

자료구조(Data Structures)

5장. 스택

담당 교수 : 조 미경

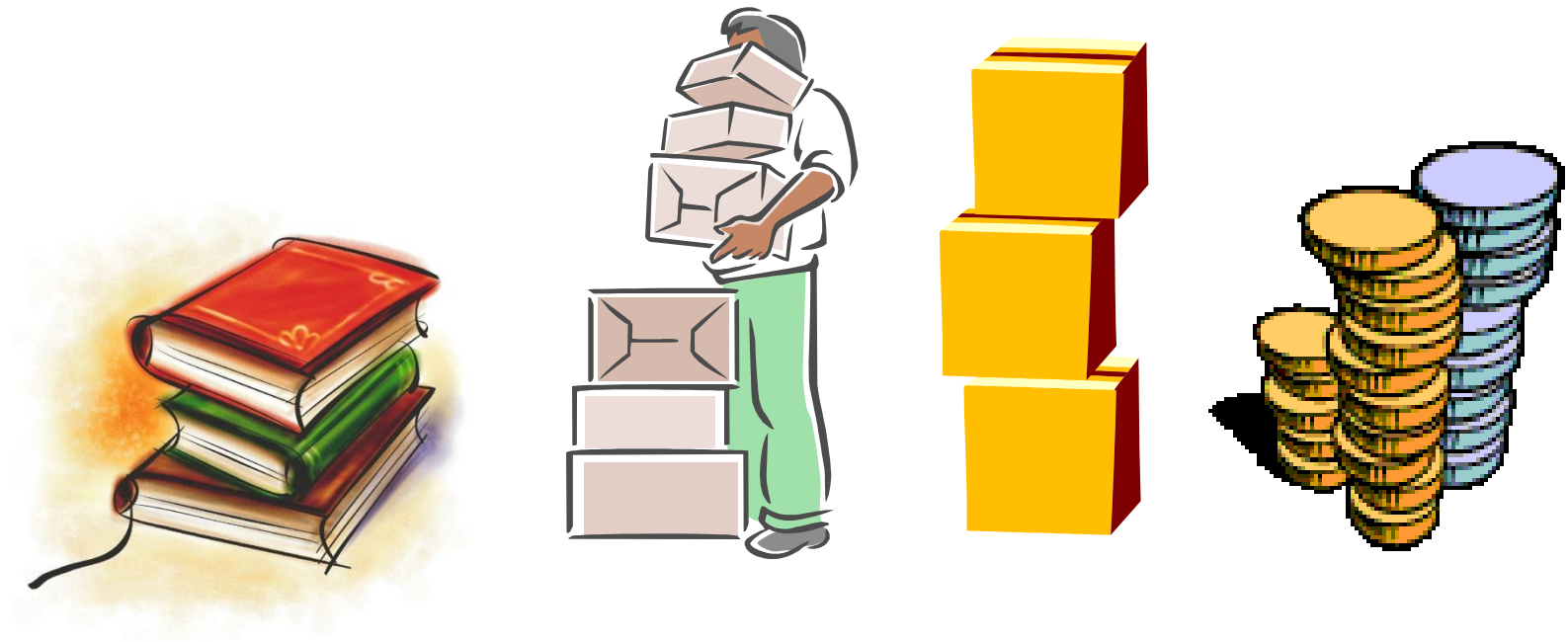
이번 장에서 학습할 내용



- * 스택이란?
- * 스택 추상 자료형
- * 배열로 구현한 스택
- * 연결리스트로 구현한 스택
- * 스택 활용 문제:
 - 괄호 검사,
 - 수식의 계산,
 - 미로탐색문제

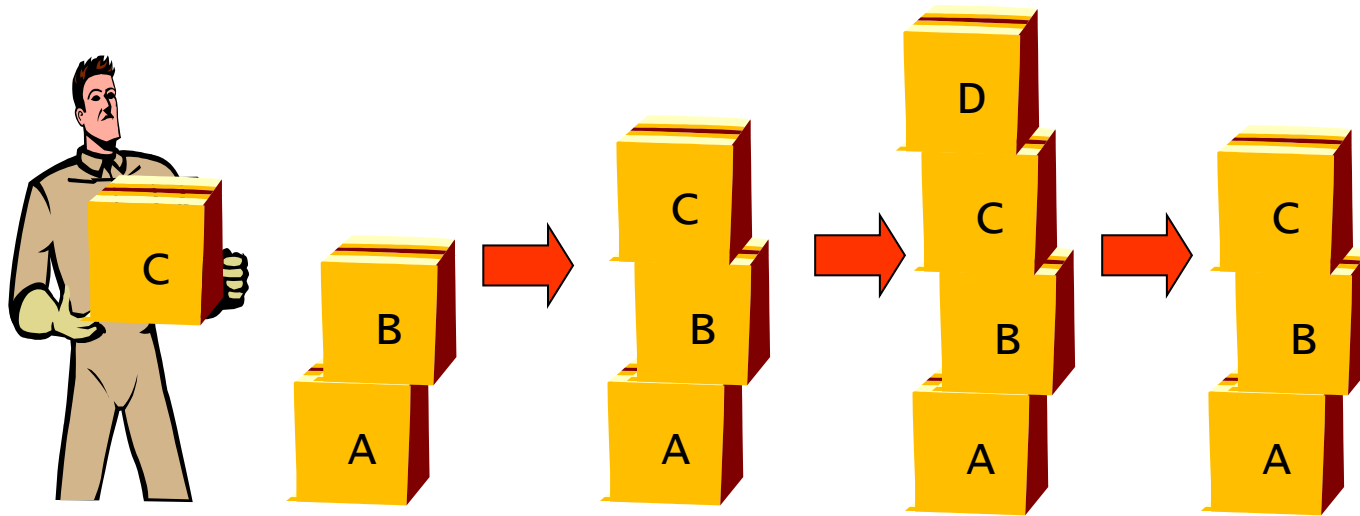
스택이란?

