성
- `is_empty(s)` ::= 스택이 비어있는지를 검사
- `is_full(s)` ::= 스택이 가득 찼는가를 검사
- `push(s, e)` ::= 스택의 맨 위에 요소 `e`를 추가
- `pop(s)` ::= 스택의 맨 위에 있는 요소를 삭제
- `peek(s)` ::= 스택의 맨 위에 있는 요소를 삭제하지 않고 반환

스택 연산(1)

- push(): 스택에 데이터를 추가
- pop(): 스택에서 데이터를 삭제



스택 연산(2)

- `create()`: 스택을 생성
- `is_empty(s)`: 스택이 공백상태인지 검사
- `is_full(s)`: 스택이 포화상태인지 검사
- `pop(s)`: 요소를 스택에서 완전히 삭제하면서 가져옴
- `peek(s)`: 요소를 스택에서 삭제하지 않고 보기만 하는 연산

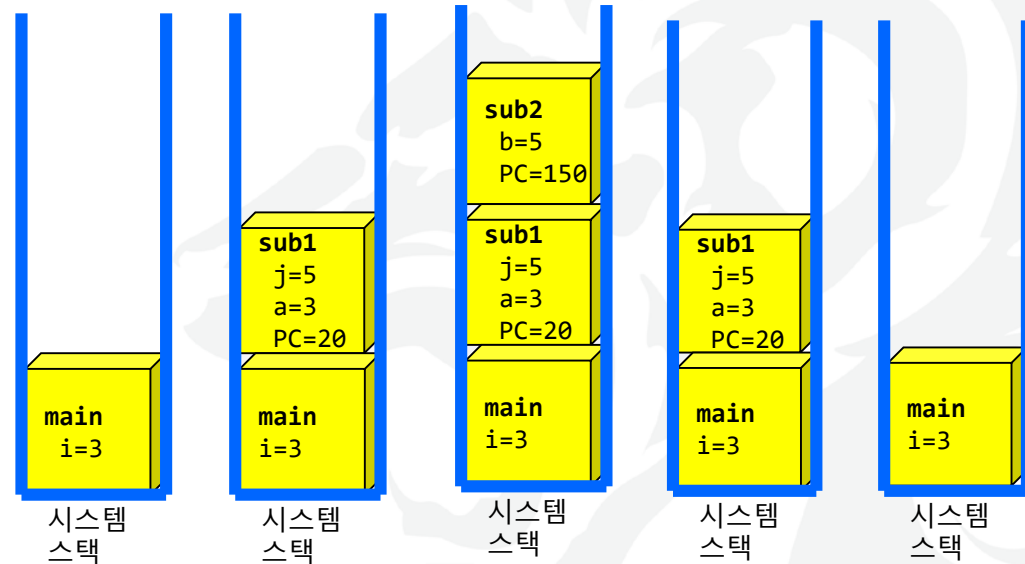
스택의 용도 예(1)

- 자료의 출력 순서가 입력 순서의 역순으로 이뤄져야 할 때
 - 입력: (A,B,C,D,E) -> 출력: (E,D,C,B,A)
- 에디터에서 되돌리기(undo) 기능
 - 최근 수행한 명령어들 중에서 가장 최근에 수행한 것부터 되돌리기
- 함수 호출에서 복귀주소(PC: Program Counter) 기억
 - 시스템 스택: 컴퓨터 OS만 사용함, 사용자는 접근 안됨
 - 시스템 스택에는 함수가 호출될 때마다 활성화 레코드(activation record)가 만들어지고, 여기에 복귀주소가 기록됨

스택의 용도 예(2)

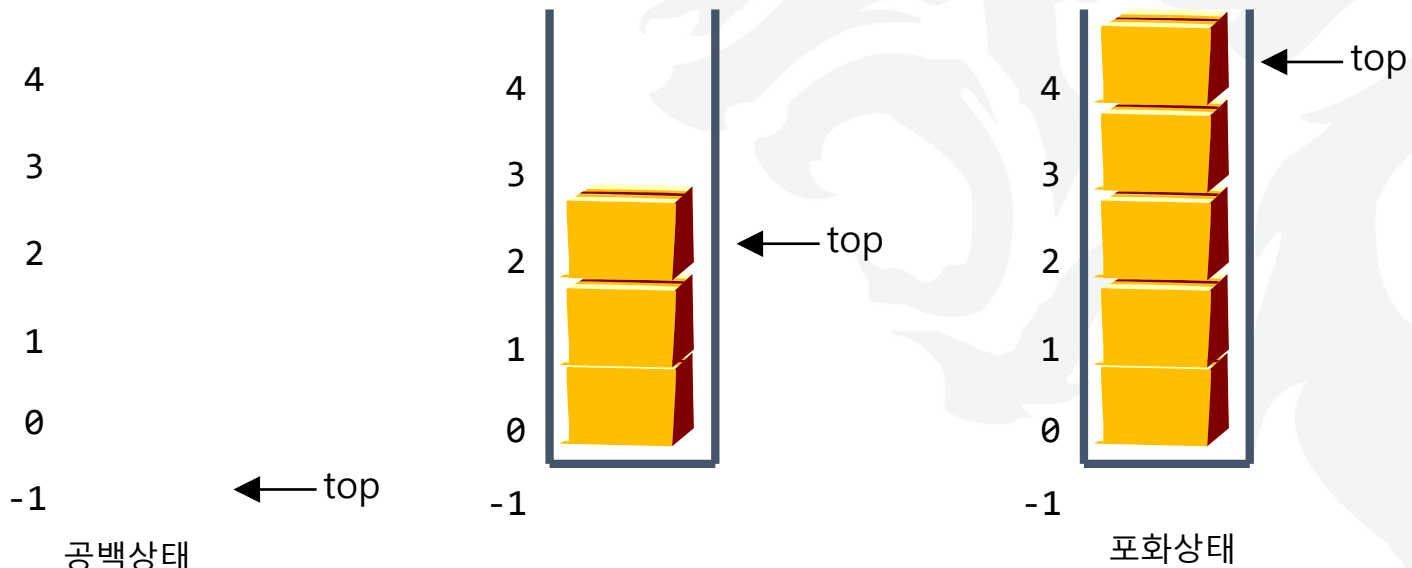
- 함수 호출시 복귀주소(PC: Program Counter)를 시스템 스택에 저장함

```
1  int main()  
  {  
    int i=3;  
20  sub1(i);  
    ...  
  }  
  
100 int sub1(int a)  
   {  
     int j=5;  
150  sub2(j);  
     ...  
   }  
  
200 void sub2(int b)  
   {  
     ...  
   }
```



