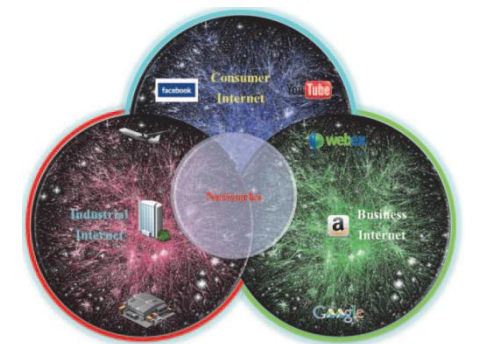




*Prof. Chang Choi*



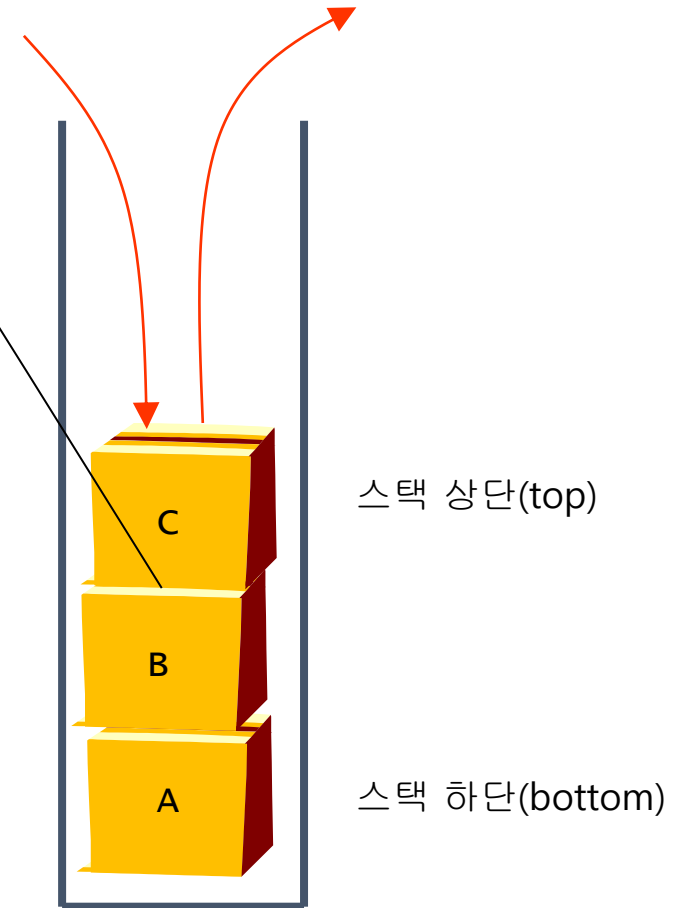
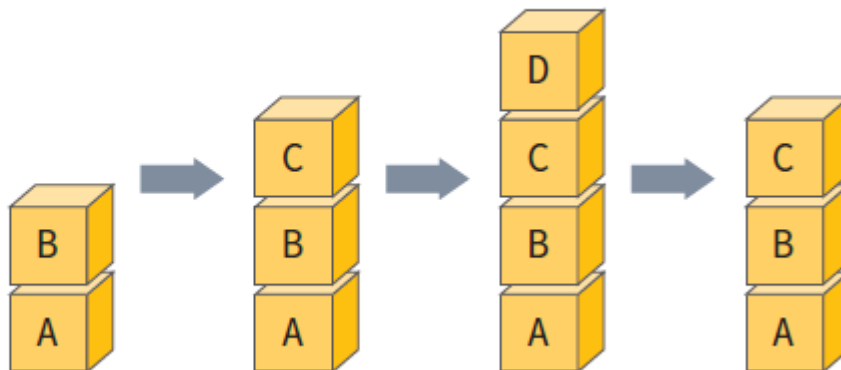
## 스택이란?

- 스택(stack): 쌓아놓은 더미



요소(element)