- 스택(stack): 쌓아놓은 더미

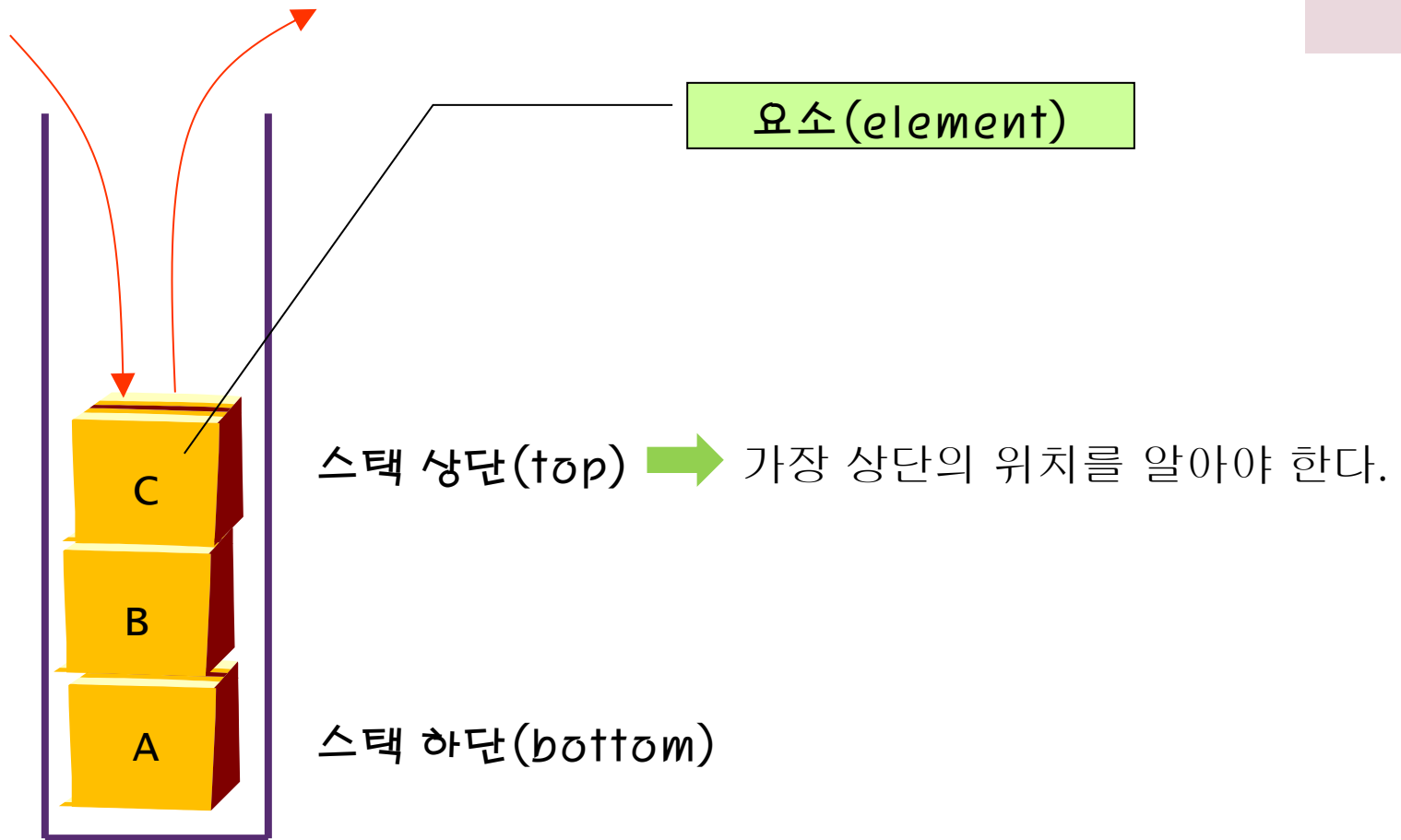


스택의 특징

- **후입선출(LIFO: Last-In First-Out)**: 가장 최근에 들어온 데이터가 가장 먼저 나감.



스택의 구조

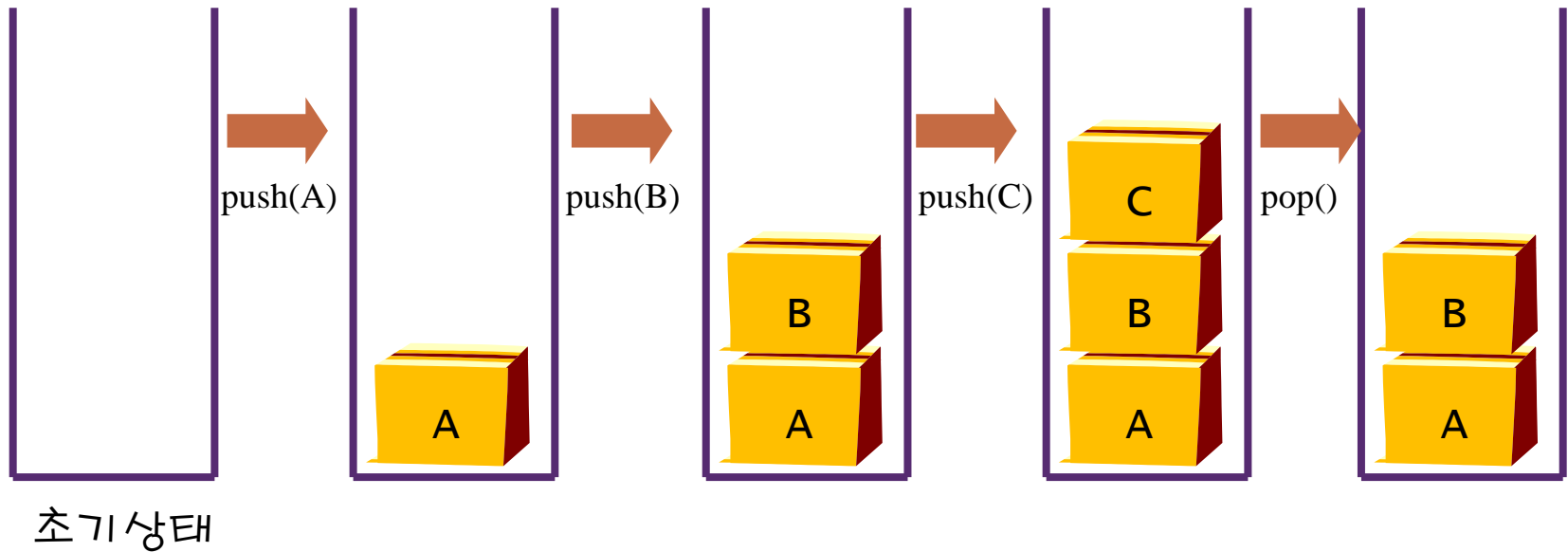


스택 추상데이터타입(ADT)

- 객체: n개의 element형의 요소들의 선형 리스트
- 연산:
 - `create()` ::= 스택을 생성한다.
 - `is_empty(s)` ::= 스택이 비어있는지를 검사한다.
 - `is_full(s)` ::= 스택이 가득 찼는가를 검사한다.
 - `push(s, e)` ::= 스택의 맨 위에 요소 `e`를 추가한다.
 - `pop(s)` ::= 스택의 맨 위에 있는 요소를 삭제한다.
 - `peek(s)` ::= 스택의 맨 위에 있는 요소를 삭제하지 않고 반환한다.

