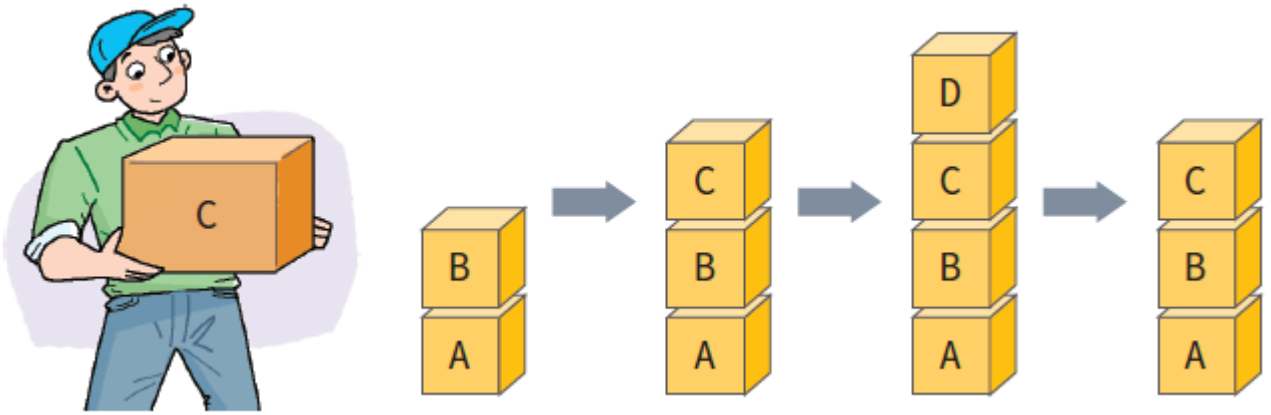
4장 스택

4.1 스택이란?

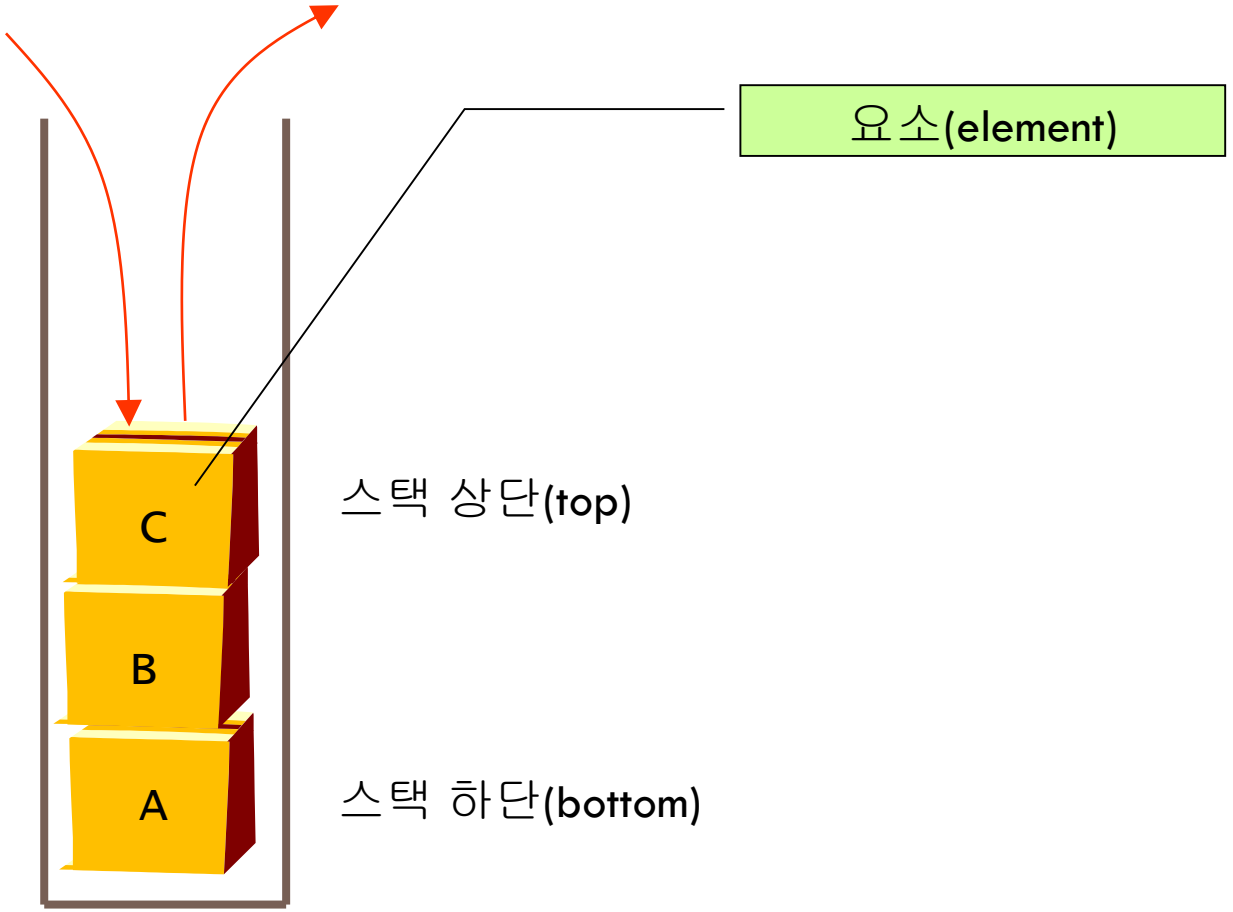
- 스택(stack): 쌓아놓은 더미

스택의 특징

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

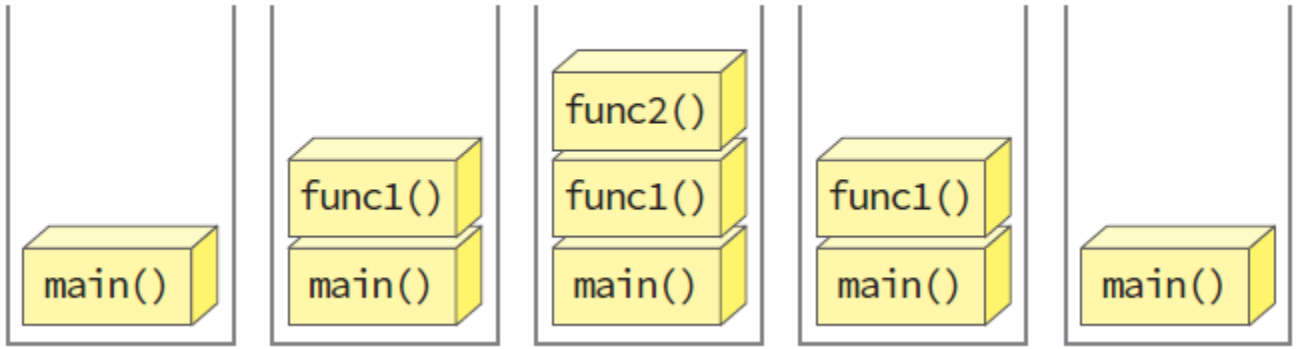


스택의 구조



예제: 시스템 스택을 이용한 함수 호출


```
void func2(){
    return;
}

void func1(){
    func2();
}

int main(void){
    func1();
    return 0;
}
```

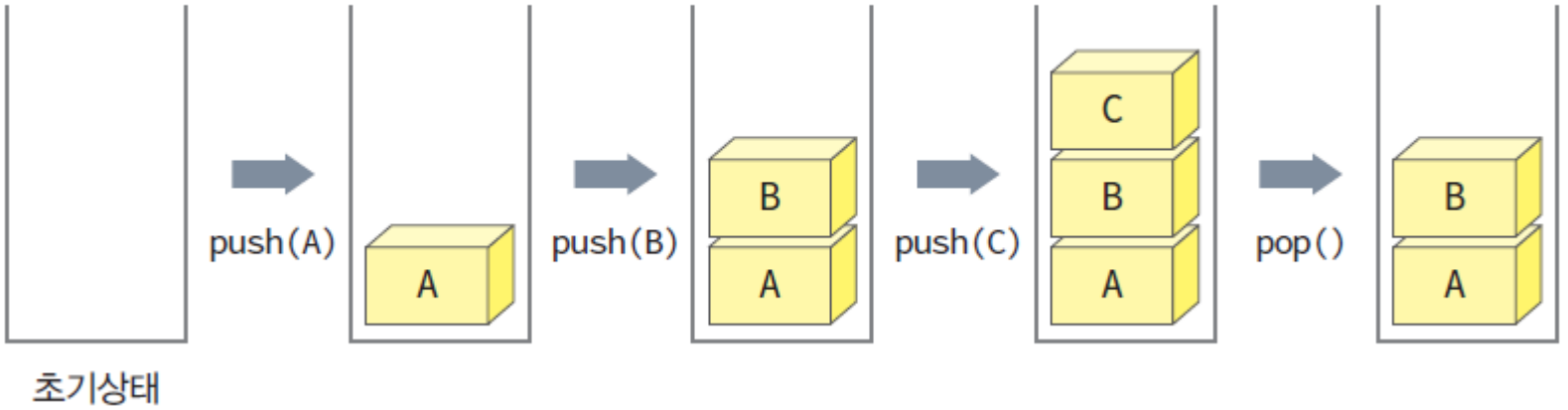


스택 추상데이터타입(ADT)