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

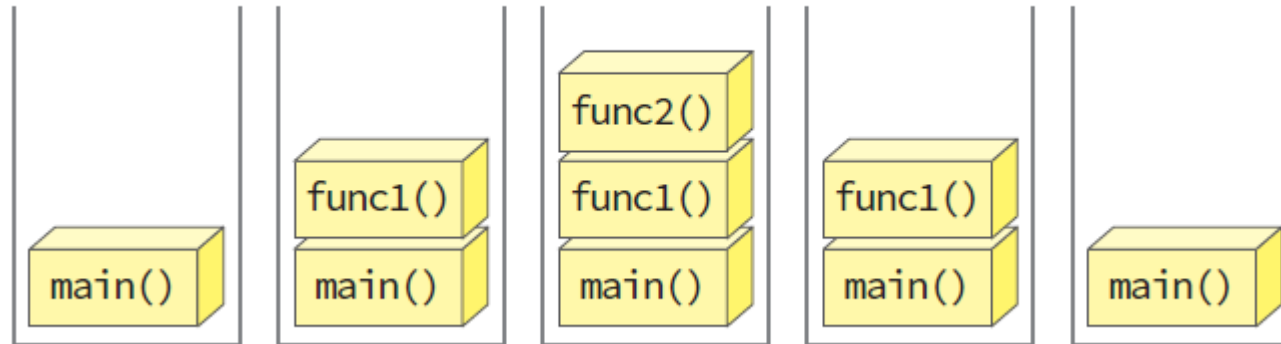


## 예제: 시스템 스택을 이용한 함수 호출

```
void func2(){
    return;
}
```

```
void func1(){
    func2();
}
```

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



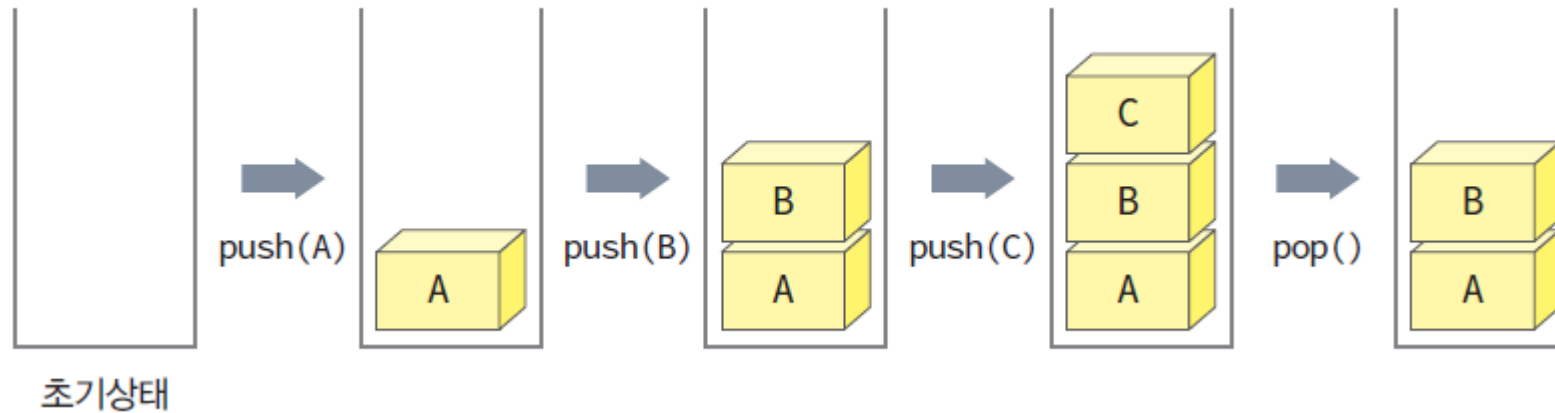
## 스택 추상데이터타입(ADT)

---

- 객체: 0개 이상의 원소를 가지는 유한 선형 리스트
- 연산:
  - `create(size) ::=` 최대 크기가 `size`인 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

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



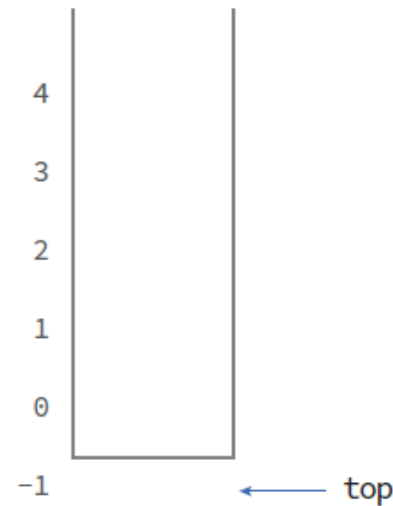
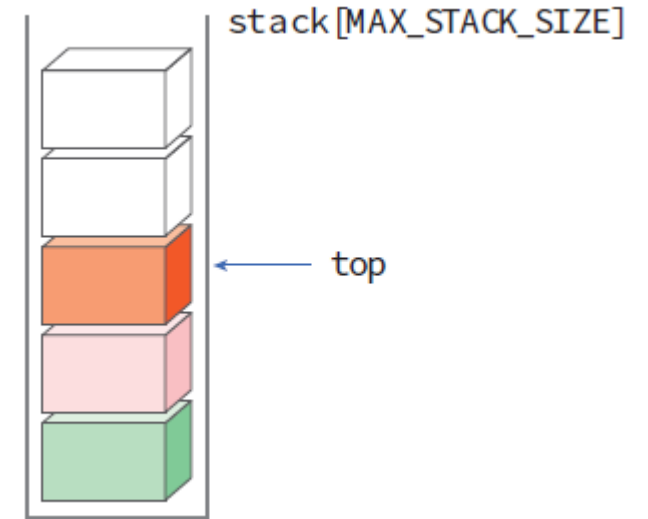
- `is_empty(s)`: 스택이 공백상태인지 검사
- `is_full(s)`: 스택이 포화상태인지 검사
- `create()`: 스택을 생성
- `peek(s)`: 요소를 스택에서 삭제하지 않고 보기만 하는 연산
  - (참고) `pop` 연산은 요소를 스택에서 완전히 삭제하면서 가져온다.

