3 장. 스택 (stack)

Jinseog Kim

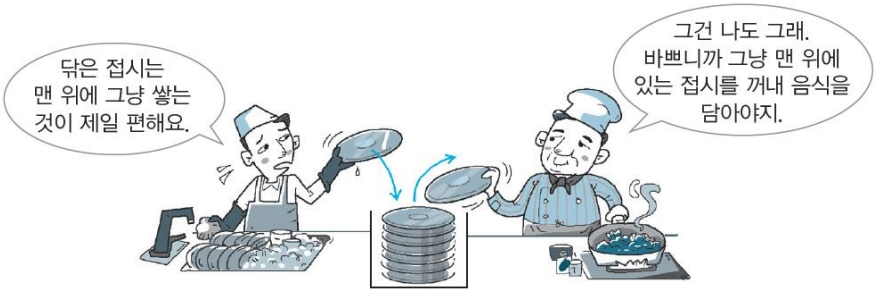
dongguk UNIVERSITY WISE

학습목표

1. 자료구조 스택을 이해한다.
2. 배열을 이용한 스택을 구현한다.
3. 스택의 응용

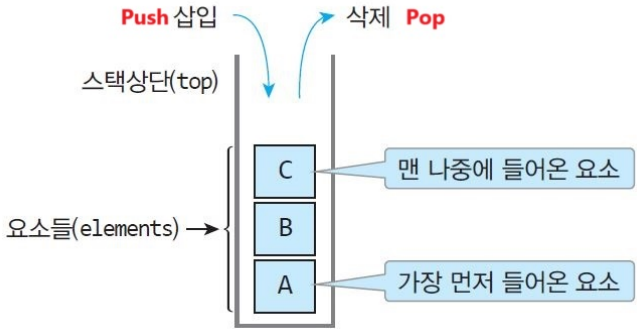
스택이란?

- 스택은 마지막에 추가된 요소가 가장 먼저 제거되도록 설계한 선형 데이터 구조
- **후입선출 (LIFO; Last-In First-Out)** - 마지막에 삽입된 요소가 가장 먼저 제거
- 주요 연산으로는 push(요소 삽입, 추가) 와 pop(요소 삭제), peek(맨위 요소 확인) 등등이 있음



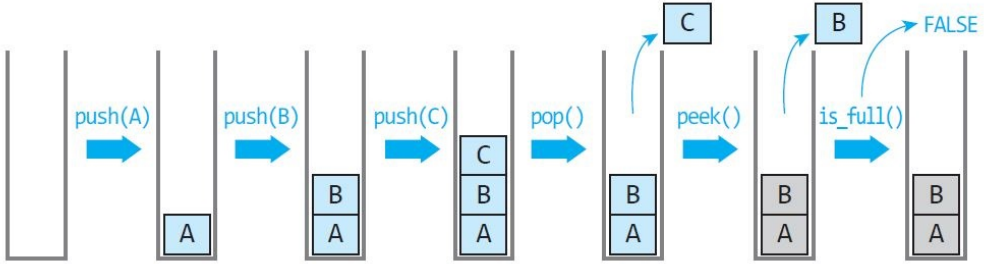
스택이란?

- 스택의 구조
  - 스택 상단: top
  - 스택 하단: 불필요



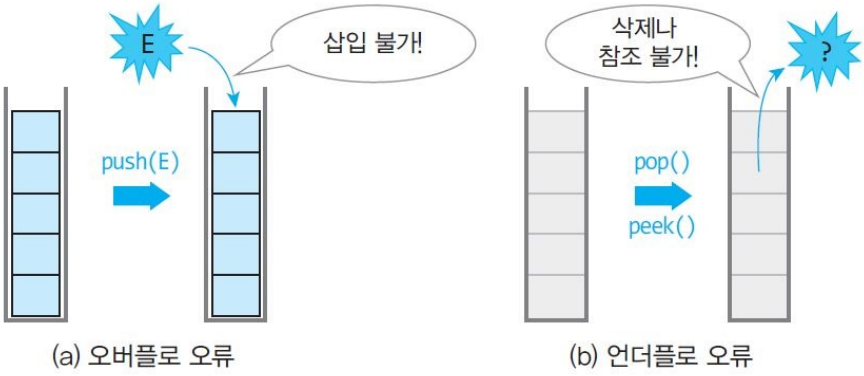
스택의 추상자료형 (ADT)