·객체: 0개 이상의 원소를 가지는 유한 선형 리스트
·연산:
- create(size) ::= 최대 크기가 size인 공백 스택을 생성한다.
- is_full(s) ::=
  if(스택의 원소수 == size) return TRUE;
  else return FALSE;
- is_empty(s) ::=
  if(스택의 원소수 == 0) return TRUE;
  else return FALSE;
- push(s, item) ::=
  if(is_full(s)) return ERROR_STACKFULL;
  else 스택의 맨 위에 item을 추가한다.
- pop(s) ::=
  if(is_empty(s)) return ERROR_STACKEMPTY;
  else 스택의 맨 위의 원소를 제거해서 반환한다.
- peek(s) ::=
  if(is_empty(s)) return ERROR_STACKEMPTY;
  else 스택의 맨 위의 원소를 제거하지 않고 반환한다.

스택의 연산 (1/2)

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

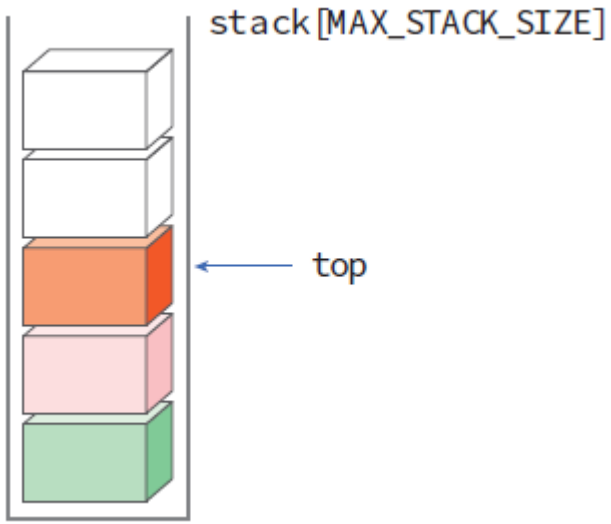


스택의 연산 (2/2)

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

4.2 스택의 구현

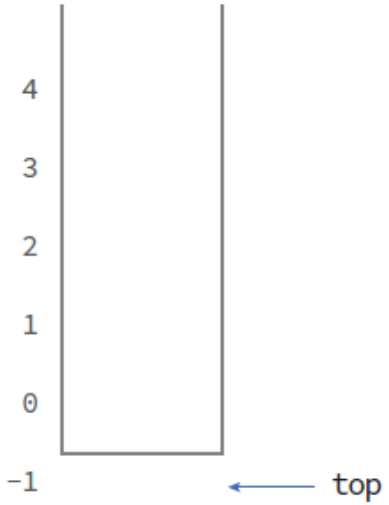
- 배열을 이용한 스택 구현
- 1차원 배열 stack[ ]
- 스택에서 가장 최근에 입력되었던 자료를 가리키는 top 변수
- 가장 먼저 들어온 요소는 stack[0]에, 가장 최근에 들어온 요소는 stack[top]에 저장
- 스택이 공백상태이면 top은 -1



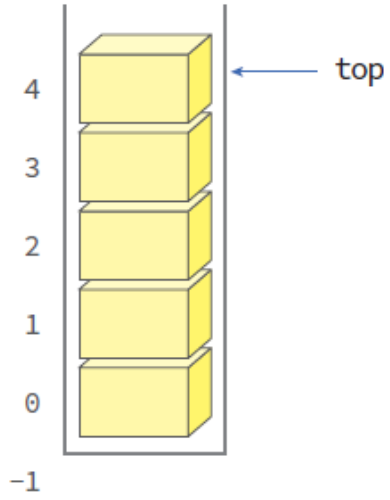
is_empty, is_full 연산


```
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
```



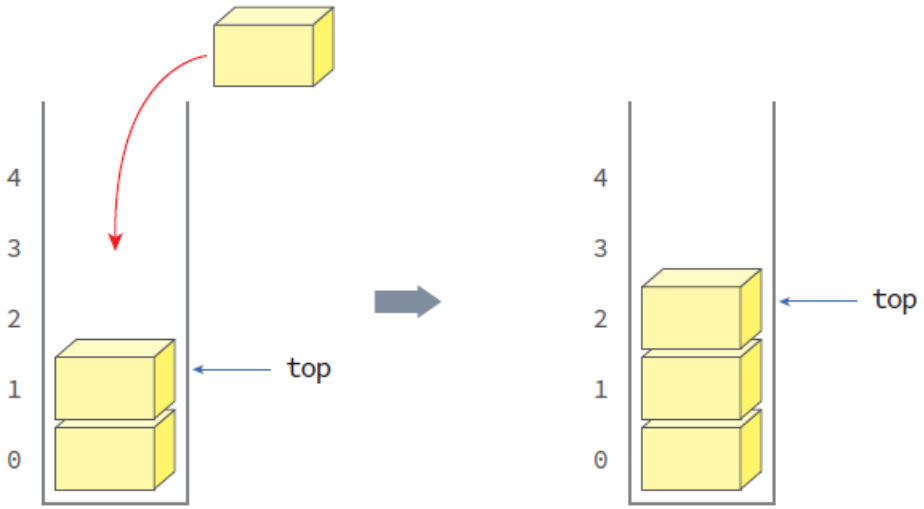
(a) 공백상태



(b) 포화상태

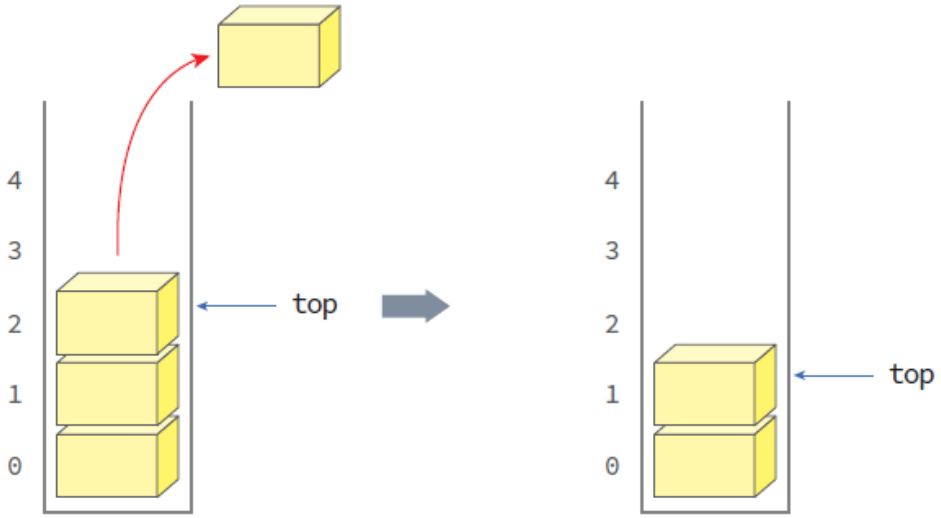
push 연산


```
push(S, x):

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



pop 연산


```
pop(S, x):

if is_empty(S) then
     error "underflow"
else e ← stack[top]
     top ← top-1
     return e
```



전역 변수로 구현: stack1.c (1/2)


```c
#include <stdio.h>
#include <stdlib.h>

#define MAX_STACK_SIZE 100  // 스택의 최대 크기
typedef int element;        // 데이터의 자료형
element  stack[MAX_STACK_SIZE]; // 1차원 배열
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));
}

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

전역 변수로 구현: stack1.c (2/2)


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

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

int main(void)
{
    push(1);
    push(2);
    push(3);
    printf("%d\n", pop());
    printf("%d\n", pop());
    printf("%d\n", pop());
    return 0;
}
```

```
3
2
1

--------------------------------
Process exited after 0.01792 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```

구조체 배열 사용: stack3.c (1/3)


```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_STACK_SIZE 100

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

// 스택 초기화 함수
void init_stack(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));
}
```