## 배열을 이용한 스택의 구현

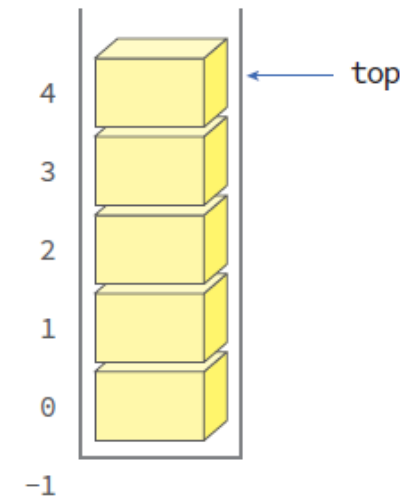
- 1차원 배열 `stack[ ]`
- 스택에서 가장 최근에 입력되었던 자료를 가리키는 `top` 변수
- 가장 먼저 들어온 요소는 `stack[0]`에, 가장 최근에 들어온 요소는 `stack[top]`에 저장
- 스택이 공백상태이면 `top`은 -1

```
is_empty(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

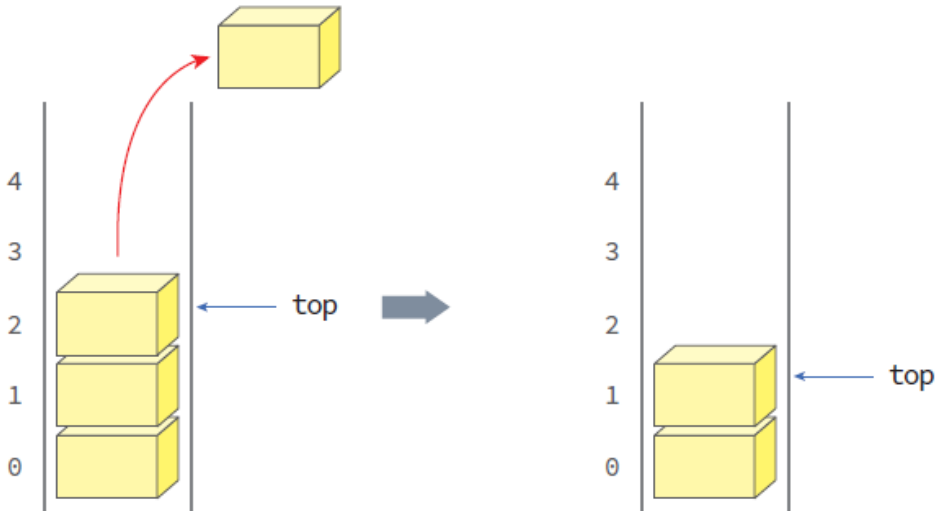
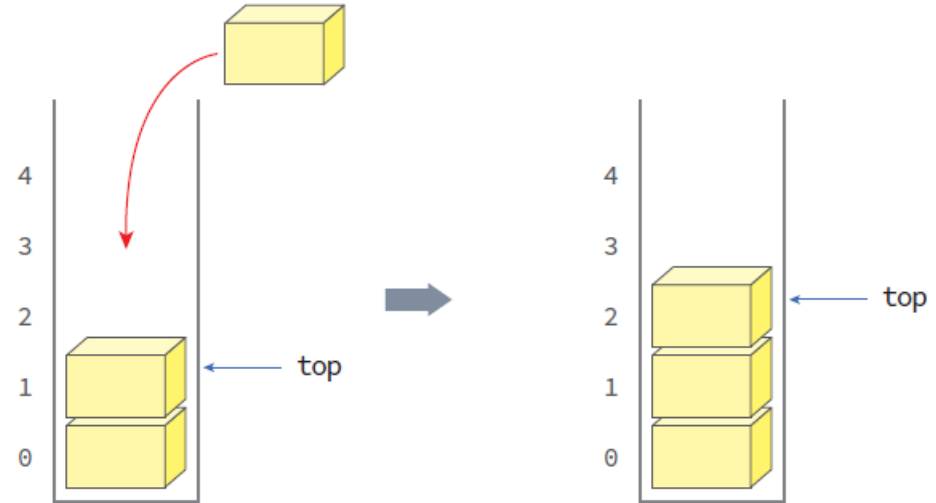
## push & pop 연산

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

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

```
pop(S, x):
```

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



## 전역 변수로 구현하는 방법

```
#define _CRT_SECURE_NO_WARNINGS
#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;
}

// 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);
    push(4);
    printf("%d\n", pop());
    printf("%d\n", pop());
    printf("%d\n", pop());
    return 0;
}
```





## 구조체 배열 사용하기

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

typedef struct {
    int student_no;
    char name[MAX_STRING];
    char address[MAX_STRING];
} 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));
}
```