배열을 이용한 스택의 구현

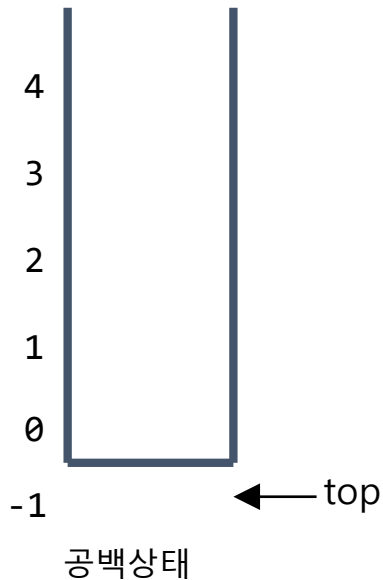
- 1차원 배열 `stack[MAX_STACK_SIZE]`
- `top` 변수: 스택에서 가장 최근에 입력되었던 자료를 가리킴
 - 가장 먼저 들어온 요소는 `stack[0]`에, 가장 최근에 들어온 요소는 `stack[top]`에 저장
 - 스택이 공백상태이면 `top`은 -1



배열을 이용한 스택의 구현: is_empty, is_full

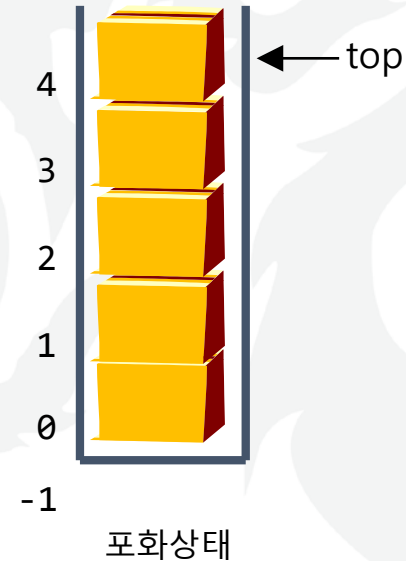
```
is_empty(S)
```

```
if top = -1  
    then return TRUE  
    else return FALSE
```



```
is_full(S)
```

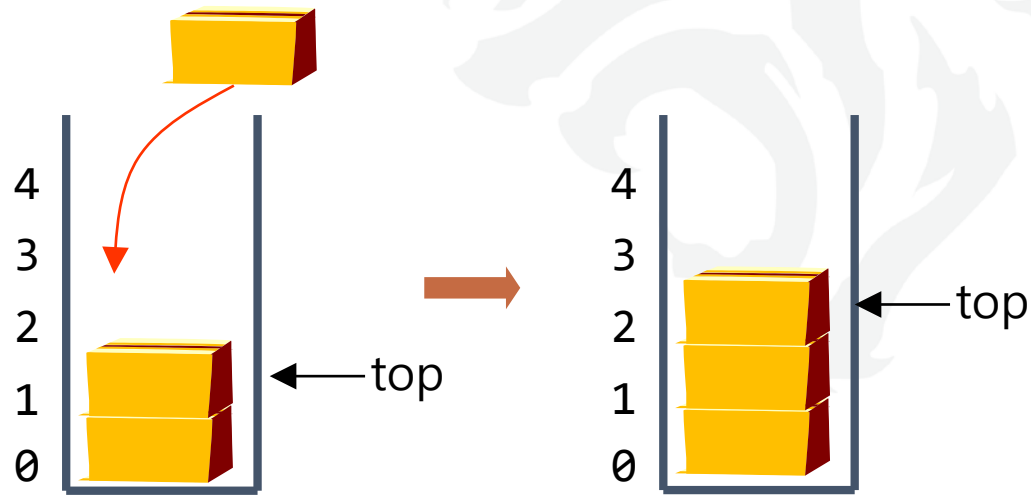
```
if top = (MAX_STACK_SIZE-1)  
    then return TRUE  
    else return FALSE
```



배열을 이용한 스택의 구현: push

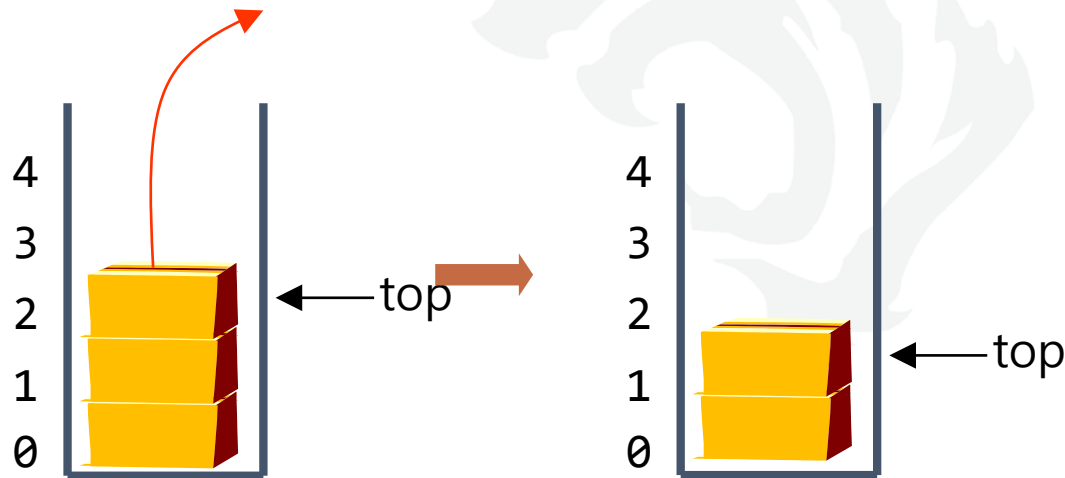
```
push(S, x)
```

```
if is_full(S)  
    then error "overflow"  
    else top  $\leftarrow$  top+1  
        stack[top]  $\leftarrow$  x
```



배열을 이용한 스택의 구현: pop

```
pop(S)  
  
if is_empty(S)  
    then error "underflow"  
    else  $e \leftarrow \text{stack}[\text{top}]$   
         $\text{top} \leftarrow \text{top} - 1$   
        return  $e$ 
```



배열을 이용한 스택 구현#1: 전역변수(1)

```
#define MAX_STACK_SIZE 100
typedef int element;
element stack[MAX_STACK_SIZE];
int top = -1;

// 공백 상태 검출 함수
int is_empty( )
{
    return (top == -1);
}

// 포화 상태 검출 함수
int is_full( )
{
    return (top == (MAX_STACK_SIZE-1));
}
```