- **데이터 (객체)**: 후입선출 방식을 가능하게 하는 요소의 모임
- **연산 (함수)**
  - `initialize()`: 스택 초기화
  - `push(e)`: 스택의 맨위에 요소 `e` 를 추가
  - `pop()`: 스택의 맨위 요소를 꺼내 반환
  - `is_full()`: 스택이 꽉 차있으면 TRUE, 아니면 FALSE
  - `is_empty()`: 스택이 비어있으면 TRUE, 아니면 FALSE
  - `peek()`: 스택의 맨위 요소를 삭제하지 않고 반환



스택에서 발생할 수 있는 오류

- **오버플로우 (overflow)**: 스택이 꽉차있어 더 이상 요소를 추가할 수 없는 경우
- **언더플로우 (underflow)**: 스택이 비어있어 요소를 가져오거나 삭제할 수 없는 경우

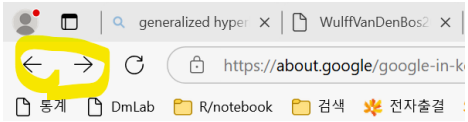


스택오류 방지 스택의 크기를 적절히 관리, 삽입/제거를 수행하기 전 스택의 상태를 확인

스택의 활용 분야

- 편집기의 되돌리기 (undo) 기능

- 웹 브라우저의 이전 페이지로 이동 기능

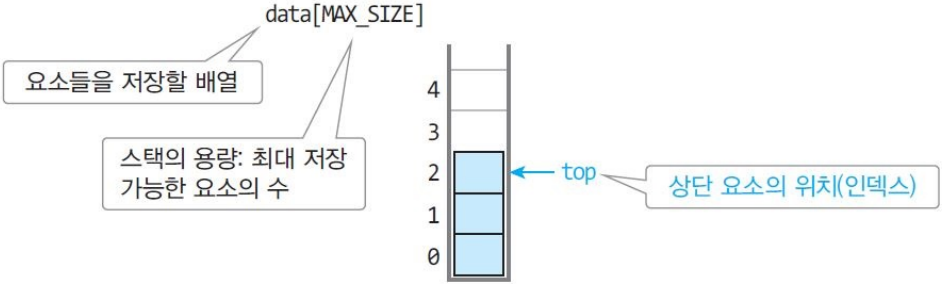


- **괄호 검사:** 코드나 수식에서 괄호의 좌우 짝이 일치하는 지를 검사
- 계산기 프로그램, 미로 탐색
- **함수 호출 관리:** 호출된 함수의 상태를 시스템 스택에 저장, 함수가 종료되면 스택에서 제거

스택의 구현 방법

스택을 구현하는 두가지 방법

1. **배열을 활용:** 고정된 크기의 배열을 이용



2. **연결 리스트 (linked list) 를 이용:** 동적 크기의 스택 구현으로 나중에 배울 것임 (5 장)

배열로 스택을 구현할 때 장단점

- 장점:
  - 구현하기 쉬움
  - 포인터를 쓰지 않아 메모리가 절약
- 단점:
  - 스택의 전체 크기가 미리 정의되어야 함 (고정 크기)
  - 필요에 따라 크게 하거나 줄일 수 없음

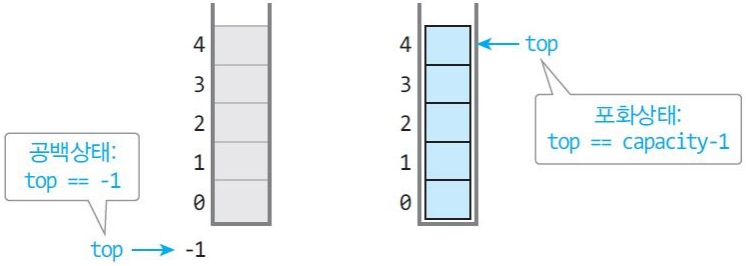
스택의 구현 (데이터)

- 데이터

```
1  typedef struct{
2    char name[20];
3    int score;
4  }Element; // 스택에 저장할 요소를 구조체로 정의
5
6  #define MAX_SIZE 100;    // 스택의 고정크기 정의
7  Element stack[MAX_SIZE]; // 구조체 배열
8  int top = -1;            // 스택의 상단 위치 (스택이 비어있는 상태)
```