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

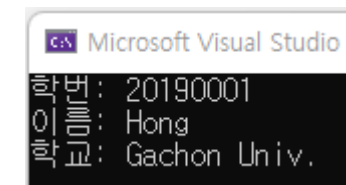
// 삭제 함수
element pop()
{
    if (is_empty()) {
        fprintf(stderr, "스택 공백 에러\n");
        exit(1);
    }
    else return stack[top--];
}

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

```
int main(void)
{
    element ie = { 20190001,
                   "Hong",
                   "Gachon Univ." };
    element oe;

    push(ie);
    oe = pop();

    printf("학번: %d\n", oe.student_no);
    printf("이름: %s\n", oe.name);
    printf("학교: %s\n", oe.address);
    return 0;
}
```



```
Microsoft Visual Studio
학번: 20190001
이름: Hong
학교: Gachon Univ.
```

## 일반적인 배열 스택 프로그램

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

// 차후에 스택이 필요하면 여기만 복사하여 붙인다.
// ===== 스택 코드의 시작 =====
#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));
}

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

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

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

// ===== 스택 코드의 끝 =====

int main(void)
{
    StackType s;

    init_stack(&s);
    push(&s, 1);
    push(&s, 2);
    push(&s, 3);
    push(&s, 4);
    push(&s, 5);
    printf("%d\n", pop(&s));
    printf("%d\n", pop(&s));
    printf("%d\n", pop(&s));
}
```



## 동적 스택 프로그램

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

// 차후에 스택이 필요하면 여기만 복사하여 붙인다.
// ===== 스택 코드의 시작 =====
#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));
}

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

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

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

// ===== 스택 코드의 끝 =====

int main(void)
{
    StackType* s;

    s = (StackType*)malloc(sizeof(StackType));

    init_stack(s);
    push(s, 5);
    push(s, 6);
    push(s, 7);
    printf("%d\n", pop(s));
    printf("%d\n", pop(s));
    printf("%d\n", pop(s));
    free(s);
}
```



## 동적 배열 스택 프로그램

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

typedef int element;
typedef struct {
    element* data; // data은 포인터로 정의된다.
    int capacity; // 현재 크기
    int top;
} StackType;

// 스택 생성 함수
void init_stack(StackType* s)
{
    s->top = -1;
    s->capacity = 100;
    s->data = (element*)malloc(s->capacity * sizeof(element));
}

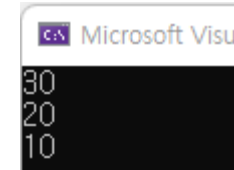
// 공백 상태 검출 함수
int is_empty(StackType* s)
{
    return (s->top == -1);
}
```

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

```
void push(StackType* s, element item)
{
    if (is_full(s)) {
        s->capacity *= 2;
        s->data = (element*)realloc(s->data, s->capacity * sizeof(element));
    }
    s->data[++(s->top)] = item;
}
```

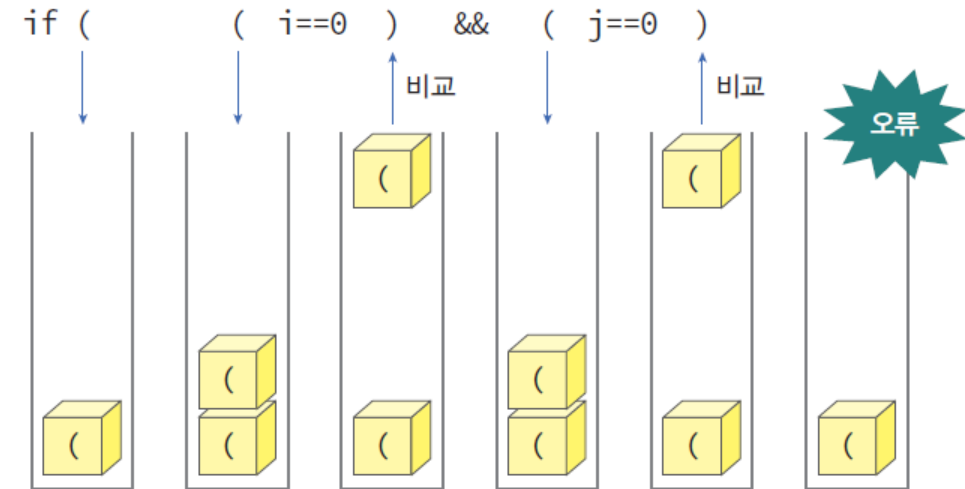
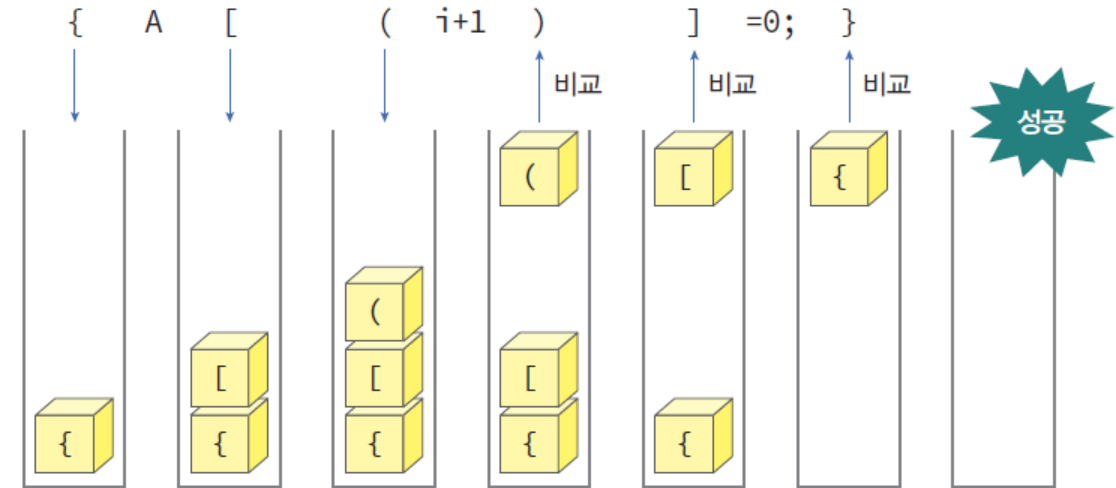
```
// 삭제 함수
element pop(StackType* s)
{
    if (is_empty(s)) {
        fprintf(stderr, "스택 공백 에러\n");
        exit(1);
    }
    else return s->data[(s->top)--];
}
```