스택의 연산

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



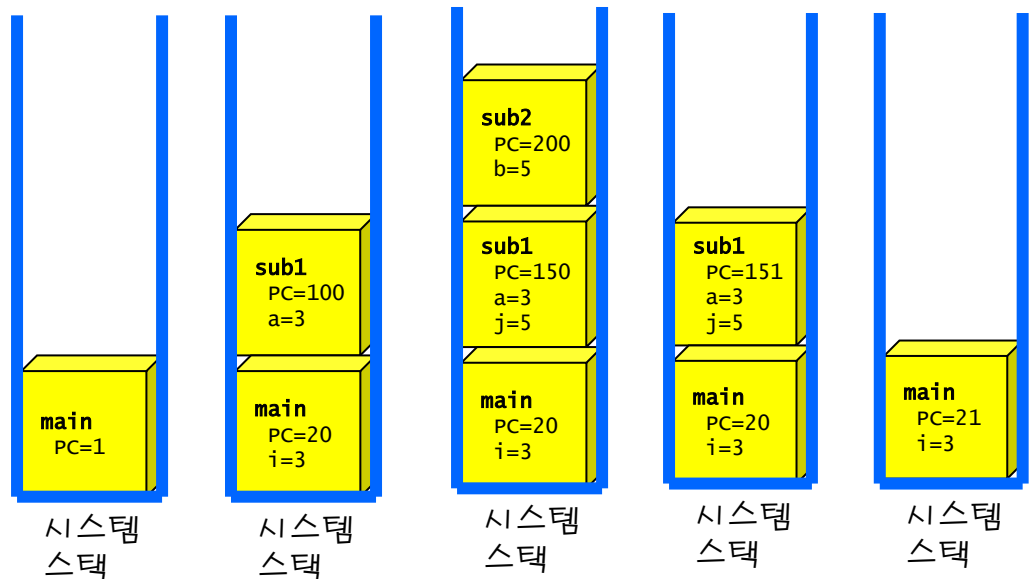
스택의 연산

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

스택의 용도

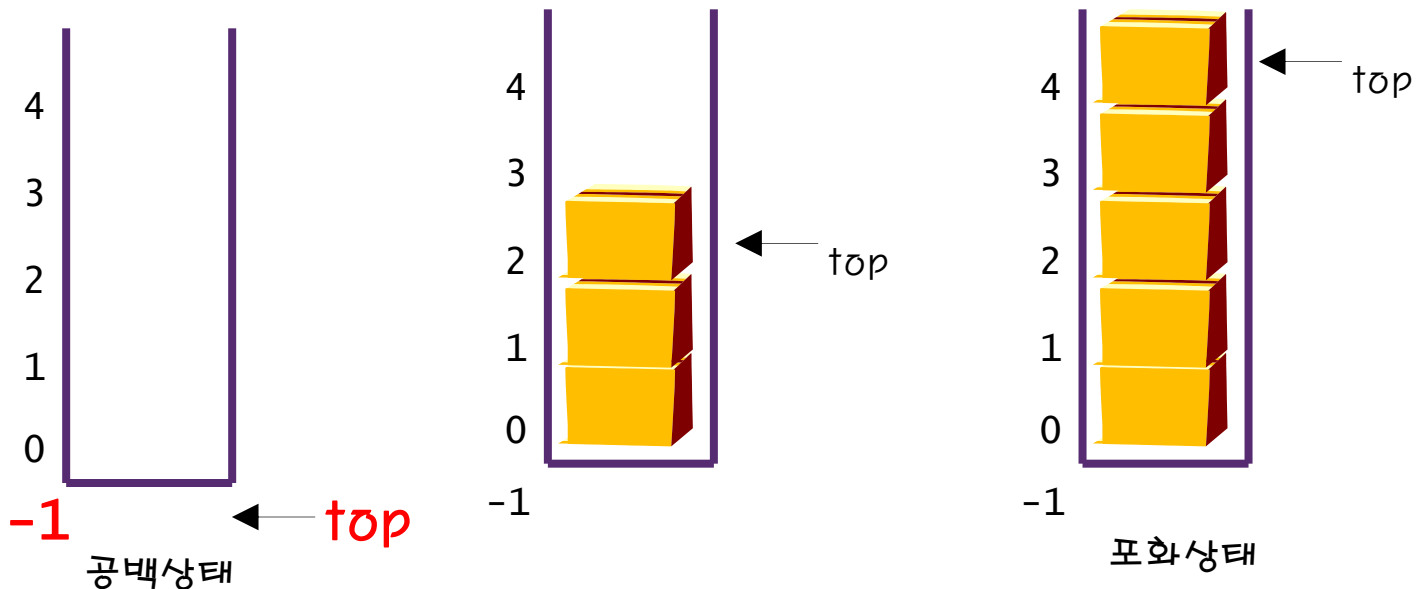
- 입력과 역순의 출력이 필요한 경우
 - ▶ 에디터에서 되돌리기(undo) 기능
 - ▶ 함수호출에서 복귀주소 기억
 - ▶ 웹 브라우저의 뒤로 버튼

```
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

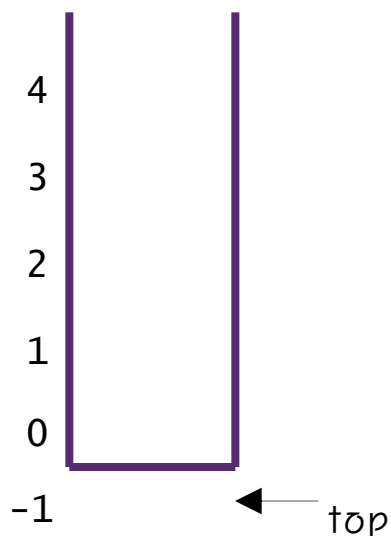
- 1차원 배열 `stack[]`
- 스택에서 가장 최근에 입력되었던 자료를 가리키는 `top` 변수
- 가장 먼저 들어온 요소는 `stack[0]`에, 가장 최근에 들어온 요소는 `stack[top]`에 저장
- 스택이 공백상태이면 `top`은 -1



is_empty(), is_full() 연산의 구현

```
is_empty(s)
```

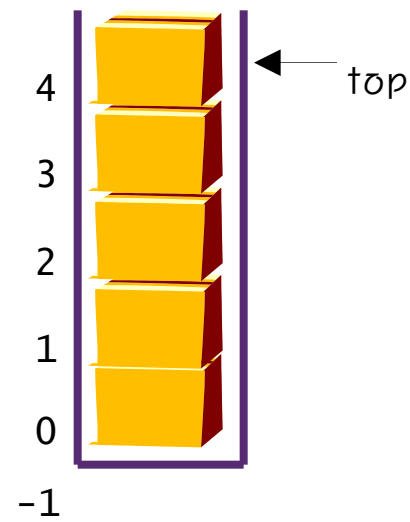
```
if top = -1  
    then return TRUE  
    else return FALSE
```



공백상태

