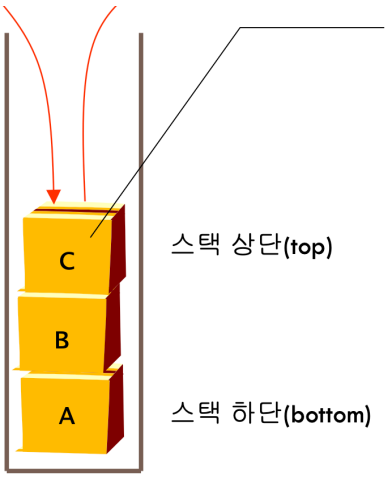
3월 3주차 - 스택

상태	완료
데드라인	@April 2, 2025
PROCESS	💚 데이터구조

▼ 1. 스택

- stack : 쌓아놓은 더미
- 특징 : LIFO(Last-In First-Out) 가장 최근에 들어온 데이터가 가장 먼저 나감



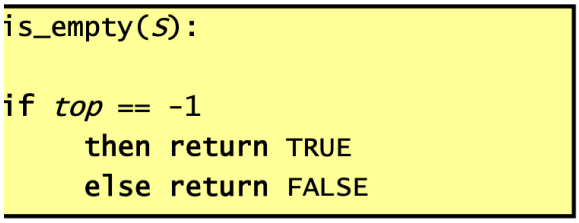
▼ Stack ADT

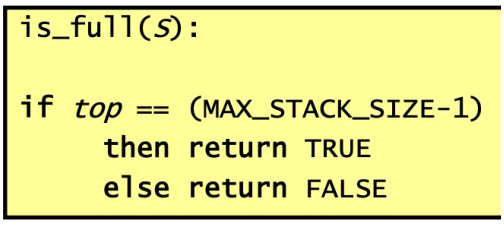
- 객체 : 0개 이상의 원소를 가지는 유한 선형 리스트
- 연산
  - **create(size)** 최대 크기가 size인 공백 스택을 형성한다.
  - **is_full(s)** : if(stack의 원소수 == size) return TRUE; else return FALSE;
  - **is_empty(s)** : if(stack의 원소수 == 0) return TRUE; else return FALSE;
  - **push(s, item)** : if(is_full(s)) return ERROR_STACKFULL; else 맨 위에 item추가
  - **pop(s)** : if(is_empty(s)) return ERROR_STACKEMPTY; else 맨 위에 제거해서 반환
  - **peek(s)** : if(is_empty(s)) return ERROR_STACKEMPTY; else 맨 위 제거안하고 반환
  - ⇒ pop연산은 요소를 스택에서 완전히 삭제하면서 가져옴

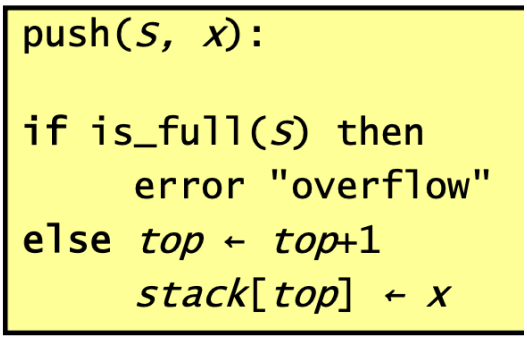
▼ 스택의 구현

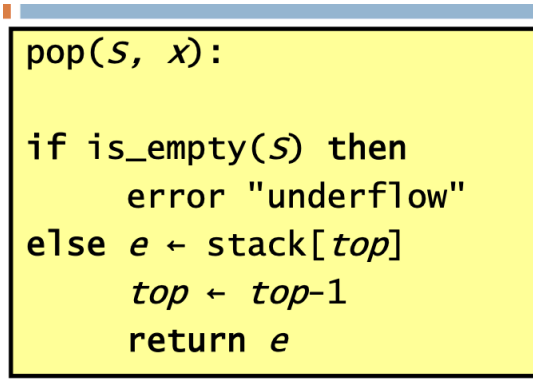
- 배열을 이용한 스택 구현

- 1차원 배열 stack[]
- 스택에서 가장 최근에 입력되었던 자료를 가리키는 top 변수
- 가장 먼저 들어온 요소는 stack[0], 가장 최근에 들ㅇ어온 요소는 stack[top]에 저장
- 스택이 공백 상태이면 top = -1









▼ 전역변수로 구현

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

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

int is_empty(){
   return top == -1;
}
int is_full(){
   return (top == (MAX-1));
}
```

```
void push(element item){
    if(is_full()){
        fprintf(stderr, "스택 포화 에러\n");
        exit(1);
    }else stack[++top] = item;
}
element pop(){
    if(is_empty()){
        fprintf(stderr, "스택 공백 에러\n");
        exit(1);
    }
    else return stack[top--];
}
element peek(){
    if(is_empty()){
        fprintf(stderr, "스택 공백 에러\n");
        exit(1);
    }else return stack[top];
}

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

▼ 구조체 배열 사용

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

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

```
void init_stack(StackType *s){
    s->top = -1;
}
int is_empty(StackType *s){
    return (s->top == -1);
}
int is_full(StackType *s){
    return (s->top == (MAX-1));
}
void push(StackType *s, element item){
    if(is_full(s)){
        fprintf(stderr, "스택 포화 에러\n");
        exit(1);
    }else s->data[++(s->top)] = item;
}
element pop(StackType *s){
    if(is_empty(s)){
        fprintf(stderr, "스택 공백 에러\n");
        exit(1);
    }else return s->data[(s->top)--];
}
element peek(StackType *s){
    if(is_empty(s)){
        fprintf(stderr, "스택 공백 에러\n");
        exit(1);
    }else return s->data[(s->top)];
}

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

▼ 동적 스택