```
int main(void)
{
    StackType s;
    init_stack(&s);
    push(&s, 10);
    push(&s, 20);
    push(&s, 30);
    printf("%d \n", pop(&s));
    printf("%d \n", pop(&s));
    printf("%d \n", pop(&s));
    free(s.data);
    return 0;
}
```



## 스택의 응용: 괄호검사

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



## 스택의 응용 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 프로그램 #1

**check\_matching(expr) :**

왼쪽 괄호이면 스택에  
삽입

**while** (입력 expr의 끝이 아니면)

ch ← expr의 다음 글자

**switch**(ch)

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

ch를 스택에 삽입

**break**

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

오른쪽 괄호이면 스택  
에서 삭제비교

**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

## 괄호 검사 프로그램 #2

```

int check_matching(const char *in)
{
    StackType s;
    char ch, open_ch;
    int i, n = strlen(in);    // n= 문자열의 길이
    init_stack(&s);           // 스택의 초기화

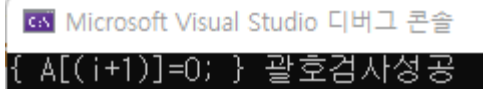
    for (i = 0; i < n; i++) {
        ch = in[i]; // ch = 다음 문자
        switch (ch) {
            case '(': case '[': case '{':
                push(&s, ch);
                break;
            case ')': case ']': case '}':
                if (is_empty(&s)) return 0;
                else {
                    open_ch = pop(&s);
                    if ((open_ch == '(' && ch != ')') ||
                        (open_ch == '[' && ch != ']') ||
                        (open_ch == '{' && ch != '}')) {
                        return 0;
                    }
                    break;
                }
            }
        }
    }
    if (!is_empty(&s)) return 0; // 스택에 남아있으면 오류
    return 1;
}
  
```



## 괄호 검사 프로그램 #3