스택의 구현 (연산)

- 상태 검사: is_empty, is_full

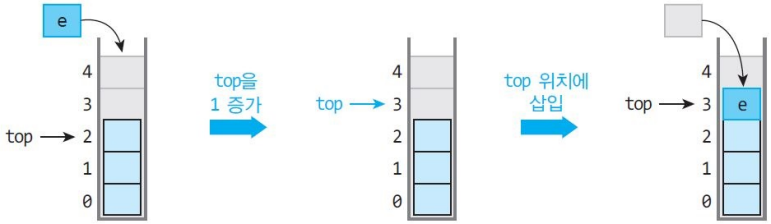



```c
int is_empty(){
  if (top == -1) return(1);
  else return(0);
}
```

```c
int is_full(){
  if (top == MAX_SIZE - 1) return(1);
  else return(0);
}
```

스택의 구현 (연산)

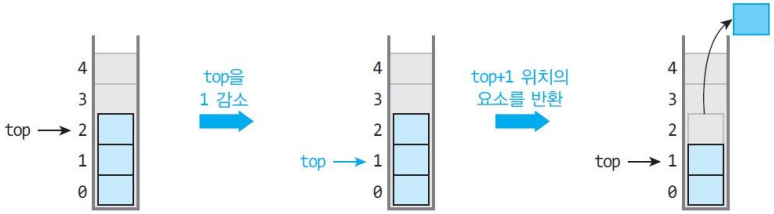
- 요소 삽입 (push)




```
1  void push(Element e){
2      if (is_full())
3          error("Overflow Error!");
4      else data[++top] = e;
5  }
```

스택의 구현 (연산)

- 상단 요소 삭제 (pop)



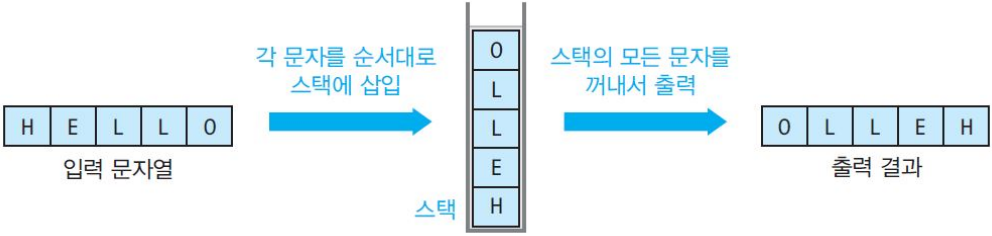

```
1  Element pop(){
2      if (is_empty())
3          error("Underflow Error!");
4      return data[top--];
5  }
```

스택의 구현 (연산)

- 상단 요소를 참조 (peek)

```
1  Element peek()
2  {
3      if (is_empty())
4          error("Underflow Error!");
5      return data[top];
6  }
```

Lab: 문자열 뒤집어 출력하기



Lab: 문자열 뒤집어 출력하기

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

#define MAX_SIZE 2000
typedef char Element;
#include "ArrayStack.h"
```

- ArrayStack.h 는 예제코드를 이용할 것

Lab: 문자열 뒤집어 출력하기

```
1  void main(){
2      char str[200];
3
4      init_stack();
5      printf(" 문자열 입력: ");
6      fgets(str, 200, stdin); //gets_s(str, 200);
7      for (int i = 0; str[i] != '\0'; i++)
8          push(str[i]);
9
10     printf(" 문자열 출력: ");
11     while (!is_empty())
12         putchar(pop());
13     printf("\n");
14 }
```

// 파일 *stream* 으로 부터 *n - 1* 개의 문자를 읽어 문자열 *str* 에 저장