배열을 이용한 스택 구현#1: 전역변수(2)

```
// 삽입함수
void push(element item)
{
    if( is_full() ) {
        fprintf(stderr, "스택 포화 에러\n");
        return;
    }
    else stack[++top] = item;
}

// 삭제함수
element pop( )
{
    if( is_empty( ) ) {
        fprintf(stderr, "스택 공백 에러\n");
        exit(1);
    }
    else return stack[top--];
}
```

배열을 이용한 스택 구현#1: 전역변수(3)

```
// 피크함수
element peek( )
{
    if( is_empty( ) ) {
        fprintf(stderr, "스택 공백 에러\n");
        exit(1);
    }
    else return stack[top];
}
```


배열을 이용한 스택 구현#1: 매개변수(1)

```
typedef int element;
typedef struct {
    element stack[MAX_STACK_SIZE];
    int top;
} StackType;
```

// 스택 초기화 함수

```
void init(StackType *s)
{
    s->top = -1;
}
```

// 공백 상태 검출 함수

```
int is_empty(StackType *s)
{
    return (s->top == -1);
}
```

배열을 이용한 스택 구현#1: 매개변수(2)

```
// 포화 상태 검출 함수
int is_full(StackType *s)
{
    return (s->top == (MAX_STACK_SIZE-1));
}

// 삽입 함수
void push(StackType *s, element item)
{
    if( is_full(s) ) {
        fprintf(stderr, "스택 포화 에러\n");
        return;
    }
    else s->stack[++(s->top)] = item;
}
```

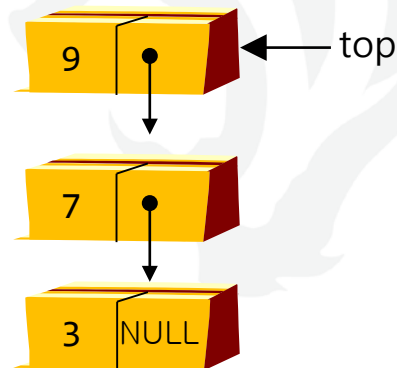
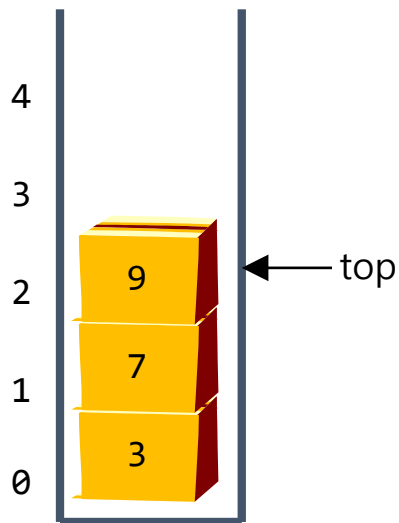
배열을 이용한 스택 구현#1: 매개변수(3)

```
// 삭제함수
element pop(StackType *s)
{
    if( is_empty(s) ) {
        fprintf(stderr, "스택 공백 에러\n");
        exit(1);
    }
    else return s->stack[(s->top)--];
}

// 피크함수
element peek(StackType *s)
{
    if( is_empty(s) ) {
        fprintf(stderr, "스택 공백 에러\n");
        exit(1);
    }
    else return s->stack[s->top];
}
```

연결된 스택

- 연결된 스택(linked stack): 연결리스트를 이용하여 구현한 스택
- 장점: 크기가 제한되지 않음
- 단점: 구현이 복잡하고 삽입이나 삭제 시간이 오래 걸림
 - 동적 메모리 할당 및 해제 때문



연결된 스택 구조

```
typedef int element;
```

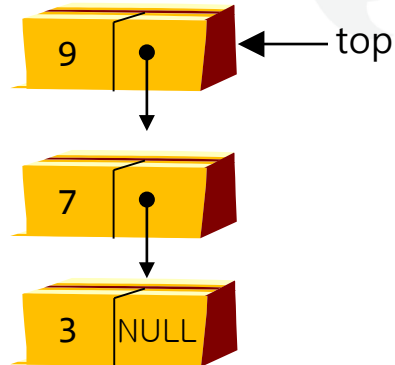
요소의 타입

```
typedef struct StackNode {  
    element item;  
    struct StackNode *link;  
} StackNode;
```

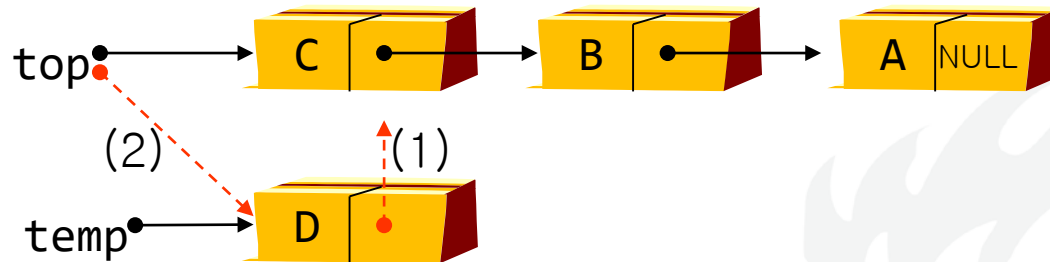
노드의 타입

```
typedef struct {  
    StackNode *top;  
} LinkedStackType;
```

연결된 스택의 관련 데이터

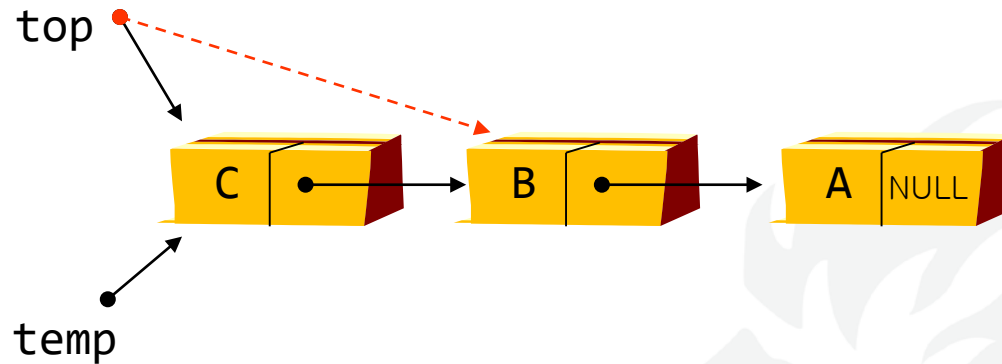


연결된 스택: push



```
// 삽입 함수
void push(LinkedStackType *s, element item)
{
    StackNode *temp=(StackNode *)malloc(sizeof(StackNode));
    if( temp == NULL ){
        fprintf(stderr, "메모리 할당에러\n");
        return;
    }
    else{
        temp->item = item;
        temp->link = s->top;
        s->top = temp;
    }
}
```