```
int main(void)
{
    char *p = "{ A[(i+1)]=0; }";
    if (check_matching(p) == 1)
        printf("%s 괄호검사성공\n", p);
    else
        printf("%s 괄호검사실패\n", p);
    return 0;
}
```

- [Ch 4] paren\_matching.c 파일 참조



Microsoft Visual Studio 디버그 콘솔  
{ A[(i+1)]=0; } 괄호검사성공

## 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 계산순서

- 중위표기식-> 후위표기식->계산
- 2+3\*4 -> 234\*+ -> 14
- 모두 스택을 사용

### 수식을 왼쪽에서 오른쪽으로 스캔하

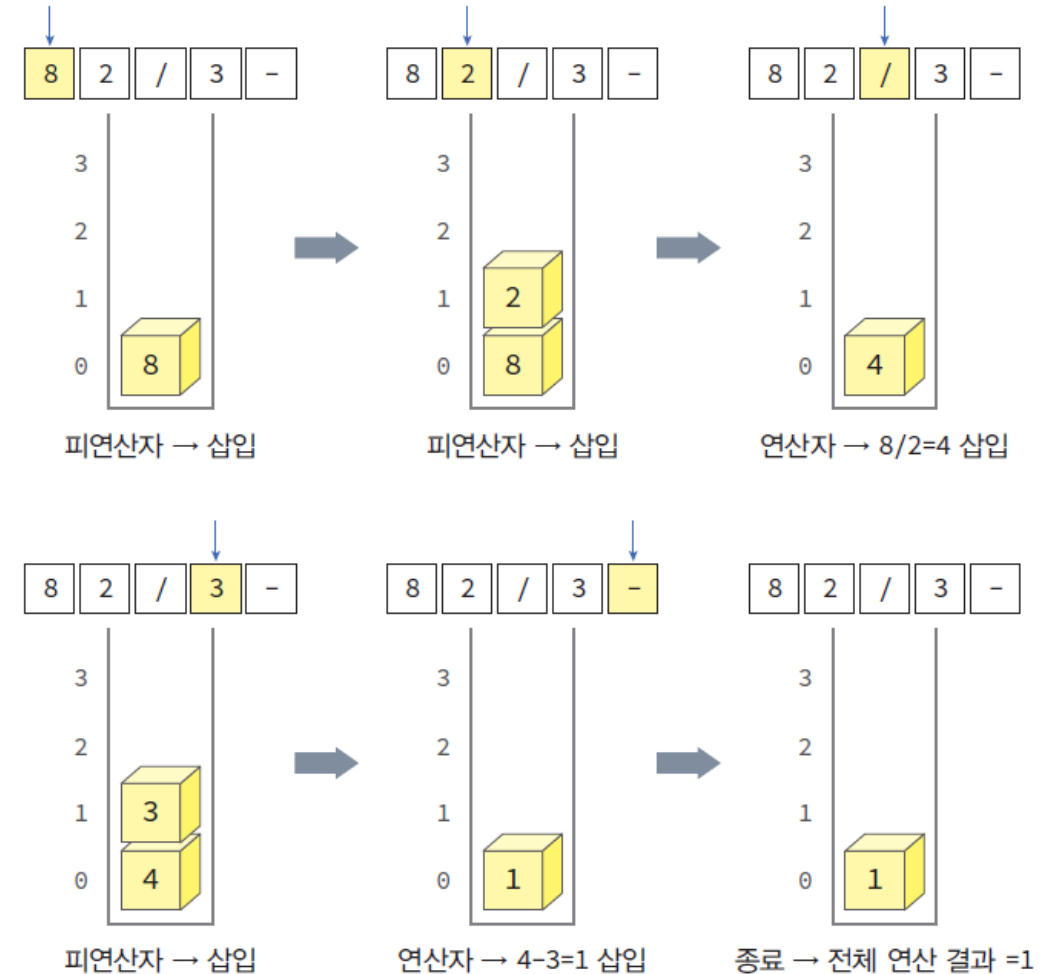
피연산자이면 스택에 저장하고 연산자이면

필요한 수만큼의 피연산자를 스택에서 꺼내

연산을 실행하고 연산의 결과를 다시 스택에 저장

| 토<br>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   |     |     |     |     |     |     |

### (예) 82/3-32\*+



## 후위 표기식 계산 알고리즘 #1

```

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);
  
```

```

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

// 프로그램 “일반적인 배열 스택 프로그램”에서 스택 코드 추가
typedef char element;           // 교체!
// ...
// 프로그램 “일반적인 배열 스택 프로그램”에서 스택 코드 추가 끝

// 후위 표기 수식 계산 함수
int eval(char exp[])
{
    int op1, op2, value, i = 0;
    int len = strlen(exp);
    char ch;
    StackType s;
  
```

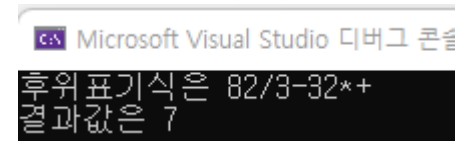
## 후위 표기식 계산 알고리즘 #1