구조체 배열 사용: stack3.c (2/3)


```
28  void push(StackType *s, element item)
29  {
30      if (is_full(s)) {
31          fprintf(stderr, "스 택  포 화  에 러 \n");
32          return;
33      }
34      else s->data[++(s->top)] = item;
35  }
36  // 삭 제 함 수
37  element pop(StackType *s)
38  {
39      if (is_empty(s)) {
40          fprintf(stderr, "스 택  공 백  에 러 \n");
41          exit(1);
42      }
43      else return s->data[(s->top)--];
44  }
45  // 피 크 함 수
46  element peek(StackType *s)
47  {
48      if (is_empty(s)) {
49          fprintf(stderr, "스 택  공 백  에 러 \n");
50          exit(1);
51      }
52      else return s->data[s->top];
53  }
```

구조체 배열 사용: stack3.c (2/3)


```
55  int main(void)
56  {
57      StackType s;
58
59      init_stack(&s);
60      push(&s, 1);
61      push(&s, 2);
62      push(&s, 3);
63      printf("%d\n", pop(&s));
64      printf("%d\n", pop(&s));
65      printf("%d\n", pop(&s));
66  }
```

```
C:₩Users₩wjlee₩Documents₩
3
2
1

--------------------------------
Process exited after 0.0171 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```

동적 스택: stack4.c


```
60  int main(void)
61  {
62      StackType *s;
63
64
65      s = (StackType *)malloc(sizeof(StackType));
66      init_stack(s);
67      push(s, 1);
68      push(s, 2);
69      push(s, 3);
70      printf("%d\n", pop(s));
71      printf("%d\n", pop(s));
72
73      printf("%d\n", pop(s));
74      free(s);
75  }
```

변수 선언 대신,
malloc() 함수 호출로
동적 할당

4.3 동적 배열 스택: stack5.c (1/3)

- malloc()을 호출하여서 실행 시간에 메모리를 할당 받아서 스택을 생성한다.
- 예제에서는 초기에는 100개의 스택 공간을 확보하지만, 필요 시에는 2배씩 스택 공간을 동적으로 확보한다(push 함수).

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef int element;
5  typedef struct {
6      element *data;      // data은 포인터로 정의된다.
7      int capacity;       // 현재 크기
8      int top;
9  } StackType;
```

동적 배열 스택: stack5.c (2/3)


```
11  // 스택 생성 함수
12  void init_stack(StackType *s)
13  {
14      s->top = -1;
15      s->capacity = 100;
16      s->data = (element *)malloc(s->capacity * sizeof(element));
17  }
18
19  // 공백 상태 검출 함수
20  int is_empty(StackType *s)
21  {
22      return (s->top == -1);
23  }
24
25  // 포화 상태 검출 함수
26  int is_full(StackType *s)
27  {
28      return (s->top == (s->capacity - 1));
29  }
30
31  void push(StackType *s, element item)
32  {
33      if (is_full(s)) {
34          s->capacity *= 2;
35          s->data =
36              (element *)realloc(s->data, s->capacity * sizeof(element));
37      }
38      s->data[++(s->top)] = item;
39  }
```

동적 배열 스택: stack5.c (3/3)


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

int main(void)
{
    StackType s;
    init_stack(&s);
    push(&s, 1);
    push(&s, 2);
    push(&s, 3);
    printf("%d \n", pop(&s));
    printf("%d \n", pop(&s));
    printf("%d \n", pop(&s));
    free(s.data);
    return 0;
}
```

```
3
2
1

--------------------------------
Process exited after 0.0171 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```

4.4 스택의 응용: 괄호 검사

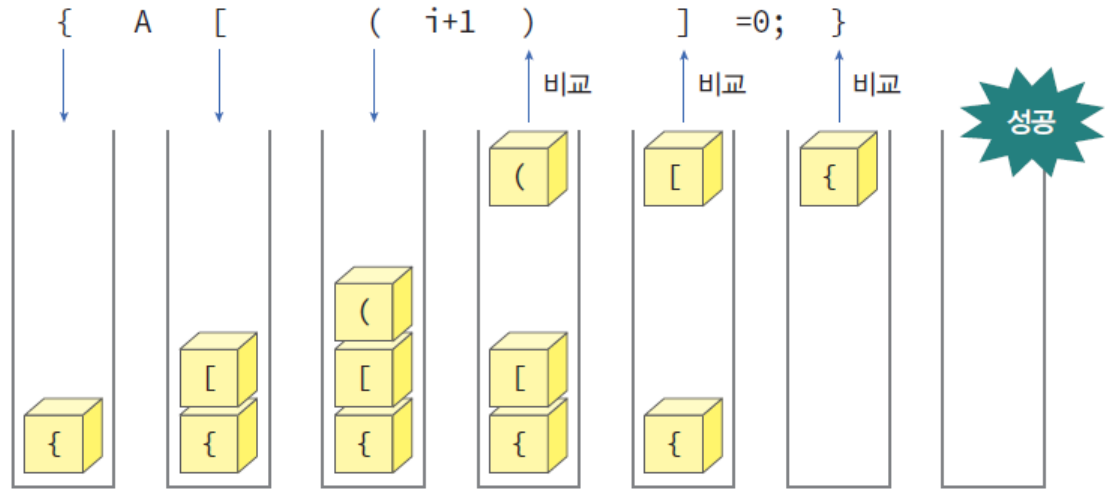
- 괄호의 종류: 대괄호 (‘[’, ‘]’), 중괄호 (‘{’, ‘}’), 소괄호 (‘(’, ‘)’)
- 조건
  1. 왼쪽 괄호의 개수와 오른쪽 괄호의 개수가 같아야 한다.
  2. 같은 괄호에서 왼쪽 괄호는 오른쪽 괄호보다 먼저 나와야 한다.
  3. 괄호 사이에는 포함 관계만 존재한다. (다른 종류 괄호의 교차X)
- 잘못된 괄호 사용의 예

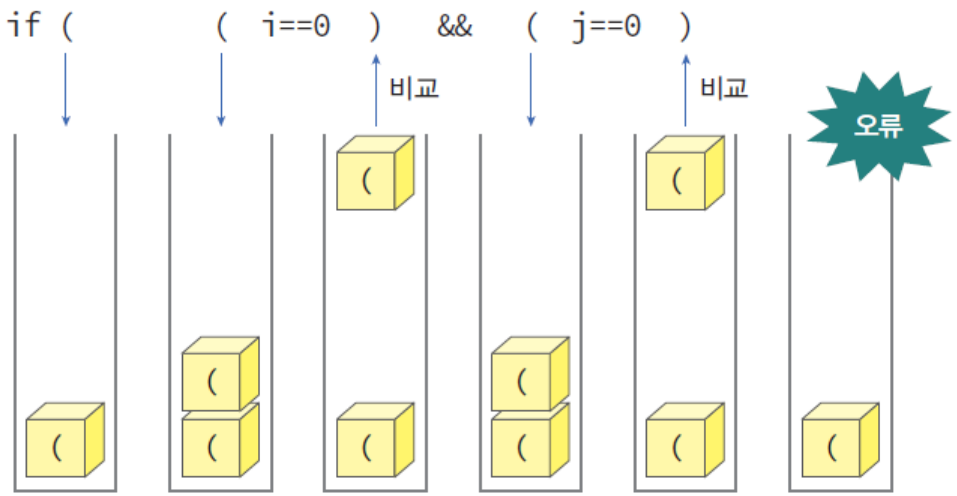
  (a(b)

  a(b)c)

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

스택을 이용한 괄호 검사





알고리즘

알고리즘의 개요

- 문자열에 있는 괄호를 차례대로 조사하면서 왼쪽 괄호를 만나면 스택에 삽입하고, 오른쪽 괄호를 만나면 스택에서 top 괄호를 삭제한 후 오른쪽 괄호와 짝이 맞는지를 검사한다.
- 이 때, 스택이 비어 있으면 조건 1 또는 조건 2 등을 위배하게 되고 괄호의 짝이 맞지 않으면 조건 3 등에 위배된다.
- 마지막 괄호까지를 조사한 후에도 스택에 괄호가 남아 있으면 조건 1에 위배되므로 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: paren_matching.cpp (1/3)


```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_STACK_SIZE 100

typedef char element;

// ===== 스택 코드의 시작 =====
#define MAX_STACK_SIZE 100

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

// 스택 초기화 함수
void init_stack(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));
}
```

괄호 검사 프로그램: paren_matching.cpp (2/3)


```
34  // 삽입 함수
35  void push(StackType *s, element item)
36  {
37      if (is_full(s)) {
38          fprintf(stderr, "스택 포화 에러\n");
39          return;
40      }
41      else s->data[++(s->top)] = item;
42  }
43
44  // 삭제 함수:
45  element pop(StackType *s)
46  {
47      if (is_empty(s)) {
48          fprintf(stderr, "스택 공백 에러\n");
49          exit(1);
50      }
51      else return s->data[(s->top)--];
52  }
53
54  // 피크 함수:
55  element peek(StackType *s)
56  {
57      if (is_empty(s)) {
58          fprintf(stderr, "스택 공백 에러\n");
59          exit(1);
60      }
61      else return s->data[s->top];
62  }
63  // ===== 스택 코드의 끝 =====
```