```
char *fgets(char *str, int n, FILE *stream);
```

Lab: 문자열 뒤집어 출력하기

- 실행결과

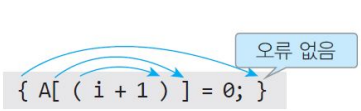
  문자열 입력: `data structure`
  문자열 출력:
  `erutcurts atad`

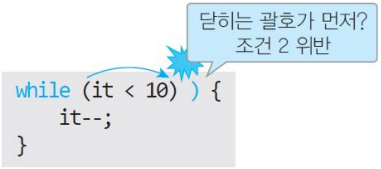
스택의 응용: 괄호 검사

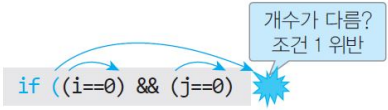
```
int find_array_max(int score[], int n)
{
    int i, tmp=score[0];
    for( i=1 ; i<n ; i++ ) {
        if( score[i] > tmp ) {
            tmp = score[i];
        }
    }
    return tmp;
}
```

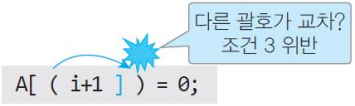
스택의 응용: 괄호 검사

- 괄호 검사 조건
  1. 왼쪽과 오른쪽의 **괄호의 수는 같아야** 함
  2. 같은 종류의 괄호는 **왼쪽 먼저** 나와야 함
  3. 서로 다른 괄호가 교차하여 쌍을 이루면 안됨 (나중에 시작한 괄호와 같은 괄호가 먼저 나와야)





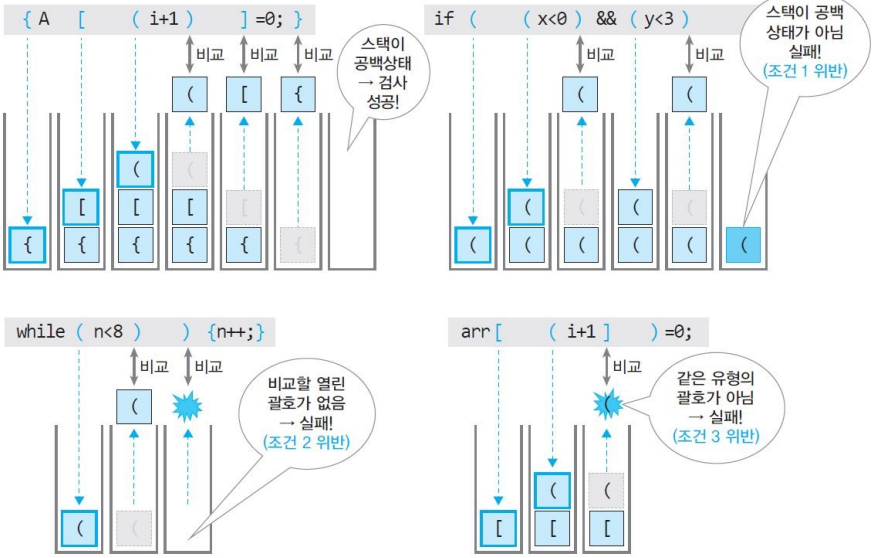




스택의 응용: 괄호 검사 알고리즘

```
1  check_matching(expr)
2    init_stack()
3    while(expr 의 끝이 아니면) :
4      ch <- expr 의 다음 글자
5      if ch in { '(', '[', '{' } : //왼쪽 괄호의 종류
6        push(ch)
7      else if ch in { ')', ']', '}' }: //오른쪽 괄호의 종류
8        if is_empty():
9          오류 (by 2: 왼쪽 먼저나와야 함)
10       else:
11         open_ch <- pop()
12         if (ch 와 open_ch 가 같은 짝이 아니면):
13           오류 (by 3: 다른 종류의 괄호가 교차하면 안됨)
14
15   if is_empty() == FALSE: 오류 (by 1: 괄호의 수는 같아야)
16   else: 성공
```

스택의 응용: 괄호 검사 예


{ A [(i+1)] =0; }
비교
비교
비교
스택이 공백상태 → 검사 성공!
if ((x<0) && (y<3)
비교
비교
스택이 공백 상태가 아님 실패! (조건 1 위반)
while (n<8) {n++;}
비교
비교
비교할 열린 괄호가 없음 → 실패! (조건 2 위반)
arr[(i+1]) =0;
비교
같은 유형의 괄호가 아님 → 실패! (조건 3 위반)

스택의 응용: 괄호 검사 구현

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

#define MAX_SIZE 100
typedef char Element; // 스택 요소 Element 의 자료형
#include "ArrayStack.h" // 스택의 ADT(데이터와 연산) 포함
```