```
init_stack(&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);
}
```

```
int main(void)
{
    int result;
    printf("후위표기식은 82/3-32*+\n");
    result = eval("82/3-32*+");
    printf("결과값은 %d\n", result);
    return 0;
}
```

- [Ch 4] postfix.c 파일 참조

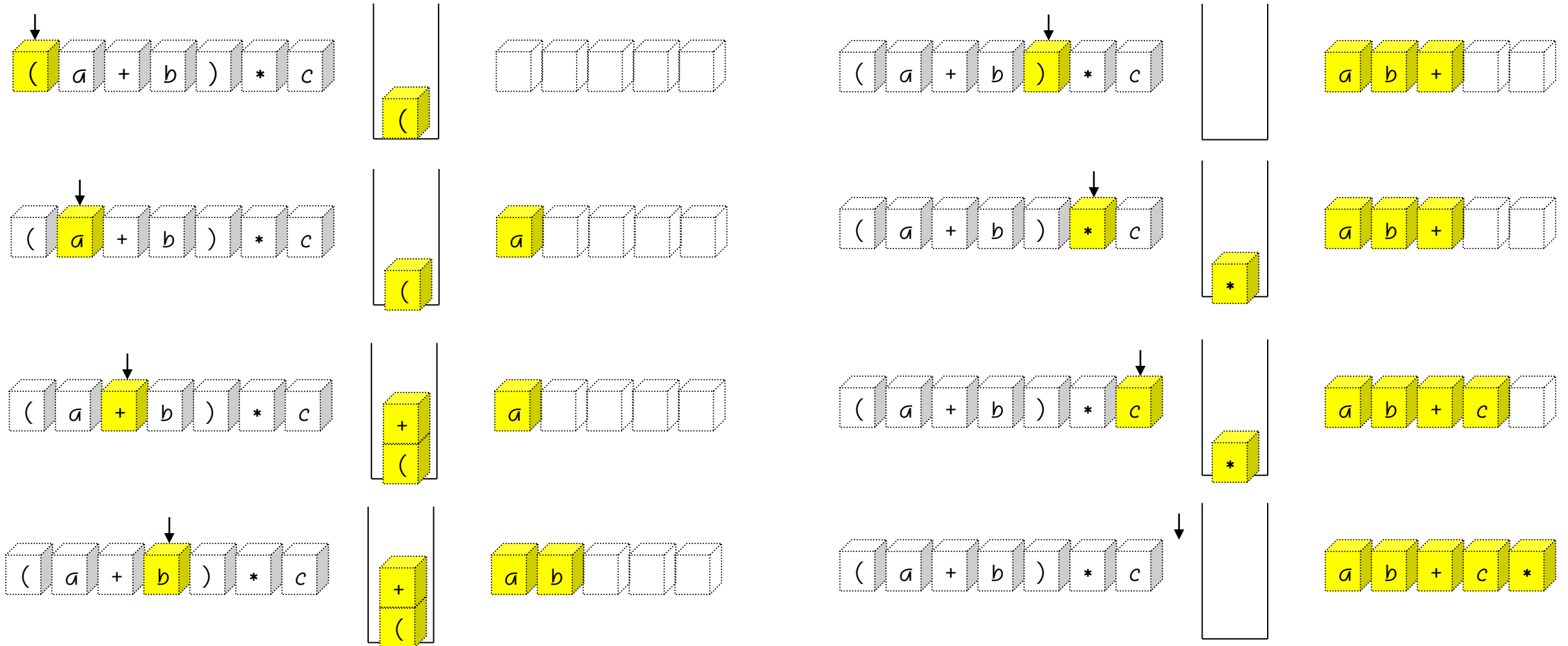


## 중위표기식->후위표기식

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

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

## 중위표기식->후위표기식 예



## 중위표기식->후위표기식 프로그램 #1

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

// 프로그램 “일반적인 배열 스택 프로그램” 에서 스택 코드 추가
typedef char element;          // 교체!
// ...
// 프로그램 “일반적인 배열 스택 프로그램” 에서 스택 코드 추가 끝

// 연산자의 우선순위를 반환한다.
int prec(char op)
{
    switch (op) {
        case '(': case ')': return 0;
        case '+': case '-': return 1;
        case '*': case '/': return 2;
    }
    return -1;
}
```

```
// 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_stack(&s);                      // 스택 초기화
    for (i = 0; i < len; i++) {
        ch = exp[i];
        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;
        }
    }
}
```

## 중위표기식->후위표기식 프로그램 #2

```

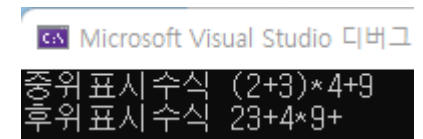
        case '(':    // 왼쪽 괄호
                    push(&s, ch);
                    break;
        case ')':    // 오른쪽 괄호
                    top_op = pop(&s);

                    while (top_op != '(') { // 왼쪽 괄호를 만날때까지 출력
                        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));
}
    
```

```

//
int main(void)
{
    char *s = "(2+3)*4+9";
    printf("중위표시수식 %s \n", s);
    printf("후위표시수식 ");
    infix_to_postfix(s);
    printf("\n");
    return 0;
}
    
```