연결된 스택: pop



```
// 삭제 함수
element pop(LinkedStackType *s)
{
    if( is_empty(s) ) {
        fprintf(stderr, "스택이 비어있음\n");
        exit(1);
    }
    else{
        StackNode *temp=s->top;
        element item = temp->item;
        s->top = s->top->link;
        free(temp);
        return item;
    }
}
```

스택의 응용: 괄호 검사

- 괄호의 종류: 대괄호 ('[', ']'), 중괄호 ('{', '}'), 소괄호 ('(', ')')
- 조건
 - 왼쪽 괄호의 개수와 오른쪽 괄호의 개수가 같아야 함
 - 같은 괄호에서 왼쪽 괄호는 오른쪽 괄호보다 먼저 나와야 함
 - 서로 다른 타입의 왼쪽 괄호와 오른쪽 괄호 쌍은 서로를 교차하면 안됨
- 잘못된 괄호 사용의 예
 - (a(b)
 - a(b)c)
 - a{b(c[d]e)f)

괄호 검사 알고리즘(1)

- 문자열에 있는 괄호를 차례대로 조사하면서 왼쪽 괄호를 만나면 스택에 삽입하고, 오른쪽 괄호를 만나면 스택에서 top 괄호를 삭제한 후 오른쪽 괄호와 짝이 맞는지를 검사
- 이 때, 스택이 비어 있으면 조건 1 또는 조건 2 등을 위반하게 되고 괄호의 짝이 맞지 않으면 조건 3 등에 위반
- 마지막 괄호까지 조사한 후에도 스택에 괄호가 남아 있으면 조건 1에 위반되므로 0(거짓)을 반환하고, 그렇지 않으면 1(참)을 반환

괄호 검사 알고리즘(2)

check_matching(expr)

while (입력 expr의 끝이 아니면)

ch ← expr의 다음 글자

switch(ch)

case '(': **case** '[': **case** '{':

ch를 스택에 삽입

break

case ')': **case** ']': **case** '}':

if (스택이 비어 있으면)

then 오류

else 스택에서 open_ch를 꺼낸다

if (ch 와 open_ch가 같은 짝이 아니면)

then 오류 보고

break

if(스택이 비어 있지 않으면)

then 오류

왼쪽 괄호이면
스택에 삽입

오른쪽 괄호이면
스택에서 삭제비교

괄호 검사 코드(1)

```
int check_matching(char *in)
{
    StackType s;
    char ch, open_ch;
    int i, n = strlen(in);
    init(&s);

    for (i = 0; i < n; i++) {
        ch = in[i];
        switch(ch){
            case '(': case '[': case '{':
                push(&s, ch);
                break;
```

괄호 검사 코드(2)

```
case ')': case ']': case '}':
    if(is_empty(&s)) return FALSE;
    else {
        open_ch = pop(&s);
        if ((open_ch == '(' && ch != ')') || (open_ch == '[' && ch != ']') ||
            (open_ch == '{' && ch != '}')) {
            return FALSE;
        }
        break;
    }
}
if(!is_empty(&s)) return FALSE;
return TRUE;
}

int main()
{
    if( check_matching("{ A[(i+1)]=0; }") == TRUE )
        printf("괄호검사성공\n");
    else
        printf("괄호검사실패\n");
}
```

스택의 응용: 수식 계산

- 수식의 표기 방법: 전위(prefix), 중위(infix), 후위(postfix)

중위 표기법	전위 표기법	후위 표기법
$2 + 3 * 4$	$+ 2 * 3 4$	$2 3 4 * +$
$a * b + 5$	$+ 5 * a b$	$a b * 5 +$
$(1 + 2) + 7$	$+ 7 + 1 2$	$1 2 + 7 +$

- 컴퓨터에서의 수식 계산 순서
 - 중위표기식-> 후위표기식->계산
 - $2 + 3 * 4 \rightarrow 2 3 4 * + \rightarrow 14$
 - 스택 사용

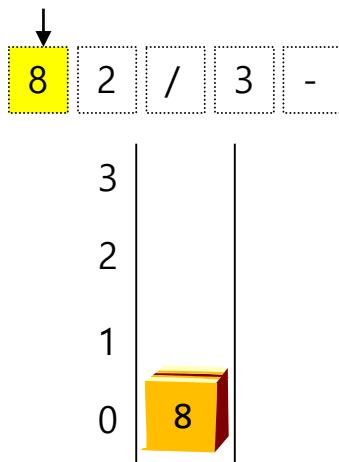
후위 표기식 계산(1)

- 수식을 왼쪽에서 오른쪽으로 스캔하여, 피연산자이면 스택에 저장하고 연산자이면 필요한 수만큼의 피연산자를 스택에서 꺼내 연산을 실행하고 연산의 결과를 다시 스택에 저장

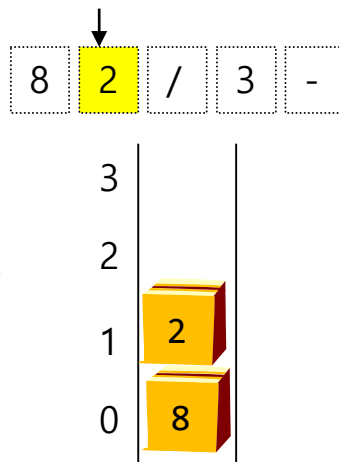
- (예) $8\ 2\ /\ 3\ -\ 3\ 2\ *\ +$
 $(8\ /\ 2\ -\ 3\ +\ 3\ *\ 2)$

토큰	스택						
	[0]	[1]	[2]	[3]	[4]	[5]	[6]
8	8						
2	8	2					
/	4						
3	4	3					
-	1						
3	1	3					
2	1	3	2				
*	1	6					
+	7						

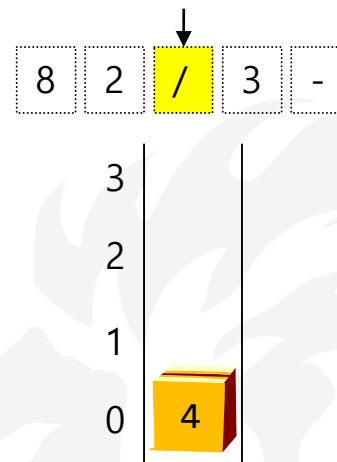
후위 표기식 계산(2)