괄호 검사 프로그램: paren_matching.cpp (3/3)


```
65  // 에러 발생 -> 0 리턴
66  // 에러 없음 -> 1 리턴
67  int check_matching(const char *in)
68  {
69      StackType s;
70      char ch, open_ch;
71      int i, n = strlen(in);       // n= 문자열의 길이
72      init_stack(&s);              // 스택의 초기화
73
74      for (i = 0; i < n; i++)
75      {
76          ch = in[i];      // ch = 다음 문자
77          switch (ch)
78          {
79          case '(':   case '[':    case '{':
80              push(&s, ch);
81              break;
82          case ')':   case ']':    case '}':
83              if (is_empty(&s))  return 0;
84              else {
85                  open_ch = pop(&s);
86                  if ((open_ch == '(' && ch != ')') ||
87                      (open_ch == '[' && ch != ']') ||
88                      (open_ch == '{' && ch != '}')) {
89                      return 0;
90                  }
91                  break;
92              }
93          }
94      }
95
96      if (!is_empty(&s)) return 0; // 스택에 남아있으면 오류
97      return 1;
98  }
```

```
100 int main(void)
101 {
102     char *p = "{ A[(i+1)]=0; }";
103     if (check_matching(p) == 1)
104         printf("%s 괄호검사성공 \n", p);
105     else
106         printf("%s 괄호검사실패 \n", p);
107     return 0;
108 }
```

4.5 스택의 응용: 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	+*ab5	ab*5+
(1+2)*7	*+127	12+7 *

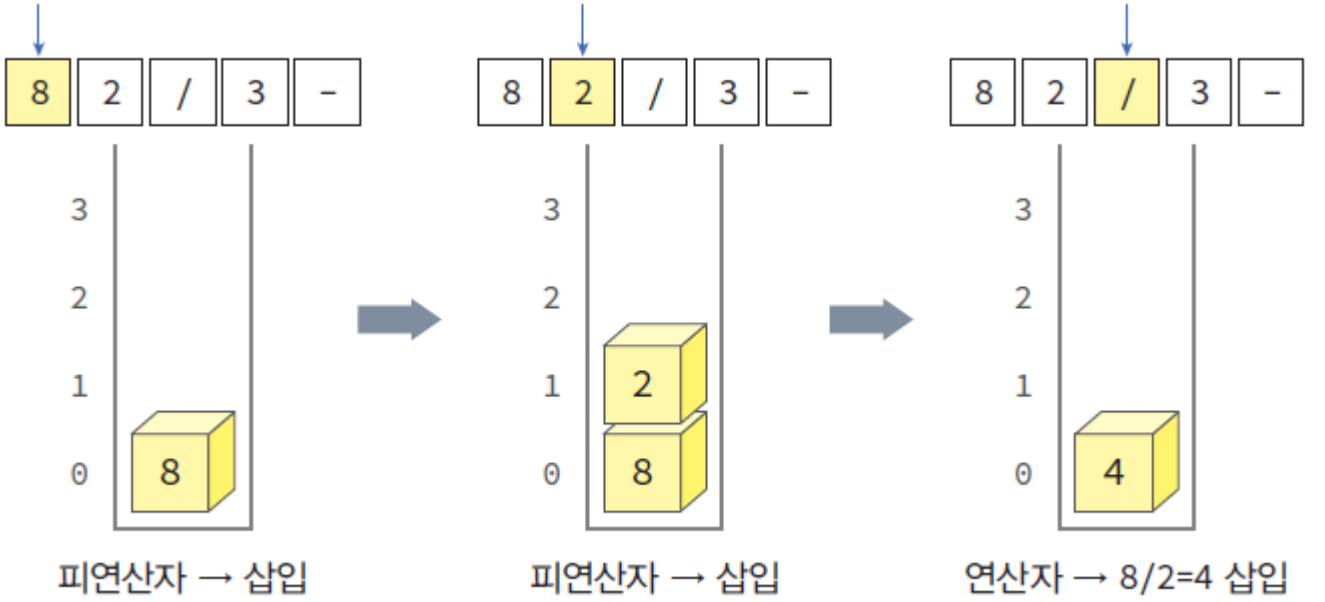
- 컴퓨터에서의 수식 계산 순서
  - 중위표기식 -> 후위표기식 -> 계산 예) 2+3*4 -> 234*+ -> 14
- 후위표기법의 장점: 괄호가 필요 없음

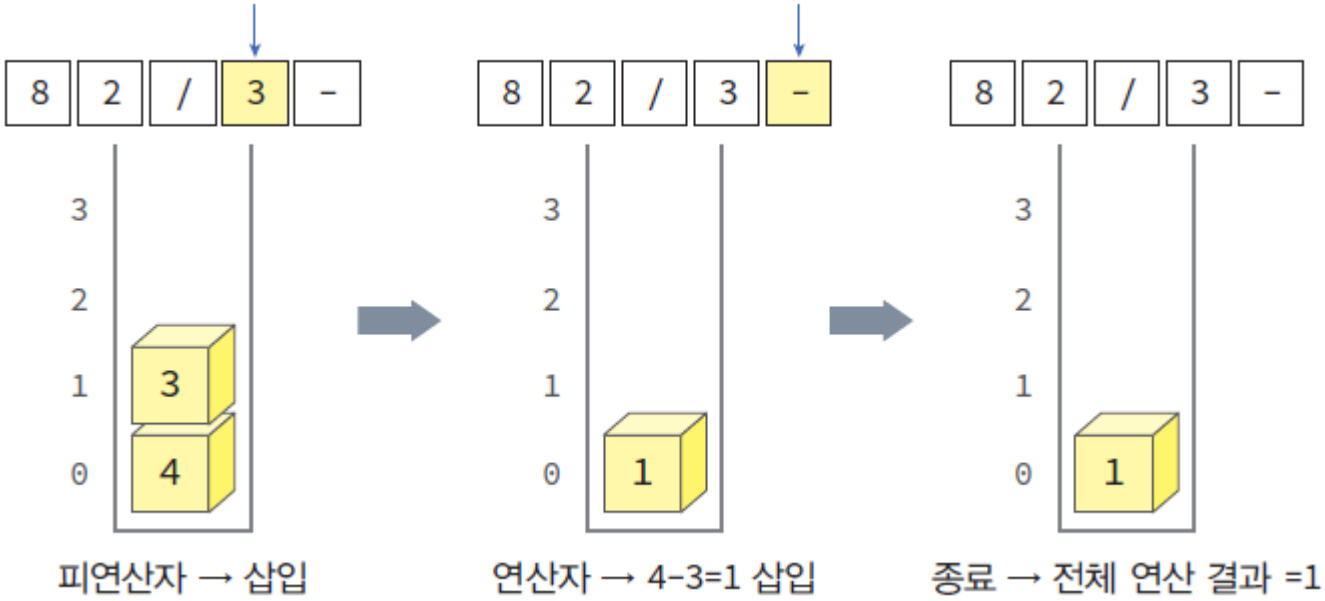
- 중위표기법 –> 후위/전위표기법 변환
  ① 중위표기식의 모든 식에 대해 괄호를 표기
  ② 이항 연산자들을 모두 오른쪽(왼쪽) 괄호 위치로 이동
  ③ 괄호를 모두 삭제한다.