```
is_full(s)
```

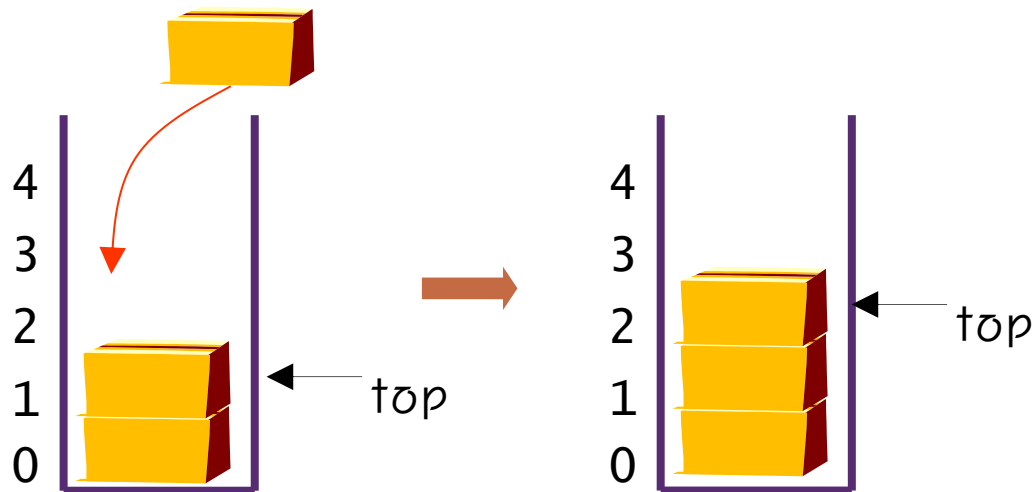
```
if top = (MAX_STACK_SIZE-1)  
    then return TRUE  
    else return FALSE
```



포화상태

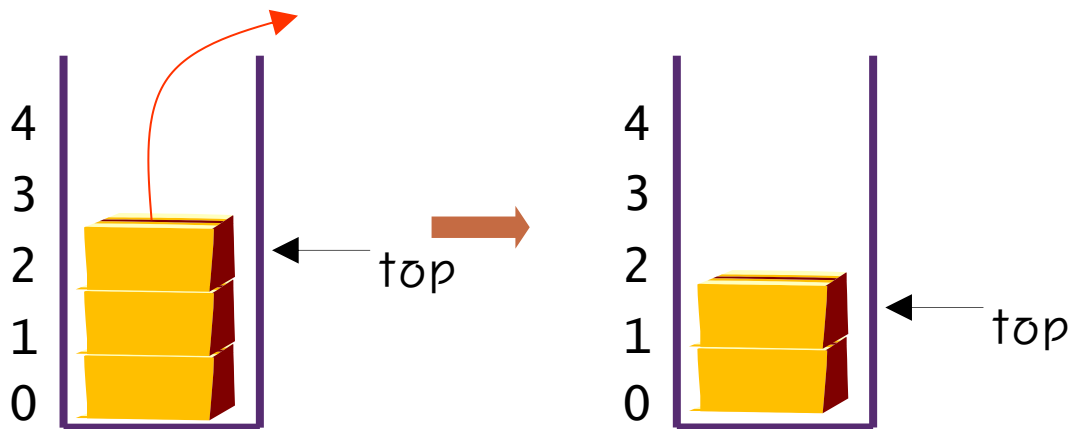
push() 연산

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



pop() 연산

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



스택 구현(C)

```
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);
}

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

배열의 요소는 element타입
으로 선언

관련 데이터를 구조체로 묶어
서 함수의 파라미터로 전달

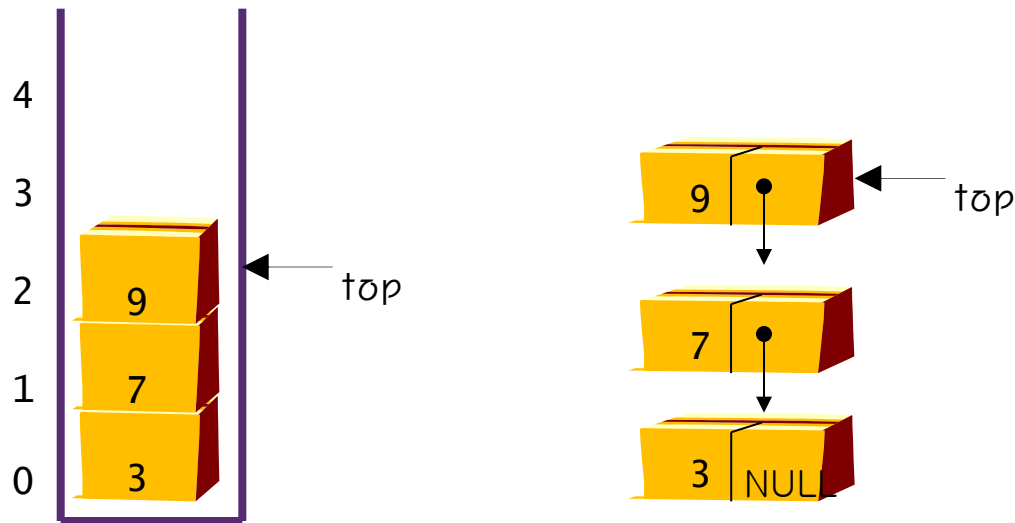
```
void push(StackType *s, element item) { // 삽입함수
    if( is_full(s) ) {
        fprintf(stderr,"스택 포화 에러\n");
        return;
    }
    else s->stack[++(s->top)] = item;
}

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

element peek(StackType *s) { // 피크함수
    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 걸린다.



연결된 스택 정의

```
typedef int element;
```

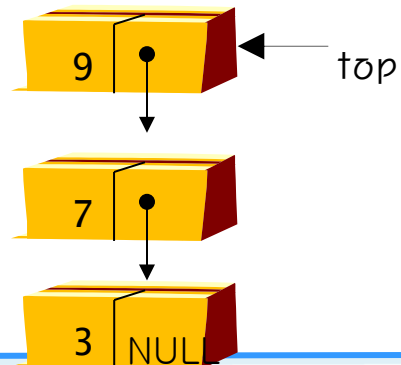
요소의 타입

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

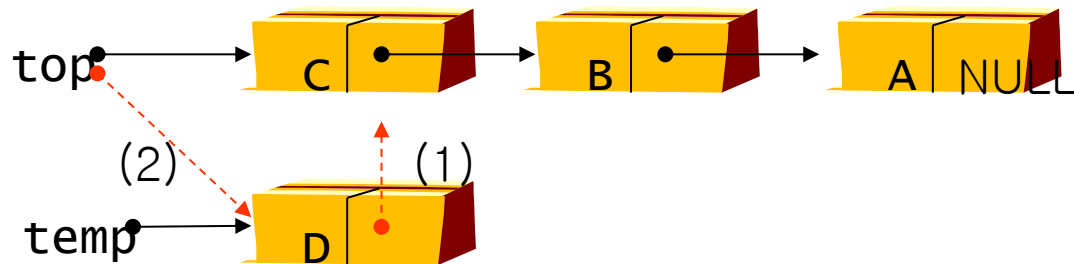
노드의 타입

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

연결된 스택의 관련 데이터

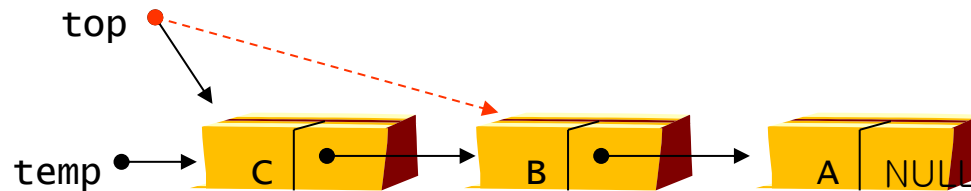


연결된 스택에서 push() 연산