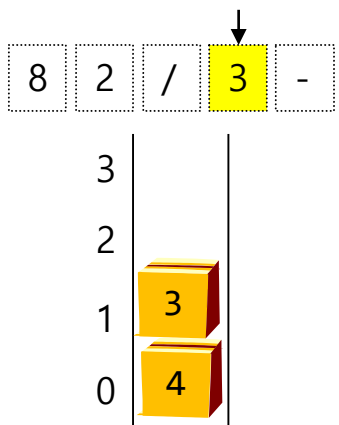
피연산자-> 삽입



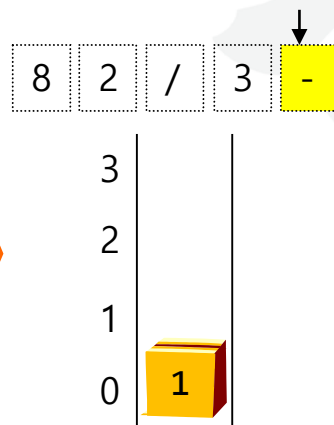
피연산자-> 삽입



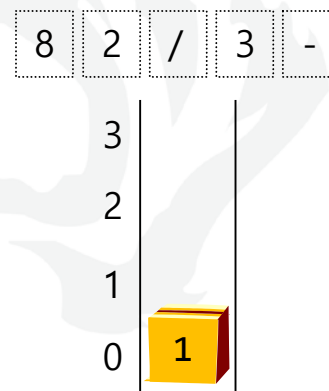
연산자-> $8/2=4$ 삽입



피연산자-> 삽입



연산자-> $4-3=1$ 삽입



종료->전체 연산 결과=1

후위 표기 수식 계산 알고리즘

스택 s 를 생성하고 초기화한다.

for 항목 in 후위표기식

do if (항목이 피연산자이면)

push(s , item)

if (항목이 연산자 op 이면)

then second \leftarrow pop(s)

first \leftarrow pop(s)

result \leftarrow first op second // op 는 $+ - * /$ 중하나

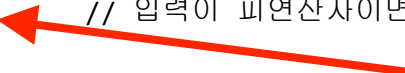
push(s , result)

final_result \leftarrow pop(s);

후위 표기식 계산 코드

```
eval(char exp[])
{
    int op1, op2, value, i=0;
    int len = strlen(exp);
    char ch;
    StackType s;

    init(&s);
    for( i=0; i<len; i++){
        ch = exp[i];
        if( ch != '+' && ch != '-' && ch != '*' && ch != '/' ){
            value = ch - '0'; // 입력이 피연산자이면
            push(&s, value);
        }
        else{ //연산자이면 피연산자를 스택에서 제거
            op2 = pop(&s);
            op1 = pop(&s);
            switch(ch){ //연산을 수행하고 스택에 저장
                case '+': push(&s, op1+op2); break;
                case '-': push(&s, op1-op2); break;
                case '*': push(&s, op1*op2); break;
                case '/': push(&s, op1/op2); break;
            }
        }
    }
    return pop(&s);
}
```



중위 표기식 -> 후위 표기식

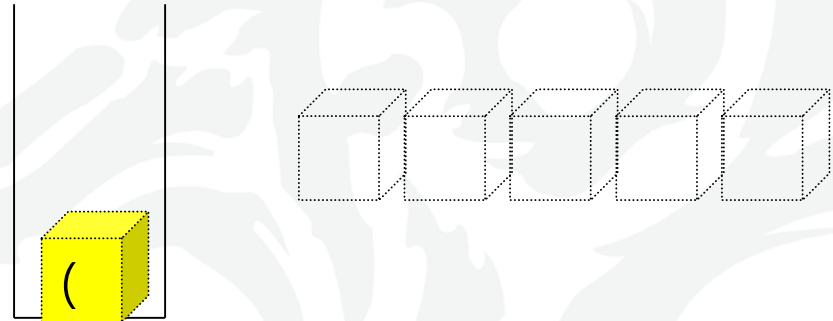
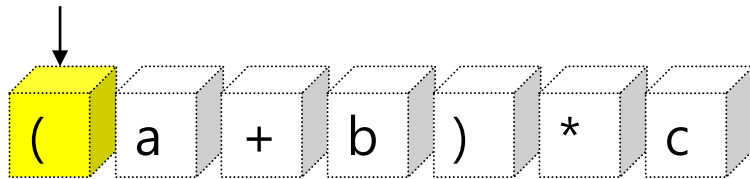
■ 중위표기와 후위표기

- 중위 표기법과 후위 표기법의 공통점은 피연산자의 순서는 동일
- 연산자들의 순서만 다름(우선순위순서)
 - 연산자만 스택에 저장했다가 출력하면 됨
- $2 + 3 * 4 \rightarrow 2\ 3\ 4\ *\ +$

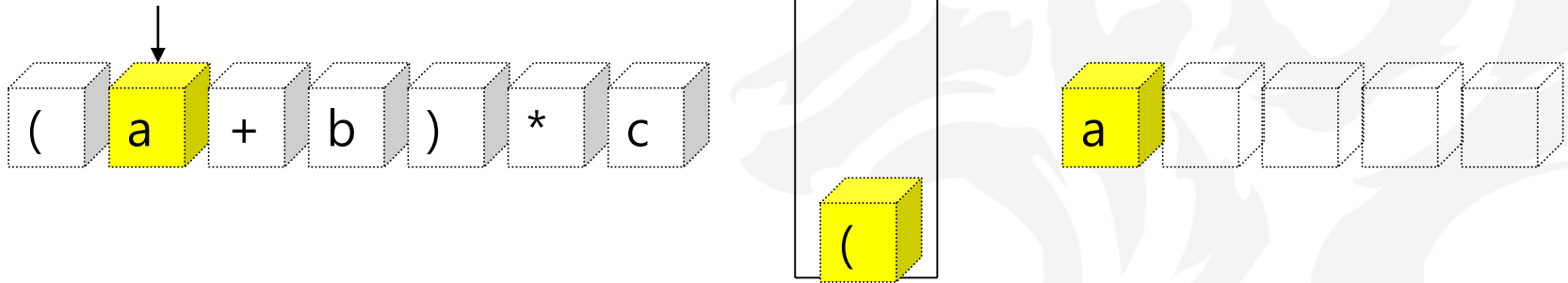
■ 알고리즘

- 피연산자를 만나면 그대로 출력
- 연산자를 만나면 스택에 저장했다가 스택보다 우선 순위가 낮은 연산자가 나오면 그때 출력
- 왼쪽 괄호는 우선순위가 가장 낮은 연산자로 취급
- 오른쪽 괄호가 나오면 스택에서 왼쪽 괄호위에 쌓여있는 모든 연산자를 출력