후위 표기식의 계산

- 수식을 왼쪽에서 오른쪽으로 스캔하여 피연산자이면 스택에 저장하고 연산자이면 필요한 수만큼의 피연산자를 스택에서 꺼내 연산을 실행하고 연산의 결과를 다시 스택에 저장
- (예) 82/3-32*+

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						


8 2 / 3 -
3
2
1
0
8
2
8
4
피연산자 → 삽입
피연산자 → 삽입
연산자 → 8/2=4 삽입


8 2 / 3 -
3
2
1
0
3
4
1
1
피연산자 → 삽입
연산자 → 4-3=1 삽입
종료 → 전체 연산 결과 =1

후위 표기식 계산 알고리즘

cal_postfix():

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

후위 표기식 계산: postfix.c (1/2)


```
62  // 후위 표기 수식 계산 함수:
63  int eval(char exp[])
64  {
65      int op1, op2, value, i = 0;
66      int len = strlen(exp);
67      char ch;
68      StackType s;
69
70      init_stack(&s);
71      for (i = 0; i<len; i++)
72      {
73          ch = exp[i];
74          if (ch != '+' && ch != '-' && ch != '*' && ch != '/')
75          {
76              value = ch - '0';    // 입력이 피연산자이면
77              push(&s, value);
78          }
79          else
80          {   //연산자이면 피연산자를 스택에서 제거
81              op2 = pop(&s);
82              op1 = pop(&s);
83              switch (ch)
84              { //연산을 수행하고 스택에 저장
85              case '+': push(&s, op1 + op2); break;
86              case '-': push(&s, op1 - op2); break;
87              case '*': push(&s, op1 * op2); break;
88              case '/': push(&s, op1 / op2); break;
89              }
90          }
91      }
92      return pop(&s);
93  }
```

후위 표기식 계산: postfix.c (2/2)


```
 95  int main(void)
 96  {
 97      int result;
 98      printf("후 위 표 기 식 은  82/3-32*+\n");
 99      result = eval("82/3-32*+");
100      printf("결 과 값 은  %d\n", result);
101      return 0;
102  }
```