스택의 응용: 괄호 검사 구현

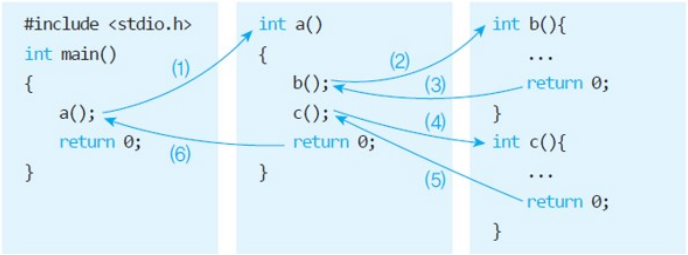
```
int check_matching(char expr[]){
    int i = 0, prev;
    init_stack();
    while (expr[i] != '\0') {
        char ch = expr[i++];
        if (ch == '[' || ch == '(' || ch == '{')
            push(ch);
        else if (ch == ']' || ch == ')' || ch == '}') {
            if (is_empty())
                return 2;       // 조건 2 위반
            prev = pop();
            if ((ch == ']' && prev != '[')
                || (ch == ')' && prev != '(')
                || (ch == '}' && prev != '{'))
                return 3;       // 조건 3 위반
        }
    }
    if (!is_empty()) return 1;  // 조건 1 위반
    else return 0;              // 괄호 정상
}
```

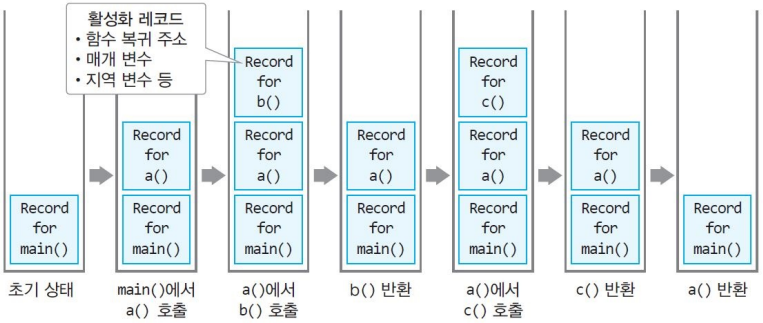
스택의 응용: 괄호 검사 구현

```
void main()
{
    char expr[4][80] = {
        "{A[(i+1)]=0;}",
        "if((i==0) && (j==0)",
        "while(n<8)){n++;}",
        "arr[(i+1]) = 0;" };

    for (int i = 0; i < 4; i++) {
        int errCode = check_matching(expr[i]);
        if (errCode == 0) printf("%-20s -> 정상\n", expr[i]);
        else printf("%-20s -> 오류 (조건%d 위반)\n", expr[i], errCode);
    }
}
```

시스템 스택과 재귀 호출


#include <stdio.h>
int main()
{
a();
return 0;
}
int a()
{
b();
c();
return 0;
}
int b(){
...
return 0;
}
int c(){
...
return 0;
}
(1)
(2)
(3)
(4)
(5)
(6)


활성화 레코드
• 함수 복귀 주소
• 매개 변수
• 지역 변수 등
Record for main()
Record for a()
Record for b()
Record for c()
초기 상태
main()에서 a() 호출
a()에서 b() 호출
b() 반환
a()에서 c() 호출
c() 반환
a() 반환

- 재귀 (recursion): 어떤 함수가 자기 자신을 다시 호출하여 문제를 해결하는 프로그래밍 기법
  - 문제 자체가 순환적인 경우 (예: 팩토리얼 등)
  - 순환적으로 정의되는 자료구조 (예: 이진 트리 등)
- **(예) 팩토리얼 구하기**
  - 반복적인 정의

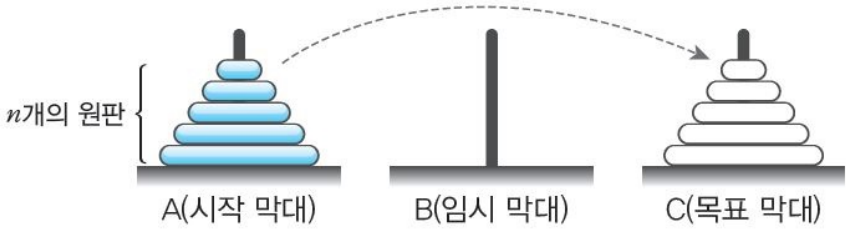
$$n! = 1 \times 2 \times 3 \times \ldots \times n$$