```
void push(LinkedStackType *s, element item) { // 삽입 함수
    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 스택에서 pop() 연산



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

스택의 응용: 괄호검사

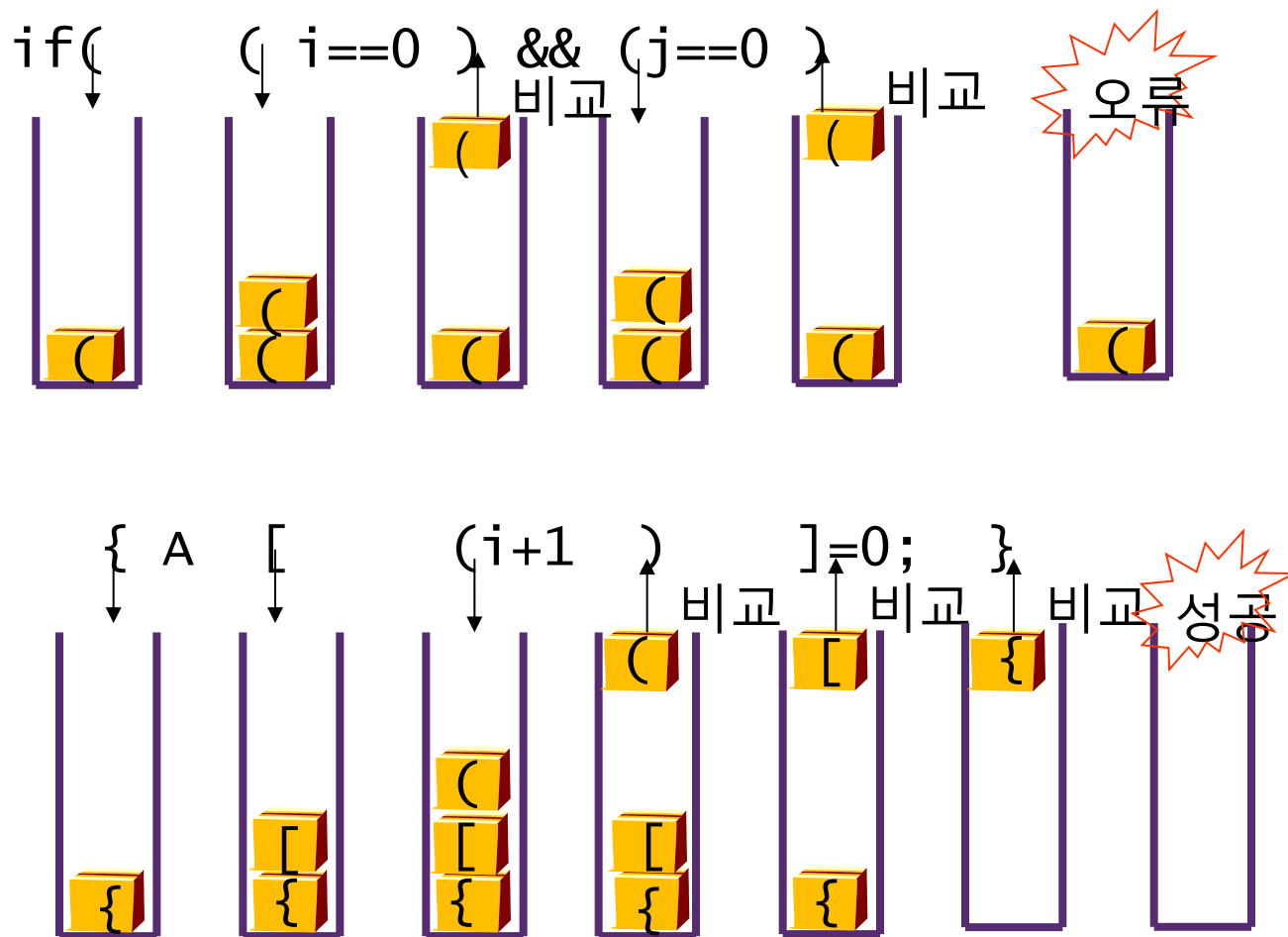
- 괄호의 종류: 대괄호 ('[', ']'), 중괄호 ('{', '}'), 소괄호 ('(', ')')
- 조건
 1. 왼쪽 괄호의 개수와 오른쪽 괄호의 개수가 같아야 한다.
 2. 같은 괄호에서 왼쪽 괄호는 오른쪽 괄호보다 먼저 나와야 한다.
 3. 괄호 사이에는 포함 관계만 존재한다.
- 잘못된 괄호 사용의 예

(a(b)

a(b)c)

a{b(c[d]e}f)

스택을 이용한 괄호 검사



알고리즘

● 알고리즘의 개요

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

괄호 검사 알고리즘

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 프로그램

● C로 구현된 함수

```
//
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;
```

```
            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;
}
```


괄호 검사 프로그램

- 메인 함수

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

- ▶ 수식을 입력하여 check_matching()함수의 인자로 넘기도록 메인 함수를 수정해 보자.

수식의 계산

- 수식의 표기방법:

- ▶ 전위(prefix), 중위(infix), 후위(postfix)

중위 표기법	전위 표기법	후위 표기법
$2+3*4$	$+2*34$	$234*+$
$a*b+5$	$+5*ab$	$ab*5+$
$(1+2)+7$	$+7+12$	$12+7+$

- 컴퓨터에서의 수식 계산순서

- ▶ 중위표기식-> 후위표기식->계산
- ▶ $2+3*4 \rightarrow 234*+ \rightarrow 14$
- ▶ 모두 스택을 사용
- ▶ 먼저 후위표기식의 계산법을 알아보자

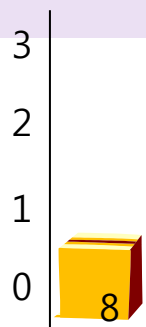
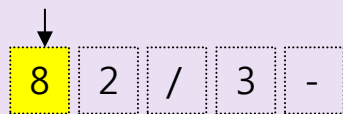
후위 표기식의 계산

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

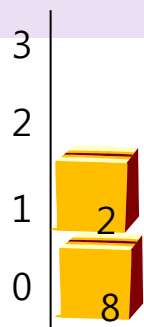
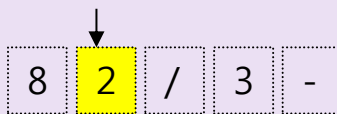
● (예) $82/3-32^{*}+$

스택에서 변화되는 값

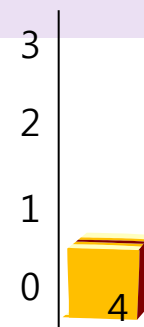
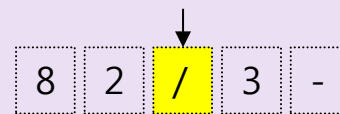
토 큰	스택						
	[0]	[1]	[2]	[3]	[4]	[5]	[6]
8	8						
2	8	2					
/	4						
3	4	3					
-	1						
3	1	3					
2	1	3	2				
*	1	6					
+	7						