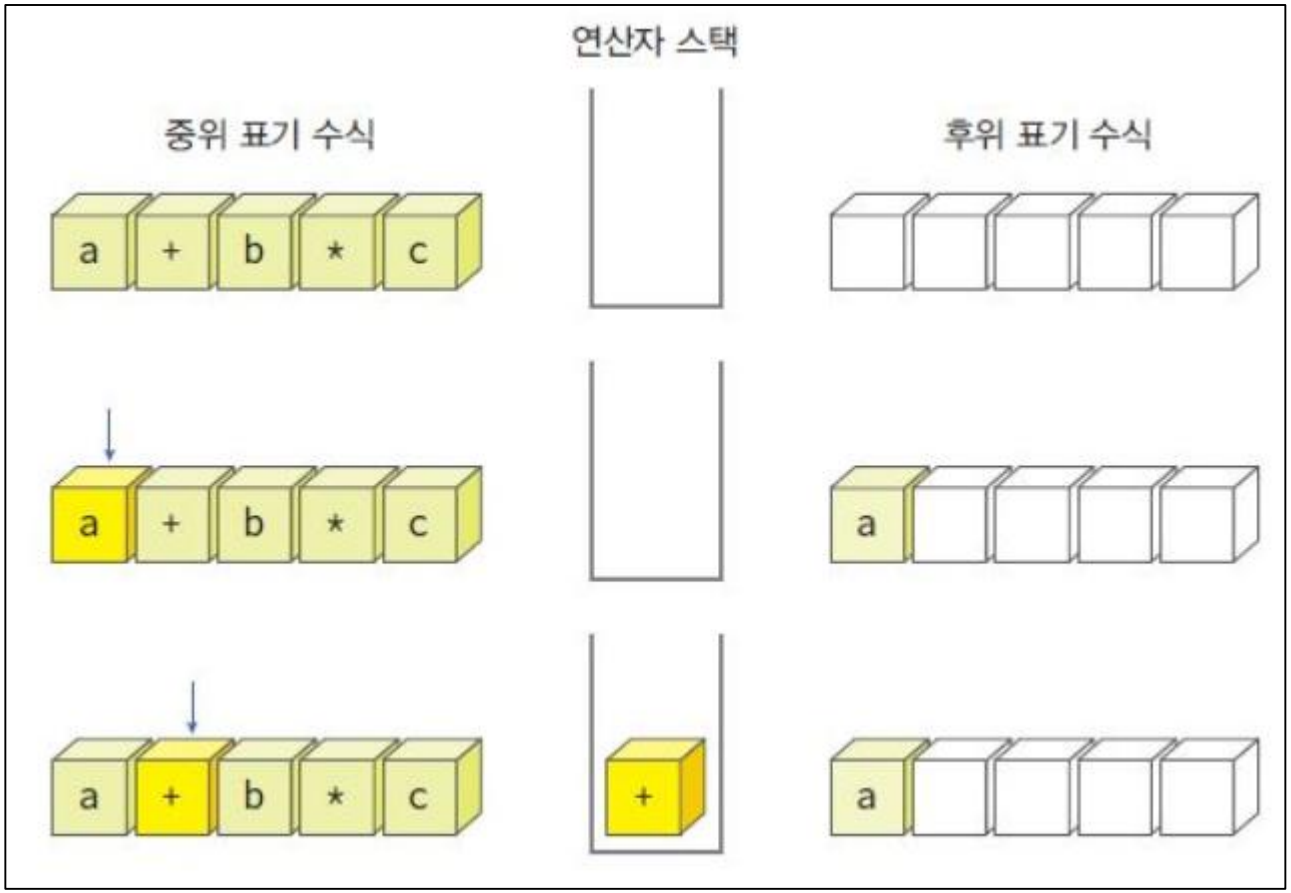
```
후위표기식은 82/3-32*+
결과값은 7

--------------------------------
Process exited after 0.3504 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```

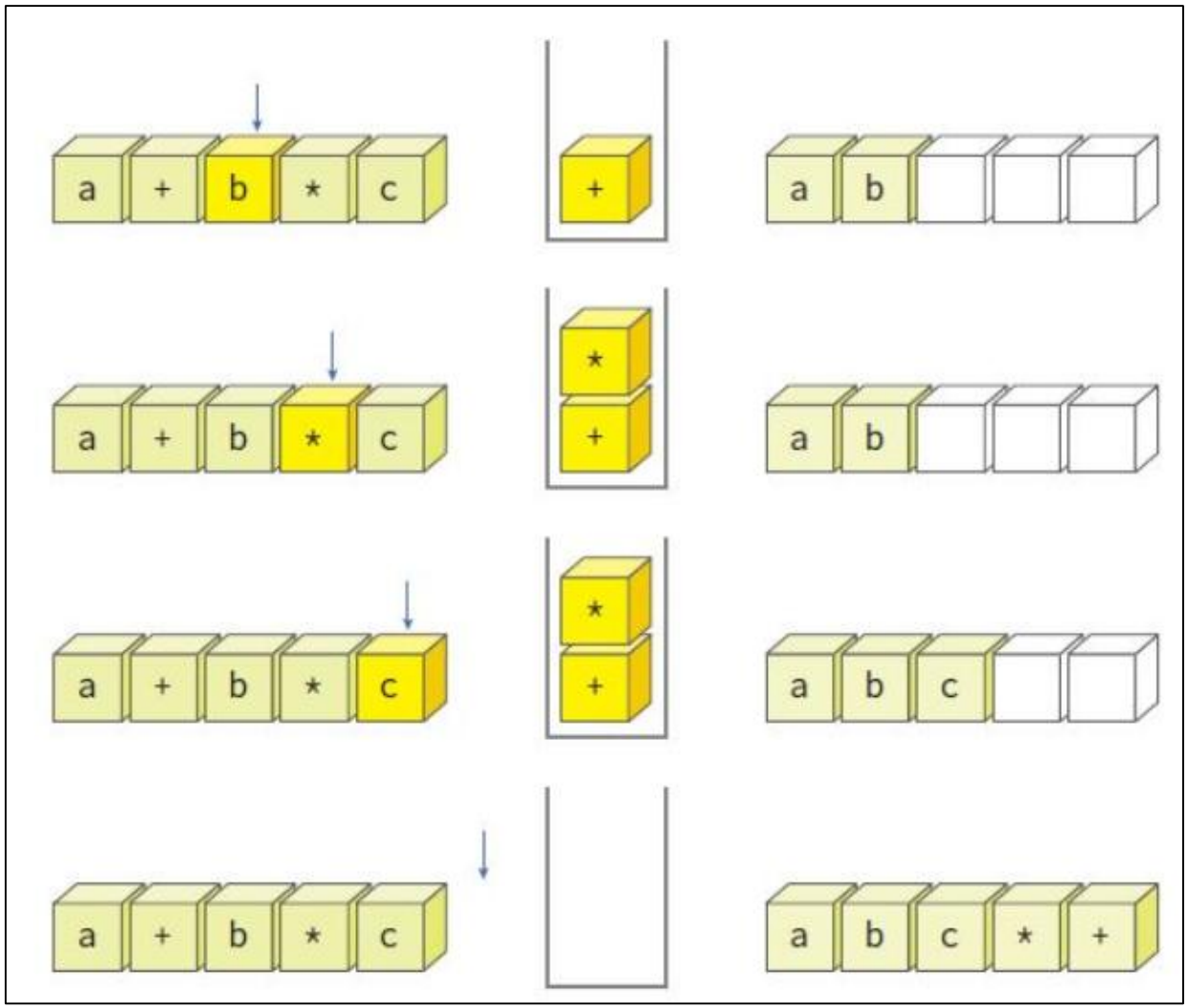
중위표기식 -> 후위표기식

- 중위표기와 후위표기
  - 중위 표기법과 후위 표기법의 공통점은 피연산자의 순서는 동일
  - 연산자들의 순서만 다름(우선순위순서)
    - 연산자만 스택에 저장했다가 출력하면 된다.
    - 2+3*4 -> 234*+
  - 후위표기식은 연산자 우선 순위를 위한 '(' ')' 가 필요 없음

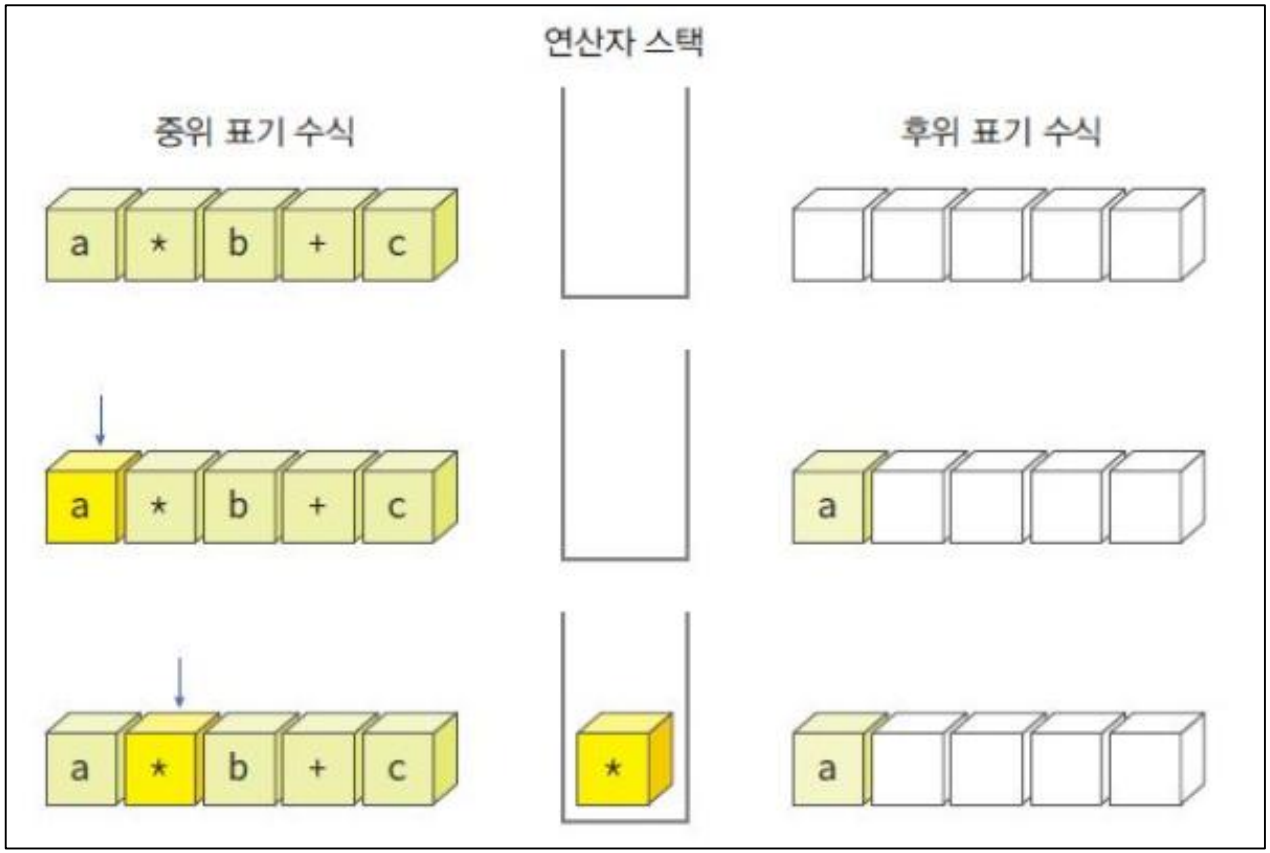
중위 표기법	후위 표기법
a+b	ab+
(a+b)*c	ab+c*
a+b*c	abc*+

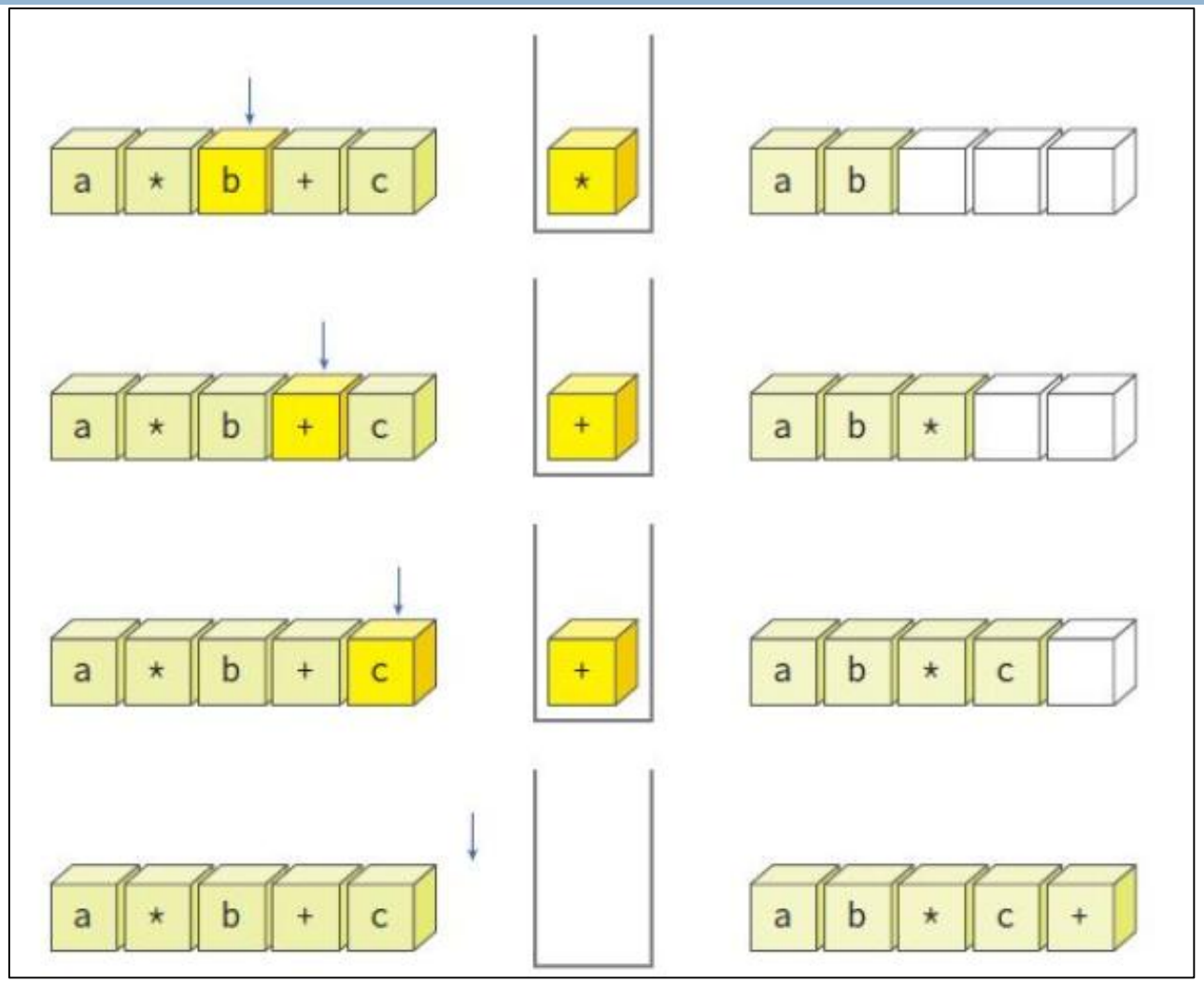
예 1: a + b * c



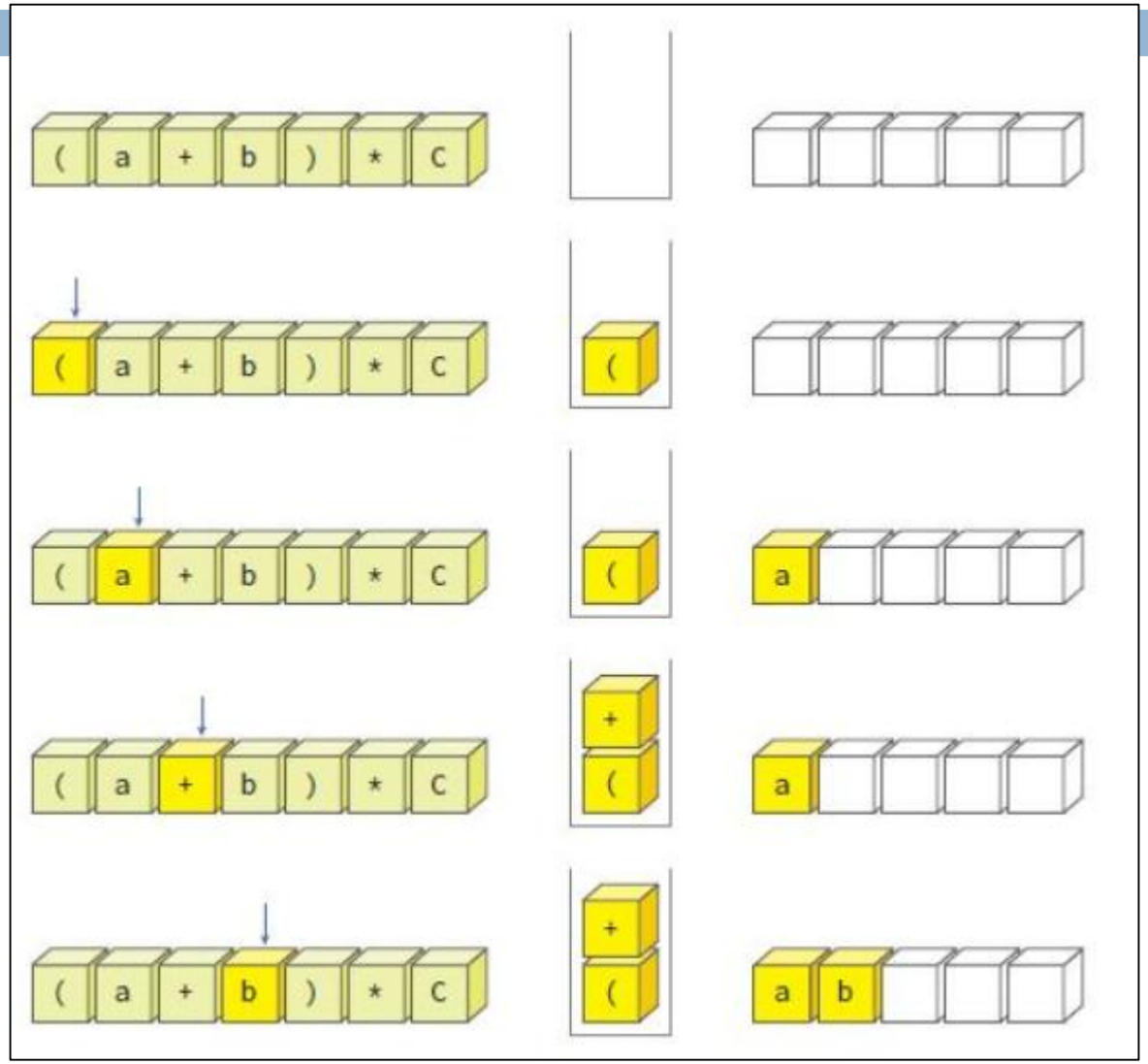

a + b * c
+
a b
a + b * c
*
+
a b
a + b * c
*
+
a b c
a + b * c
a b c * +

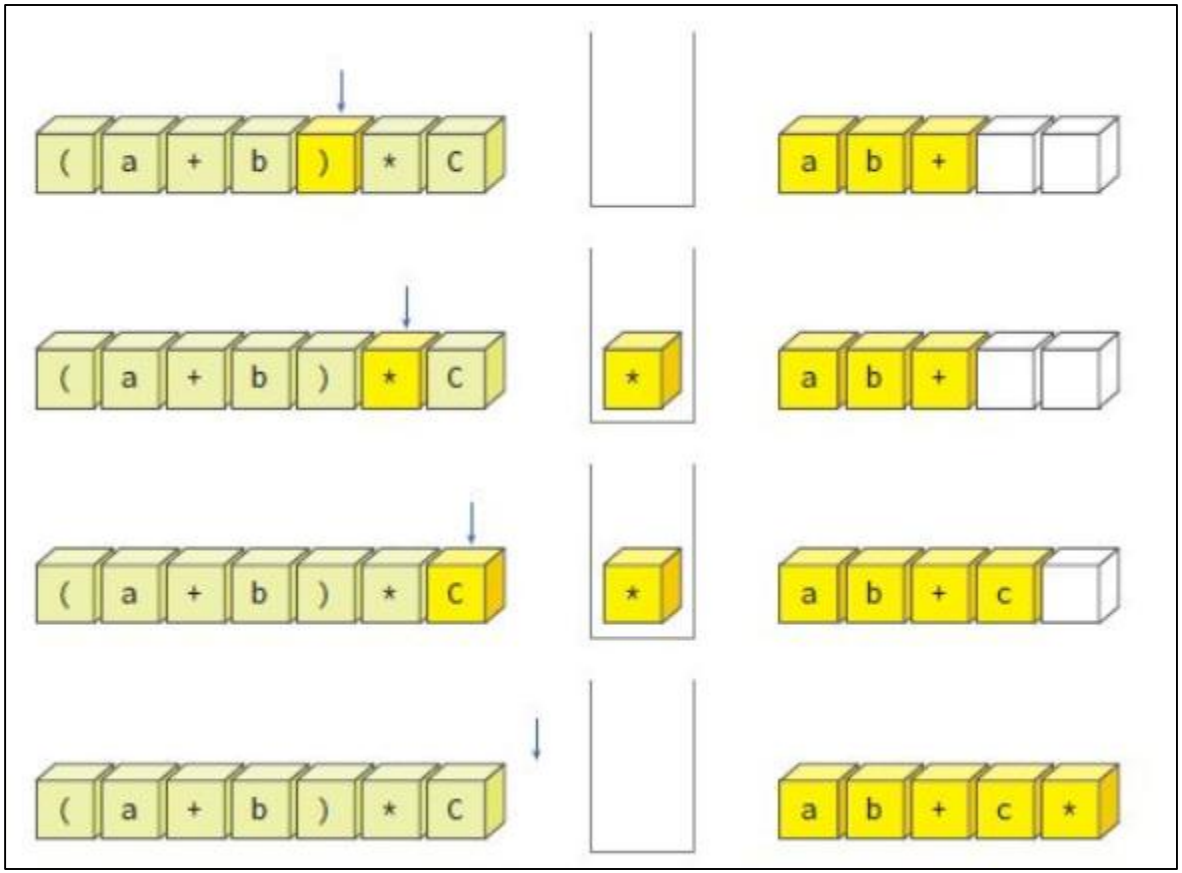
예 2: a * b + c




a * b + c
*
a b
a * b + c
+
a b *
a * b + c
+
a b * c
a * b + c
a b * c +

예 3: (a + b) * c




(a + b) * C
a b +
(a + b) * C
*
a b +
(a + b) * C
*
a b + c
(a + b) * C
a b + c *

전위표기식 -> 후위표기식 알고리즘 (1/2)

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;

전위표기식 -> 후위표기식 알고리즘 (2/2)