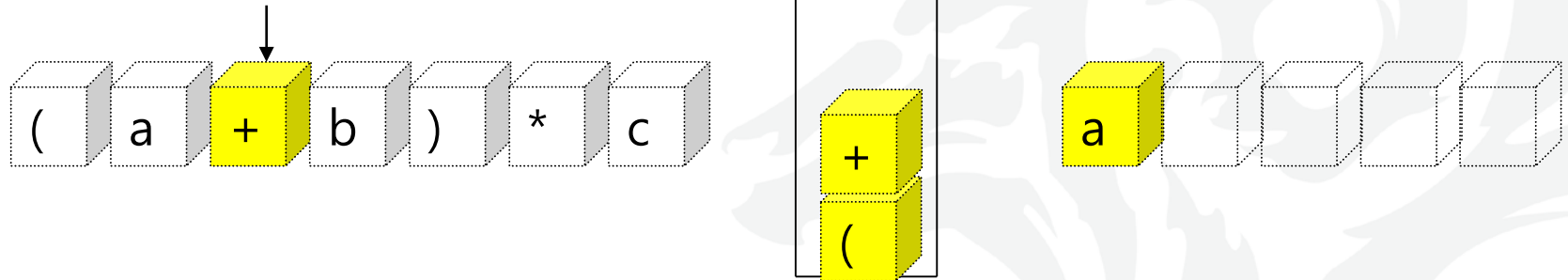
중위 표기식 -> 후위 표기식 예



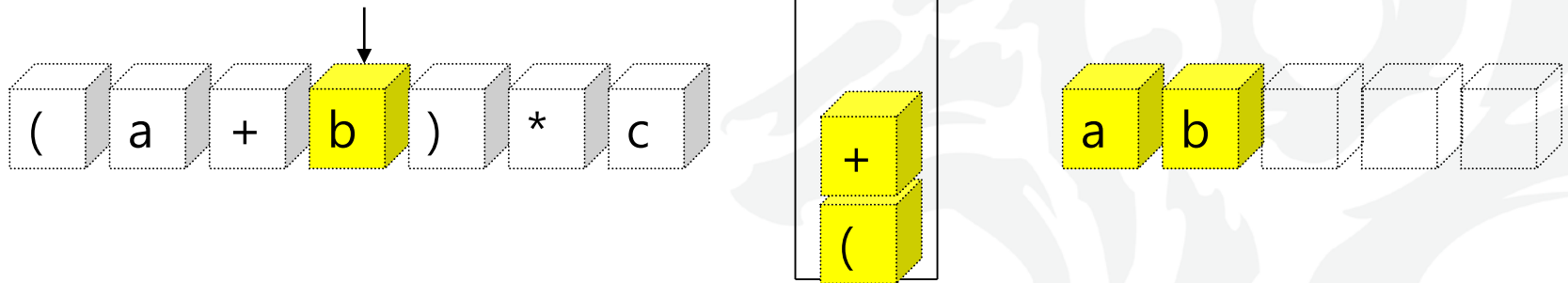
중위 표기식 -> 후위 표기식 예



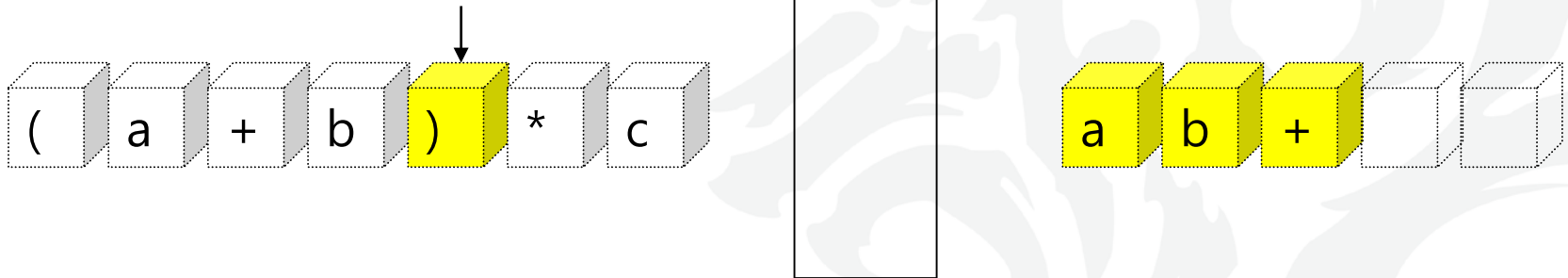
중위 표기식 -> 후위 표기식 예



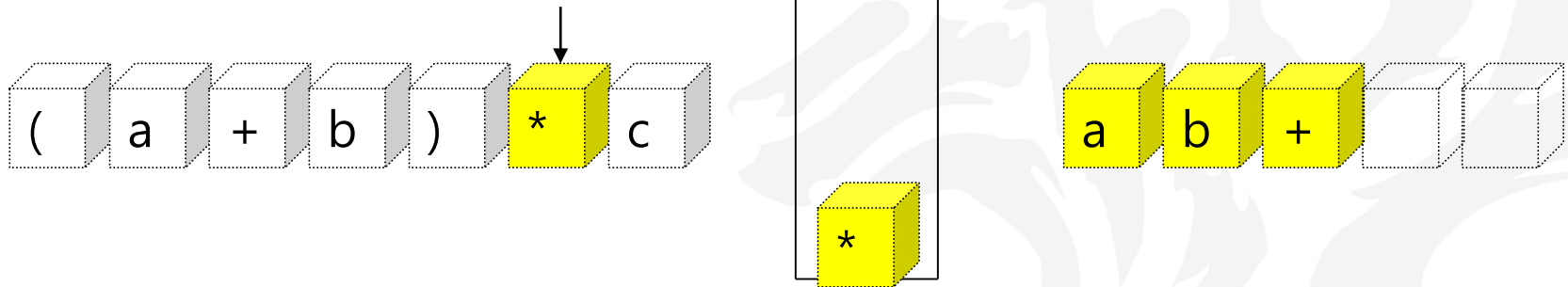
중위 표기식 -> 후위 표기식 예



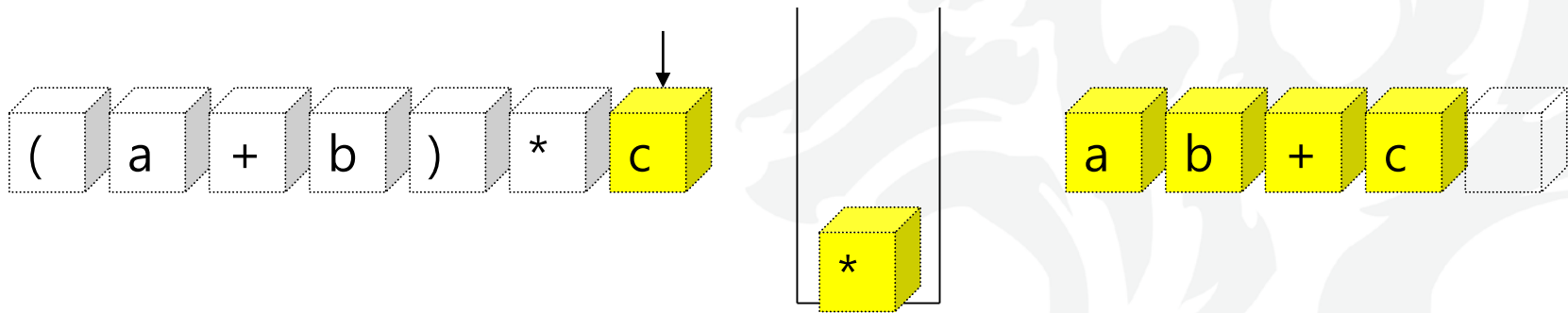
중위 표기식 -> 후위 표기식 예



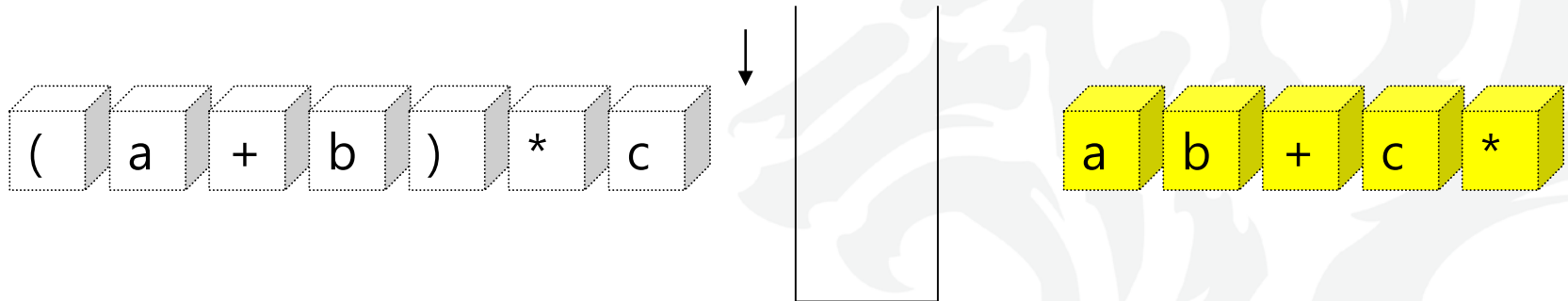
중위 표기식 -> 후위 표기식 예



중위 표기식 -> 후위 표기식 예



중위 표기식 -> 후위 표기식 예



중위 표기식 -> 후위 표기식 알고리즘(1)

```
infix_to_postfix(exp)
스택 s를 생성하고 초기화
while (exp에 처리할 문자가 남아 있으면)
    ch ← 다음에 처리할 문자
    switch (ch)
        case 연산자:
            while ( peek(s)의 우선순위 ≥ ch의 우선순위 )
                do e ← pop(s)
                e를 출력
            push(s, ch);
            break;
        case 왼쪽 괄호:
            push(s, ch);
            break;
        case 오른쪽 괄호:
            e ← pop(s);
            while( e ≠ 왼쪽괄호 )
                do e를 출력
                e ← pop(s)
            break;
        case 피연산자:
            ch를 출력
            break;
while( not is_empty(s) )
    do e ← pop(s)
    e를 출력
```

중위 표기식 -> 후위 표기식 알고리즘(2)