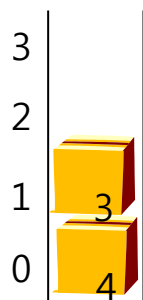
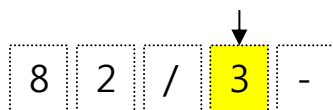
피연산자-> 삽입



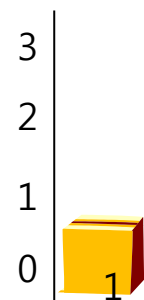
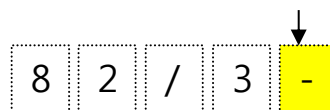
피연산자-> 삽입



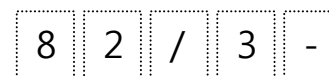
연산자-> $8/2=4$ 삽입



피연산자-> 삽입



연산자-> $4-1=1$ 삽입



종료-> 전체 연산 결과=1

후위 표기식 계산 알고리즘

```
스택 s를 생성하고 초기화한다.  
for 항목 in 후위표기식  
  do if (항목이 피연산자이면)  
    push(s, item)  
  if (항목이 연산자 op이면)  
    then second ← pop(s)  
    first ← pop(s)  
    result ← first op second // op 는 +-* /중의 하나  
    push(s, result)  
final_result ← pop(s);
```

```
// 후위 표기 수식 계산 함수
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);
}
```

후위 표기식 계산 프로그램

- 메인 함수

```
int main()
{
    printf("후위 수식 82/3-32*+ 계산 결과: %d \n", eval("82/3-32*") );
}
```

중위표기식->후위표기식

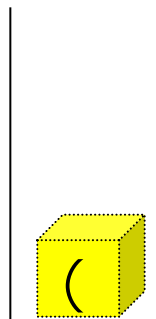
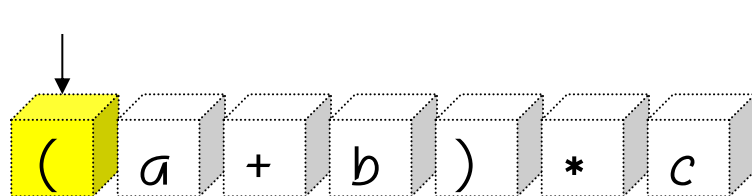
● 중위표기와 후위표기

- ▶ 중위 표기법과 후위 표기법의 공통점은 피연산자의 순서는 동일
- ▶ 연산자들의 순서만 다름(우선순위순서)
->연산자만 스택에 저장했다가 출력하면 된다.
- ▶ $2+3*4 \rightarrow 234*+$

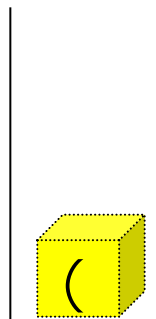
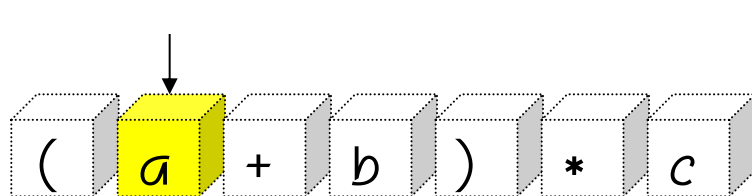
● 알고리즘

- 1) 피연산자를 만나면 그대로 출력
- 2) 연산자 op를 만나면 ‘스택’의 top에 있는 연산자와 비교하여,
 - ▶ ‘스택’의 top에 있는 연산자가 우선순위가 높으면 top에 있는 연산자를 출력하고 연산자 op는 스택에 삽입
- 3) 왼쪽 괄호는 우선순위가 가장 낮은 연산자로 취급
- 4) 오른쪽 괄호가 나오면 스택에서 왼쪽 괄호 위에 쌓여있는 모든 연산자를 출력

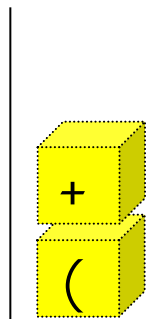
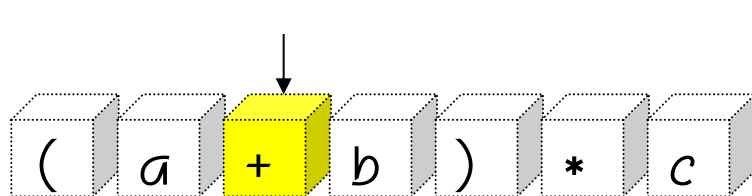
중위표기식->후위표기식



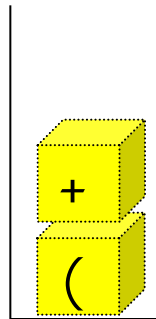
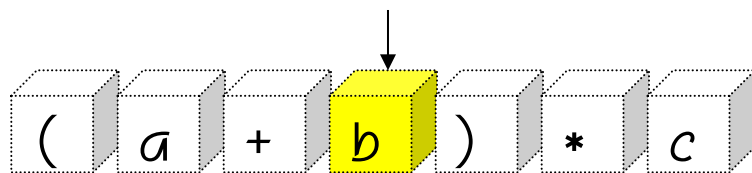
중위표기식->후위표기식



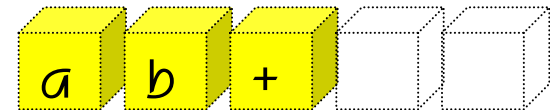
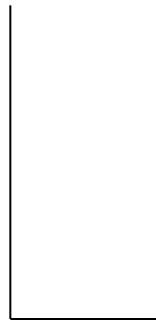
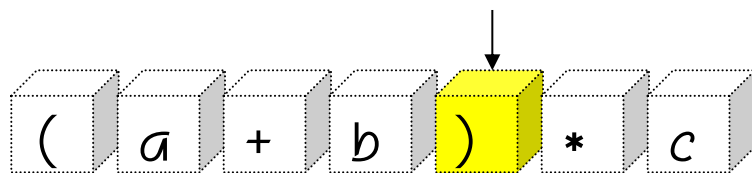
중위표기식->후위표기식