```
      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
```

전위표기식 -> 후위표기식 프로그램(infix.c) (1/2)


```
74  void infix_to_postfix(char exp[])
75  {
76      int i = 0;
77      char ch, top_op;
78      int len = strlen(exp);
79      StackType s;
80
81      init_stack(&s);                  // 스택 초기화
82      for (i = 0; i<len; i++) {
83          ch = exp[i];
84          switch (ch) {
85          case '+': case '-': case '*': case '/': // 연산자
86                                                  // 스택에 있는 연산자의 우선순위가 더 크거나 같으면 출력
87              while (!is_empty(&s) && (prec(ch) <= prec(peek(&s))))
88                  printf("%c", pop(&s));
89              push(&s, ch);
90              break;
91          case '(':   // 왼쪽 괄호
92              push(&s, ch);
93              break;
94          case ')':   // 오른쪽 괄호
95              top_op = pop(&s);
96              // 왼쪽 괄호를 만날 때까지 출력
97              while (top_op != '(') {
98                  printf("%c", top_op);
99                  top_op = pop(&s);
00              }
01              break;
02          default:        // 피연산자
03              printf("%c", ch);
04              break;
05          }
06      }
07      while (!is_empty(&s))   // 스택에 저장된 연산자들 출력
08          printf("%c", pop(&s));
09  }
```

전위표기식 -> 후위표기식 프로그램(infix.c) (2/2)

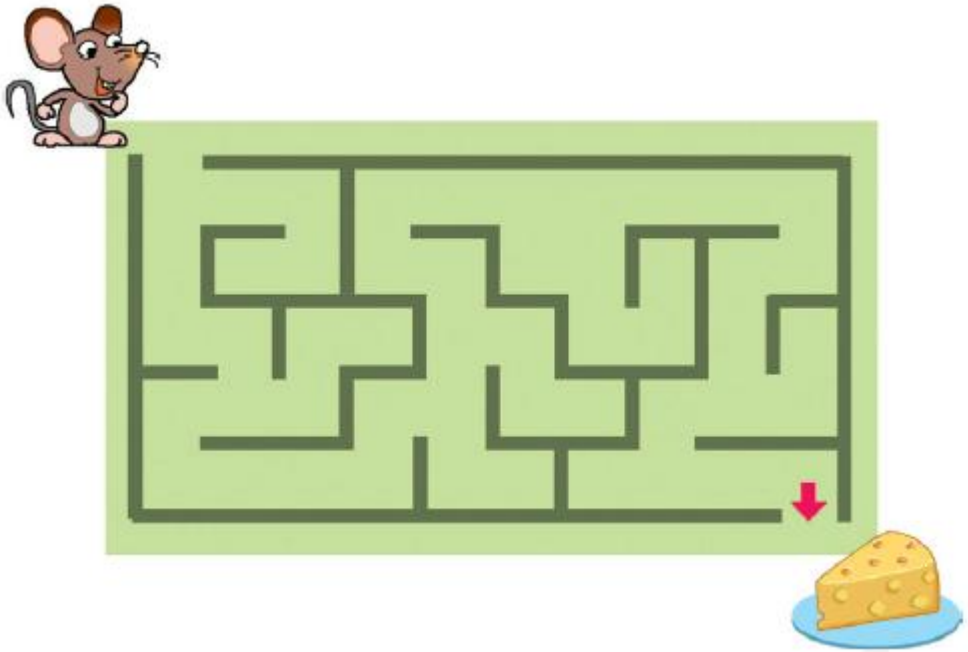