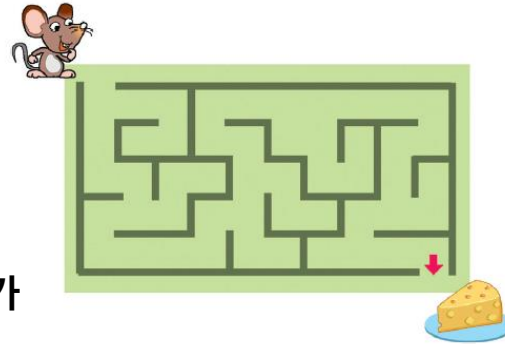
- [Ch 4] infix.c 파일 참조



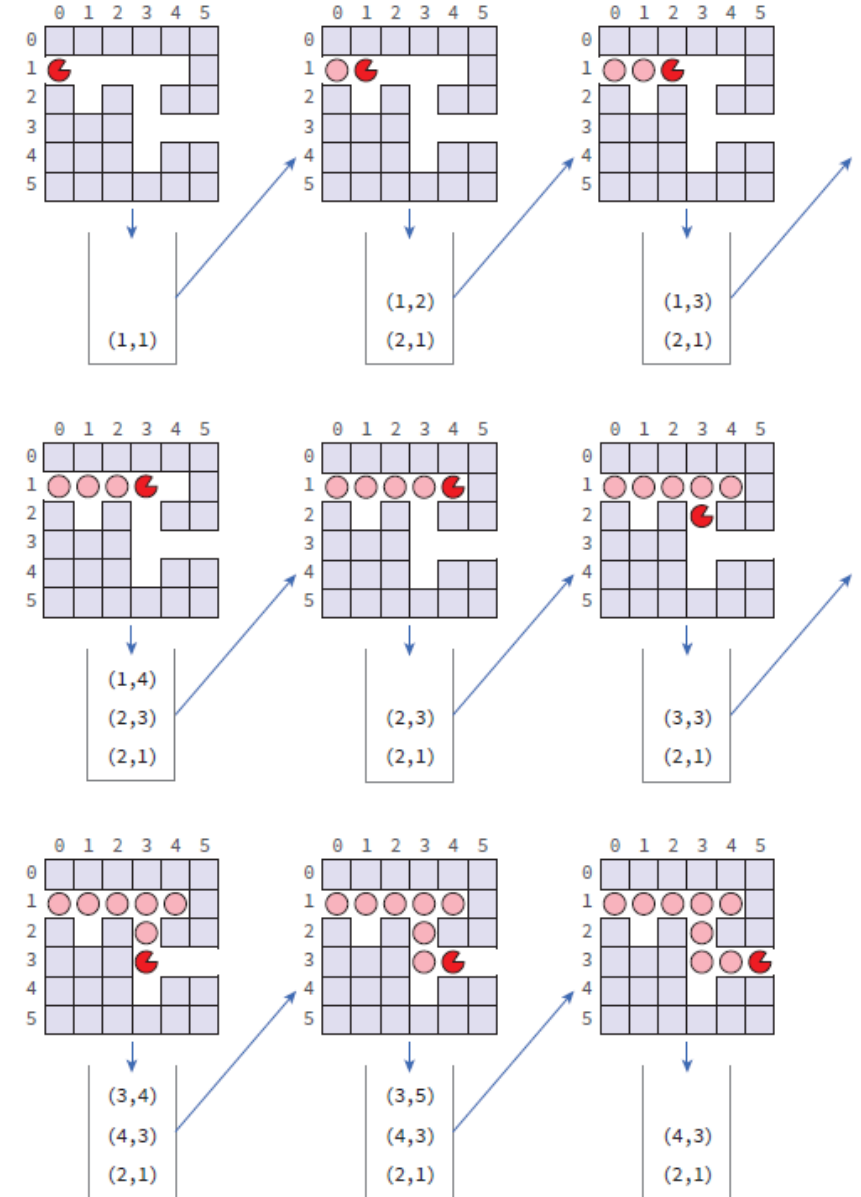


## 미로탐색문제

- 체계적인 방법 필요
- 현재의 위치에서 가능한 방향을 스택에 저장해놓았다가 막다른 길을 만나면 스택에서 다음 탐색 위치를 꺼낸다.



스택  $s$ 과 출구의 위치  $x$ , 현재 생쥐의 위치를 초기화  
 while( 현재의 위치가 출구가 아니면 )  
   do 현재 위치를 방문한 것으로 표기  
     if( 현재 위치의 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 탐색 프로그램 #1

```
// 프로그램 “일반적인 배열 스택 프로그램” 에서 스택 코드 추가
// ...
typedef struct {                // 교체!
    short r;
    short c;
} element;
// 프로그램 “일반적인 배열 스택 프로그램” 에서 스택 코드 추가 끝

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' },
};
```

```
// 위치를 스택에 삽입
void push_loc(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);
    }
}

// 미로를 화면에 출력한다.
void maze_print(char maze[MAZE_SIZE][MAZE_SIZE])
{
    printf("\n");
    for (int r = 0; r < MAZE_SIZE; r++) {
        for (int c = 0; c < MAZE_SIZE; c++) {
            printf("%c", maze[r][c]);
        }
        printf("\n");
    }
}
```

## 미로 탐색 프로그램 #2

```
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("실패\n");
            return;
        }
        else
            here = pop(&s);
    }
    printf("성공\n");
    return 0;
}
```

- [Ch 4] maze.c 파일 참조

Micro:

```
111111 111111
..01001 ..1001
100011 1...11
101011 101011
10100x 10100x
111111 111111

111111 111111
..1001 ..1001
100011 1...11
101011 101.11
10100x 10100x
111111 111111

111111 111111
..1001 ..1001
1.0011 1...11
101011 101.11
10100x 101.0x
111111 111111

111111 111111
..1001 ..1001
1..011 1...11
101011 101.11
10100x 101..x
111111 111111
성공
```

**Thank you**

**Q & A**