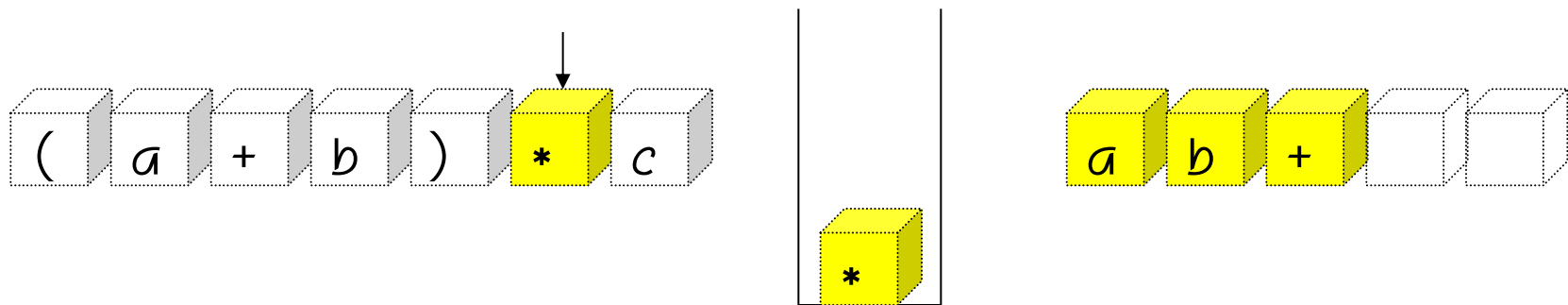
중위표기식->후위표기식



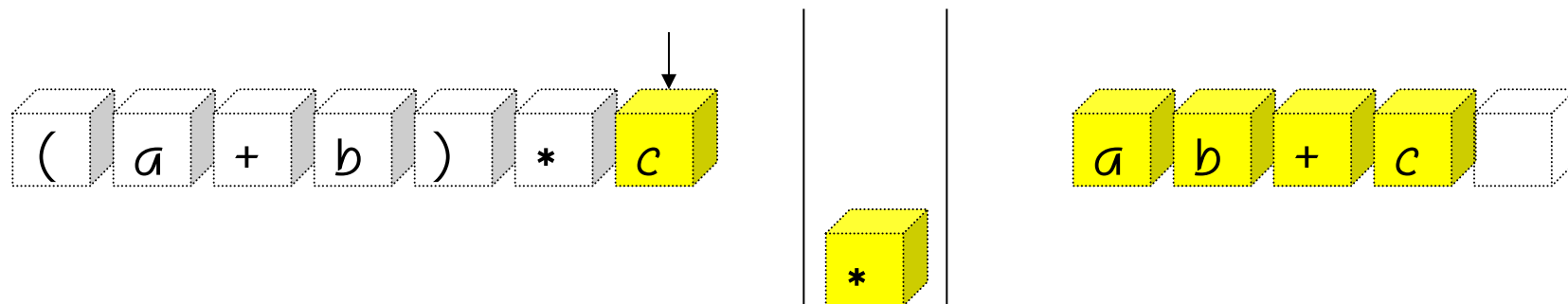
중위표기식->후위표기식



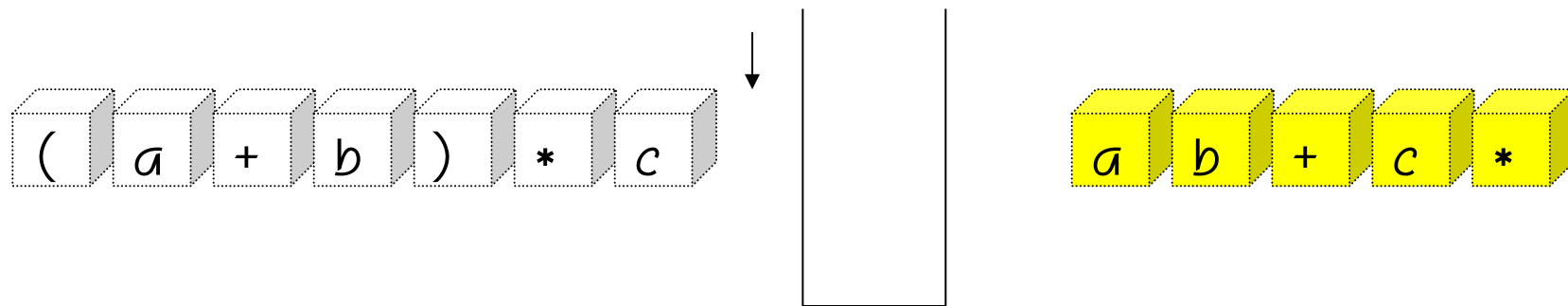
중위표기식->후위표기식



중위표기식->후위표기식



중위표기식->후위표기식




```
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->후위 변환 프로그램

```
// 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->후위 변환 프로그램

```
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;
}

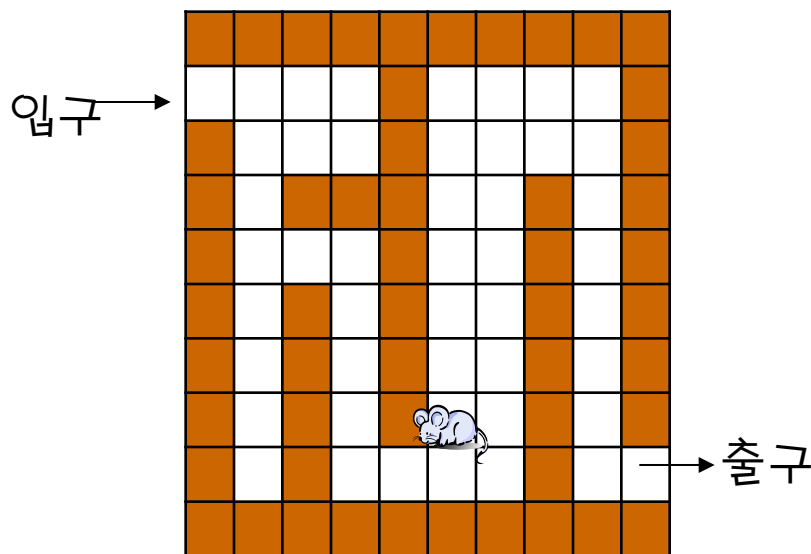
}

while( !is_empty(&s) ) // 스택에 저장된 연산자들 출력
    printf("%c", pop(&s));
}

//
main()
{
    infix_to_postfix("(2+3)*4+9");
}
```