```
63  int prec(char op)
64  {
65      switch (op) {
66      case '(': case ')': return 0;
67      case '+': case '-': return 1;
68      case '*': case '/': return 2;
69      }
70      return -1;
71  }
```

```
113 int main(void)
114 {
115     char *s = "(2+3)*4+9";
116     printf("중위표시수식 %s \n", s);
117     printf("후위표시수식 ");
118     infix_to_postfix(s);
119     printf("\n");
120     return 0;
121 }
```

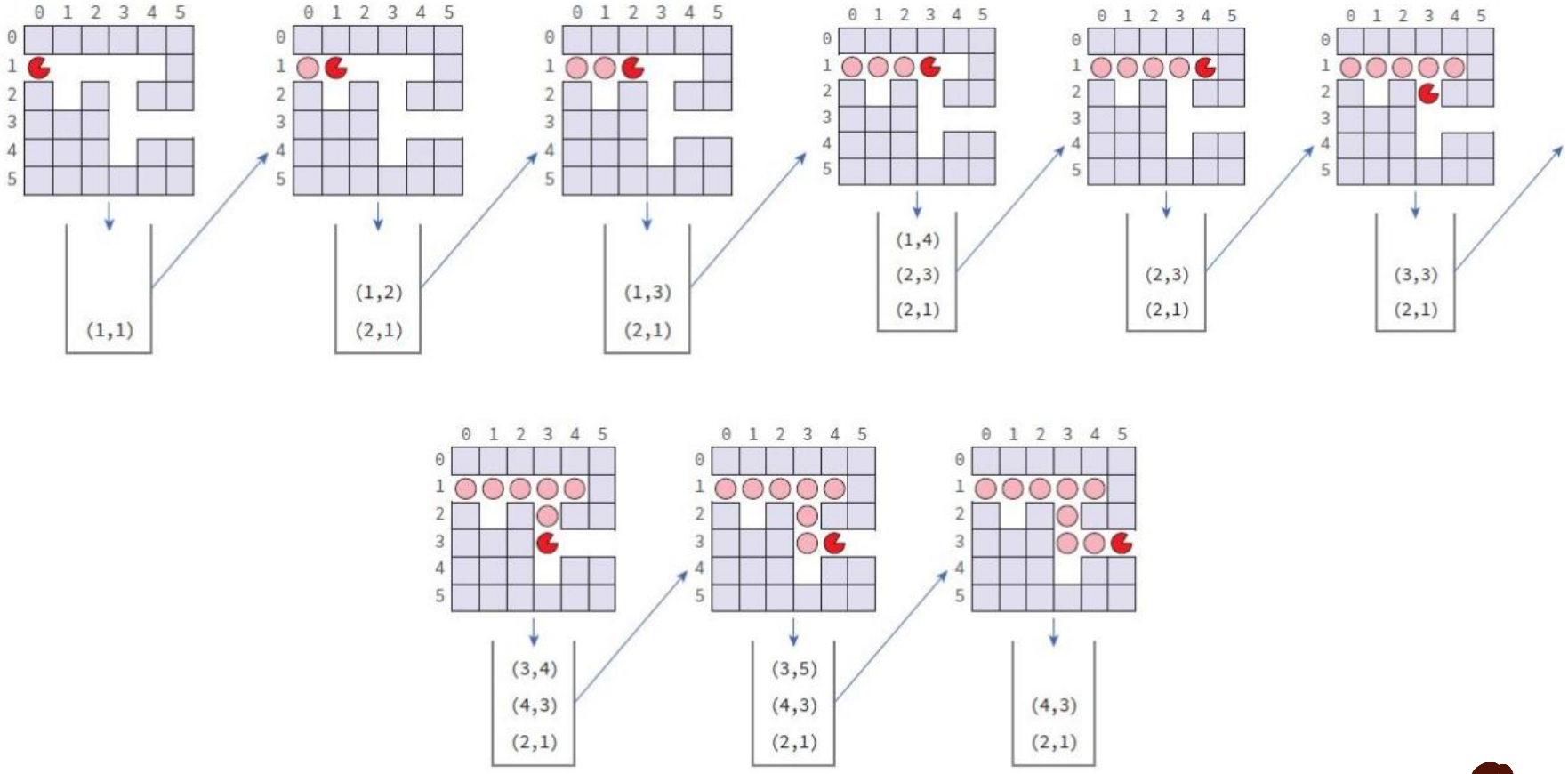
```
중위표시수식 (2+3)*4+9
후위표시수식 23+4*9+

--------------------------------
```

4.6 스택의 응용: 미로 문제

- 가능한 모든 경로로 시도하여 출구를 탐색함
- 현재의 위치에서 가능한 방향을 스택에 저장해 놓았다가, 막다른 길을 만나면 스택에서 다음 탐색 위치를 꺼낸다.



미로 탐색 알고리즘 적용 예



미로탐색 알고리즘

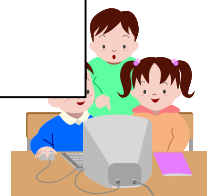
maze_search():

스택 s과 출구의 위치 x, 현재 생쥐의 위치를 초기화
while(현재의 위치가 출구가 아니면) do
    현재위치를 방문한 것으로 표기
    if(현재 위치의 위, 아래, 왼쪽, 오른쪽 위치가 아직 방문 되지 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: maze.c (1/3)


```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_STACK_SIZE 100
#define MAZE_SIZE 6

typedef struct {        // 교체!
    short r;
    short c;
} element;

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

```c
element 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: maze.c (2/3)


```
73  // 위치를 스택에 삽입
74  void push_loc(StackType *s, int r, int c)
75  {
76      if (r < 0 || c < 0) return;
77      if (maze[r][c] != '1' && maze[r][c] != '.') {
78          element tmp;
79          tmp.r = r;
80          tmp.c = c;
81          push(s, tmp);
82      }
83  }
84
85  // 미로를 화면에 출력한다.
86  void maze_print(char maze[MAZE_SIZE][MAZE_SIZE])
87  {
88      printf("\n");
89      for (int r = 0; r < MAZE_SIZE; r++) {
90          for (int c = 0; c < MAZE_SIZE; c++) {
91              printf("%c", maze[r][c]);
92          }
93          printf("\n");
94      }
95  }
```

미로 프로그램: maze.c (3/3)


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

    init_stack(&s);
    here = entry;
    while (maze[here.r][here.c] != 'x') {
        r = here.r;
        c = here.c;
        maze[r][c] = '.';
        maze_print(maze);
        push_loc(&s, r - 1, c);
        push_loc(&s, r + 1, c);
        push_loc(&s, r, c - 1);
        push_loc(&s, r, c + 1);
        if (is_empty(&s)) {
            printf("실 패 \n");
            return 1;
        }
        else
            here = pop(&s);
    }
    printf("성 공 \n");
    return 0;
}
```

```
111111
.01001
100011
101011
10100x
111111

111111
..1001
100011
101011
10100x
111111

111111
..1001
1.0011
101011
10100x
111111

111111
..1001
1..011
101011
10100x
111111
```

```
111111
..1001
1...11
101011
10100x
111111

111111
..1001
1...11
101.11
10100x
111111

111111
..1001
1...11
101.11
101.0x
111111

111111
..1001
1...11
101.11
101..x
111111
성공
```