```
// 중위 표기 수식 -> 후위 표기 수식
void infix_to_postfix(char exp[])
{
    int i = 0;
    char ch, top_op;
    int len = strlen(exp);
    StackType s;

    init(&s);          // 스택 초기화
    for (i = 0; i < len; i++) {
        ch = exp[i];
        // 연산자이면
        switch (ch) {
            case '+': case '-': case '*': case '/': // 연산자
                // 스택에 있는 연산자의 우선순위가 더 크거나 같으면 출력
                while (!is_empty(&s) && (prec(ch) <= prec(peek(&s))))
                    printf("%c", pop(&s));
                push(&s, ch);
                break;
            case '(': // 왼쪽 괄호
                push(&s, ch);
                break;
        }
    }
}
```

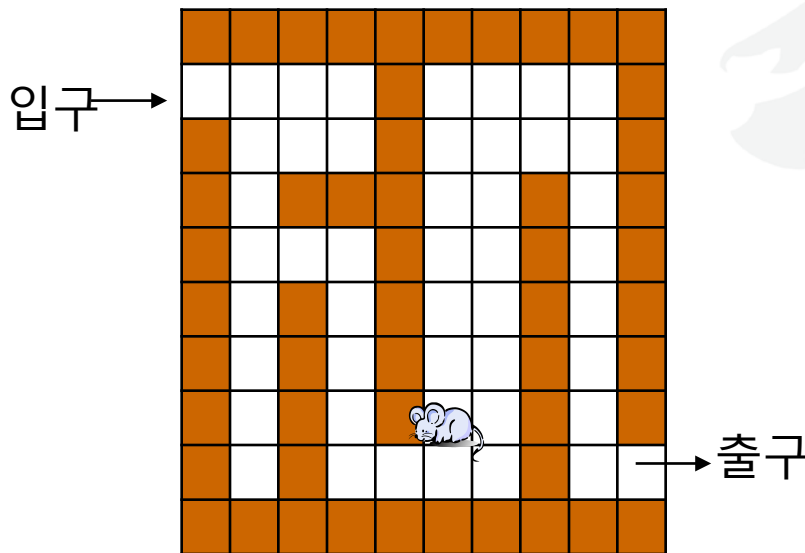
중위 표기식 -> 후위 표기식 알고리즘(3)

```
case ')': // 오른쪽 괄호
    top_op = pop(&s);
    // 왼쪽 괄호를 만날때까지 출력
    while( top_op != '(' ){
        printf("%c", top_op);
        top_op = pop(&s);
    }
    break;
default: // 피연산자
    printf("%c", ch);
    break;
}
}
while( !is_empty(&s) ) // 스택에 저장된 연산자들 출력
    printf("%c", pop(&s));
}
//
main()
{
    infix_to_postfix("(2+3)*4+9");
}
```