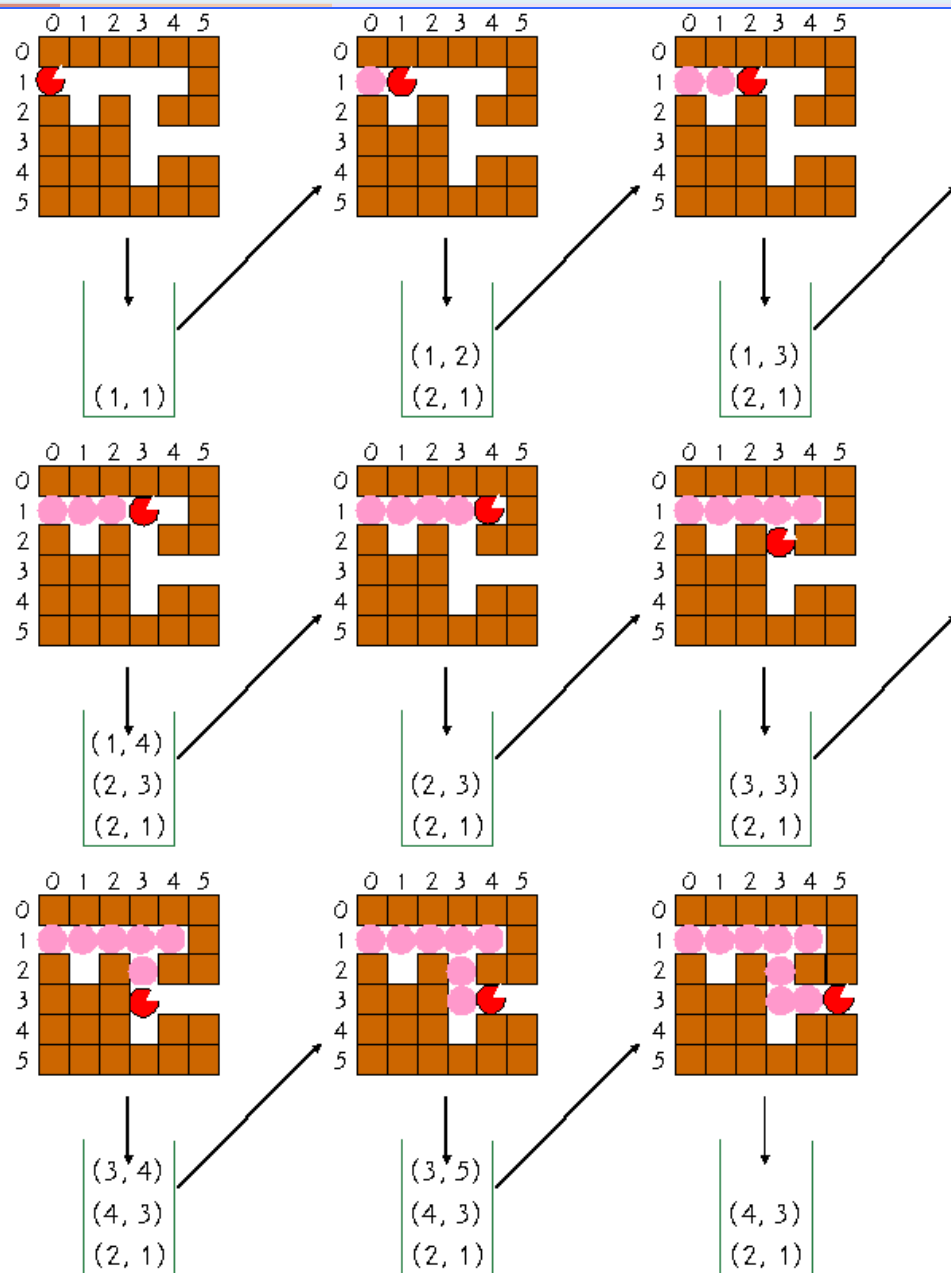
미로탐색문제

- 입구에서 출구까지 길을 찾는 문제
- 현재의 위치에서 가능한 방향을 스택에 저장해놓았다가 막다른 길을 만나면 스택에서 다음 탐색 위치를 꺼낸다.



```

      1 1 1 1 1 1 1 1 1 1
→ 0 0 0 0 1 0 0 0 0 1
    1 0 0 0 1 0 0 0 0 1
    1 0 1 1 1 0 0 1 0 1
    1 0 0 0 1 0 0 1 0 1
    1 0 1 0 1 0 0 1 0 1
    1 0 1 0 1 0 0 1 0 1
    1 0 1 0 1 m 0 1 0 1
    1 0 1 0 0 0 0 1 0 x →
    1 1 1 1 1 1 1 1 1 1
  
```



미로탐색 알고리즘

```
스택 s과 출구의 위치 x, 현재 생쥐의 위치를 초기화
while( 현재의 위치가 출구가 아니면 )
  do 현재위치를 방문한 것으로 표기
    if( 현재위치의 위, 아래, 왼쪽, 오른쪽 위치가 아직 방문되지 않았고 갈수 있으면 )
      then 그 위치들을 스택에 push
    if( is_empty(s) )
      then 실패
    else 스택에서 하나의 위치를 꺼내어 현재 위치로 만든다;
성공;
```

미로 프로그램

```
#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'},
};
```

미로 프로그램

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


미로 프로그램

```
void main()
{
    int r,c;
    StackType s;

    init(&s);
    here = entry;
    while ( maze[here.r][here.c]!='x' ){
        r = here.r;
        c = here.c;
        maze[r][c] = '.';
        pushLoc(&s, r-1,c);
        pushLoc(&s, r+1,c);
        pushLoc(&s, r,c-1);
        pushLoc(&s, r,c+1);
```

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

연습문제 1

- 아래의 중위 수식을 후위 수식으로 바꾸시오. 그리고 바꾸는 과정을 스택을 이용하여 그림으로 그리시오.

① $a * b + 2 * c * d * e - 1$

② $a / (b - c) + d * e$

③ $a + b * c - d * e$

연습문제 2

- 아래의 후위 수식이 계산되는 동안 스택 안의 값이 어떻게 변화되는지 아래 그림과 같이 그려 보시오.

▶ $1\ 2\ 3\ *\ +\ 5$

▶ $3\ 7\ 2\ /\ 4\ *\ +\ 2\ 5\ /\ -\ 2\ 3\ 4\ /\ +\ 2\ 1\ +\ /\ +$

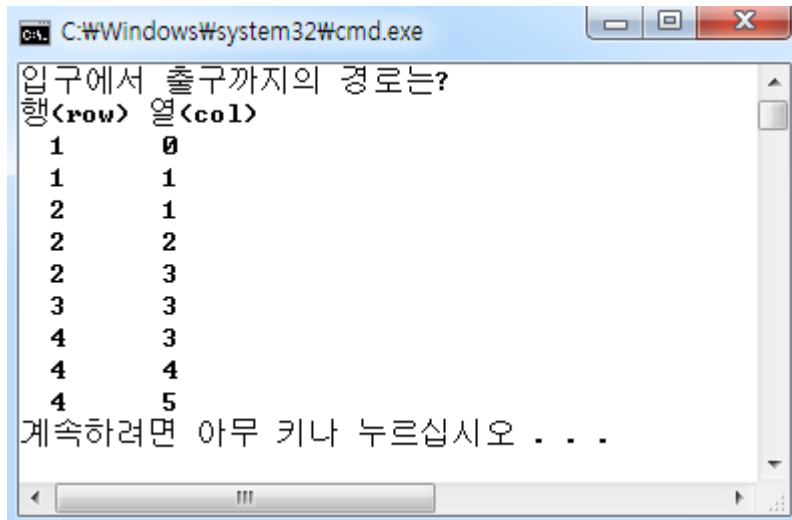
▶ $2\ 3\ 6\ 2\ /\ +\ *\ 2\ +\ 4\ /\ 3\ +$

스택에서 변화되는 값

토 큰	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						

연습문제 3

- 미로 탐색 프로그램에서 탐색 성공 시에 입구로부터 출구까지의 경로가 출력되도록 프로그램을 수정해 보시오.



```
C:\Windows\system32\cmd.exe
입구에서 출구까지의 경로는?
행<row> 열<col>
1      0
1      1
2      1
2      2
2      3
3      3
4      3
4      4
4      5
계속하려면 아무 키나 누르십시오 . . .
```