  - 재귀적인 정의

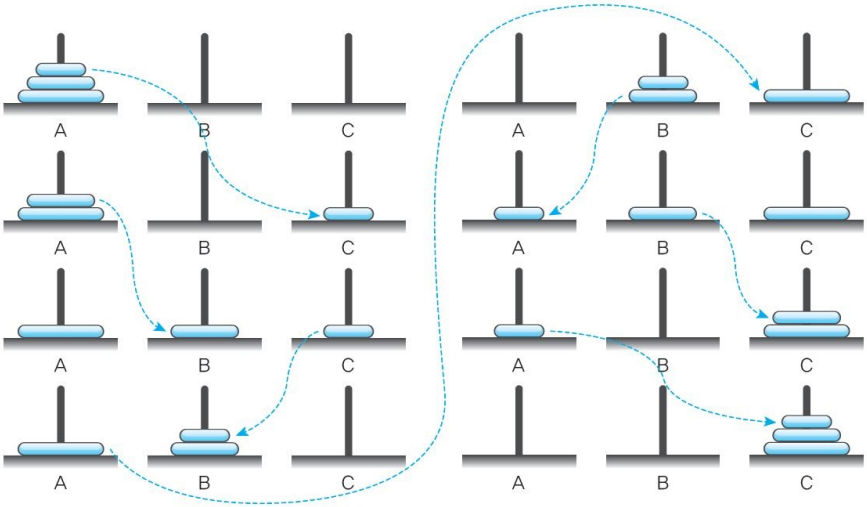
$$n! = \begin{cases} 1, & n = 1 \\ n(n-1)!, & n > 1 \end{cases}$$

연습: 하노이 탑 문제

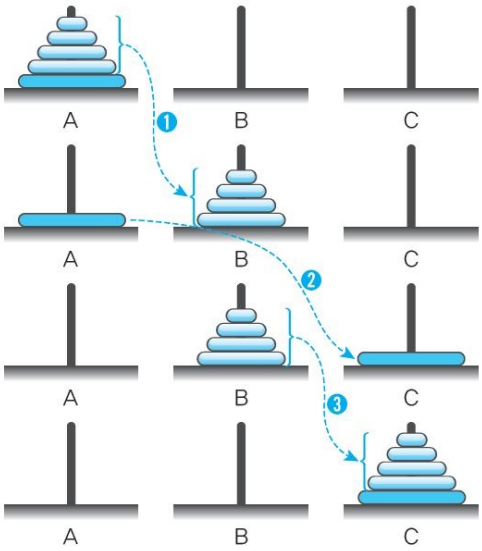
- 세 개의 기둥과 크기가 다른 원반으로 구성, 한 기둥에서 다른 기둥으로 옮기는 것
- 규칙
  1. 한 번에 하나의 원반만 이동
  2. 큰 원반 위에 작은 원반을 올릴 수 없음
  3. 모든 원반을 다른 기둥으로 옮겨야 함



연습: 하노이 탑 문제 (n=3)



연습: 하노이 탑 문제



1. 단계: A에 있는 $n-1$개의 원판을 C를 임시 막대로 이용해서 B로 이동

2. 단계: A에 남은 하나의 원판을 C로 이동

3. 단계: B에 있는 $n-1$개의 원판을 A를 임시 막대로 이용해서 C로 이동

연습: 하노이 탑 문제

```c
// 코드 3.10 하노이의 탑
#include <stdio.h>
void hanoi_tower(int n, char from, char tmp, char to)
{
    if (n == 1)
        printf(" 원판 1: %c --> %c\n", from, to);
    else {
        hanoi_tower(n - 1, from, to, tmp);           // 1 단계
        printf(" 원판 %d: %c --> %c\n", n, from, to); // 2 단계
        hanoi_tower(n - 1, tmp, from, to);           // 3 단계
    }
}

void main() {
  hanoi_tower(4, 'A', 'B', 'C');
}
```

연습문제

- 교재의 연습문제: 3.1 ~ 3.14, P3.28
- https://www.geeksforgeeks.org/top-50-problems-on-stack-data-structure-asked-in-interviews/