스택의 응용: 미로탐색문제

■ 미로탐색문제

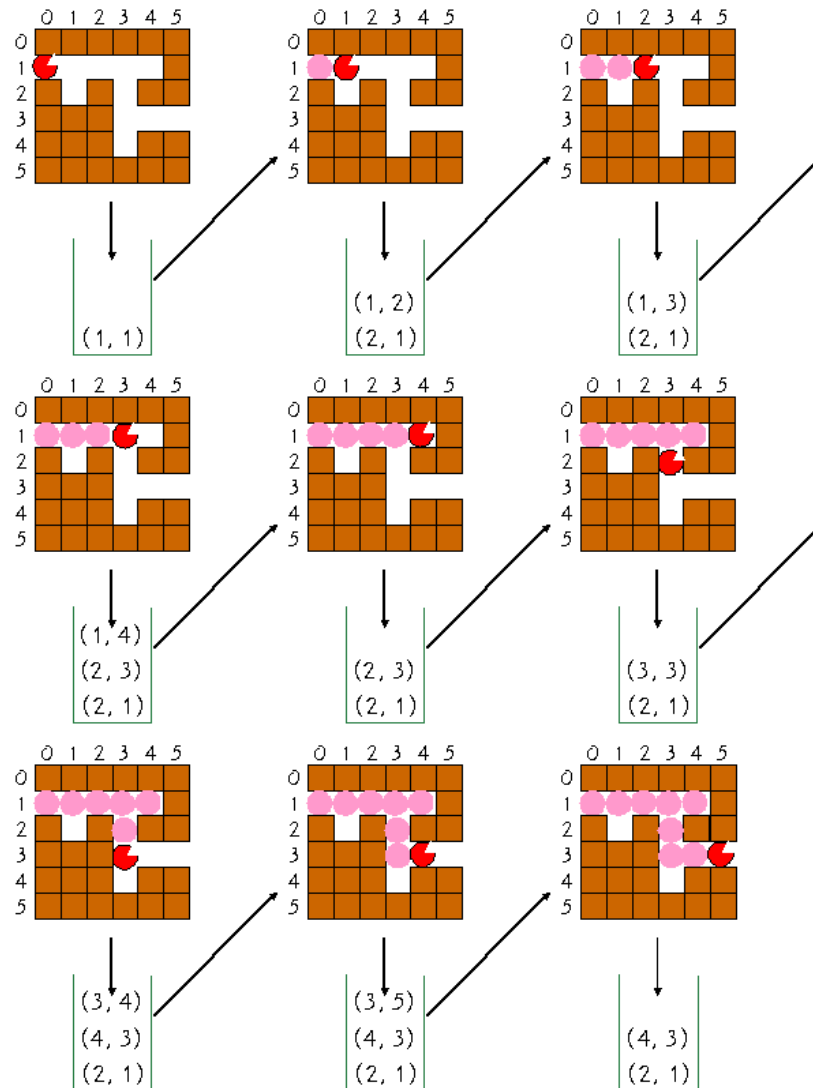
- 미로에 갇힌 생쥐가 출구를 찾는 문제
- 체계적인 방법 필요
 - 현재의 위치에서 가능한 방향을 스택에 저장해 놓았다가 막다른 길을 만나면 스택에서 다음 탐색 위치를 꺼냄



→	1	1	1	1	1	1	1	1	1
	0	0	0	0	1	0	0	0	1
	1	0	0	0	1	0	0	0	1
	1	0	1	1	1	0	0	1	0
	1	0	0	0	1	0	0	1	0
	1	0	1	0	1	0	0	1	0
	1	0	1	0	1	0	0	1	0
	1	0	1	0	1	0	0	1	0
	1	0	1	0	1	m	0	1	0
	1	0	1	0	0	0	0	1	0
	1	1	1	1	1	1	1	1	1

→

미로탐색문제 알고리즘(1)



미로탐색문제 알고리즘(2)

스택 s 과 출구의 위치 x , 현재 생쥐의 위치를 초기화
while(현재의 위치가 출구가 아니면)
 do 현재 위치를 방문한 것으로 표기
 if(현재 위치의 사방 위치가 아직 방문되지 않았고 갈 수 있으면)
 then 그 위치들을 스택에 **push**
 if(**is_empty**(s))
 then 실패
 else 스택에서 하나의 위치를 꺼내어 현재 위치로 만든다;
성공;

미로탐색문제 코드(1)

```
#define MAX_STACK_SIZE 100
#define MAZE_SIZE 6

typedef struct StackObjectRec {
    short r;
    short c;
} StackObject;

StackObject stack[MAX_STACK_SIZE];
int top = -1;
StackObject here={1,0}, entry={1,0};

char maze[MAZE_SIZE][MAZE_SIZE] = {
    {'1', '1', '1', '1', '1', '1'},
    {'e', '0', '1', '0', '0', '1'},
    {'1', '0', '0', '0', '1', '1'},
    {'1', '0', '1', '0', '1', '1'},
    {'1', '0', '1', '0', '0', 'x'},
    {'1', '1', '1', '1', '1', '1'},
};
```



미로탐색문제 코드(2)

```
void pushLoc(int r, int c)
{
    if( r < 0 || c < 0 ) return;
    if( maze[r][c] != '1' && maze[r][c] != '.' ){
        StackObject tmp;
        tmp.r = r;
        tmp.c = c;
        push(tmp);
    }
}

void printMaze(char m[MAZE_SIZE][MAZE_SIZE])
{
    ...
}
```

미로탐색문제 코드(3)

```
void printStack()
{
    int i;
    for(i=5;i>top;i--)
        printf("|      |\n");
    for(i=top;i>=0;i--)
        printf("|(%01d,%01d)|\n", stack[i].r, stack[i].c);
    printf("-----\n");
}
```

미로탐색문제 코드(4)

```
void main()
{
    int r,c;
    here = entry;
    printMaze(maze);
    printStack();
    while ( maze[here.r][here.c]!='x' ){
        printMaze(maze);
        r = here.r;
        c = here.c;
        maze[r][c] = '.';
        pushLoc(r-1,c);
        pushLoc(r+1,c);
        pushLoc(r,c-1);
        pushLoc(r,c+1);
    }
```

미로탐색문제 코드(4)

```
    printStack();  
    if( isEmpty() ){  
        printf("실패\n");  
        return;  
    }  
    else  
        here = pop();  
    printMaze(maze);  
    printStack();  
    getch();  
}  
printf("성공\n");  
}
```



Week 5: Stack