```
int main(){
    StackType *s;
    s = (StackType *)malloc(sizeof(StackType));
    ...
}
```

▼ 동적 배열 스택

- malloc()을 호출해서 실행 시간에 메모리를 할당받아 스택을 생성

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

typedef int element;
typedef struct{
    element *data;
    int capacity;
    int top;
}StackType;

void init_stack(StackType *s){
    s->top = -1;
    s->capacity = 100;
    s->data = (element *)malloc(s->capacity * sizeof(element));
}
int is_empty(StackType *s){
    return s->top == -1;
}
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;
}
element pop(StackType *s){
    if(is_empty(s)){
        fprintf(stderr, "스택 공백 에러");
```

```
        exit(1);
    }else return s→data[(s→top)--];
}

int main(){
    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;
}
```

▼ 스택의 응용

▼ 괄호 검사

1. 왼쪽 괄호의 개수와 오른쪽 괄호의 개수가 같아야 한다.
2. 같은 괄호에서 왼쪽 괄호는 오른쪽 괄호보다 먼저 나와야 한다.
3. 괄호 사이에는 포함관계만 존재한다. - 다른 종류 괄호의 교차 X

▼ 알고리즘

1. 문자열에 있는 괄호를 차례대로 조사하면서 왼쪽 괄호를 만나면 스택에 삽입, 오른쪽 괄호를 만나면 스택에서 top 괄호를 삭제한 후 오른쪽 괄호와 짝이 맞는지 확인
2. 이 때 스택이 비어있으면 조건 12 위배 괄호의 짝이 안맞으면 3에 위배
3. 마지막 괄호까지 조사한 후에도 스택에 괄호가 있으면 1에 위배 - 0반환, 그렇지 않으면 1 반환

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

typedef char element;
typedef struct {
    element data[MAX];
    int top;
```

```
}StackType;

void init_stack(StackType *s){
    s->top = -1;
}
int is_empty(StackType *s){
    return s->top == -1;
}
int is_full(StackType *s){
    return (s->top == (MAX -1));
}
void push(StackType *s, element item){
    if(is_full(s)){
        fprintf(stderr, "스택 포화 오류");
        return;
    }else s->data[++(s->top)] = item;
}
element pop(StackType *s){
    if(is_empty(s)){
        fprintf(stderr, "스택 공백 에러");
        exit(1);
    }else return s->data[(s->top)--];
}
element peek(StackType *s){
    if(is_empty(s)){
        fprintf(stderr, "스택 공백 에러");
        exit(1);
    }else return s->data[(s->top)];
}
int check_matching(const char *in){
    StackType s;
    char ch, open_ch;
    int i, n = strlen(in);
    init_stack(&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 0;
            else{
               open_ch = pop(&s);
               if((open_ch == '(' && ch != ')') || (open_ch == '{' && ch != '}') || (open_c

            }break;
      }
   }
   if(is_empty(&s)) return 1;
   return 0;
}

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

▼ 수식의 계산

prefix, infix, postfix

- 후외표기식 장점 : 괄호가 필요 없음
- 중위 → 후위/전위 변환
  - 중위표기식의 모든 식에 대해 괄호를 표기
  - 이항 연산자들을 모두 괄호 위치로 이동
  - 괄호 삭제

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

typedef char element;
typedef struct {
   element data[MAX];
```

```
    int top;
}StackType;

void init_stack(StackType *s){
    s->top = -1;
}
int is_empty(StackType *s){
    return s->top == -1;
}
int is_full(StackType *s){
    return (s->top == (MAX -1));
}
void push(StackType *s, element item){
    if(is_full(s)){
        fprintf(stderr, "스택 포화 오류");
        return;
    }else s->data[++(s->top)] = item;
}
element pop(StackType *s){
    if(is_empty(s)){
        fprintf(stderr, "스택 공백 에러");
        exit(1);
    }else return s->data[(s->top)--];
}
element peek(StackType *s){
    if(is_empty(s)){
        fprintf(stderr, "스택 공백 에러");
        exit(1);
    }else return s->data[(s->top)];
}
int eval(char exp[]){
    int op1, op2, value, i = 0;
    int len = strlen(exp);
    char ch;
    StackType s;

    for(i=0;i<len;i++){
        ch=exp[i];
        if(ch!='+' && ch != '-' && ch!='*' && ch!='/'){
            value = ch-'0';
            push(&s, value);
```

```
        }else{
            op2 = pop(&s);
            op1 = pop(&s);
            switch(ch){
                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(){
    int result;
    printf("후위 표기식은 82/3-32*+\n");
    result = eval("82/3-32*+");
    printf("%d\n", result);
    return 0;
}
```

- 전위 → 후위

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

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

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

```
    return (s→top == (MAX -1));
}
void push(StackType *s, element item){
    if(is_full(s)){
        fprintf(stderr, "스택 포화 오류");
        return;
    }else s→data[++(s→top)] = item;
}
element pop(StackType *s){
    if(is_empty(s)){
        fprintf(stderr, "스택 공백 에러");
        exit(1);
    }else return s→data[(s→top)--];
}
element peek(StackType *s){
    if(is_empty(s)){
        fprintf(stderr, "스택 공백 에러");
        exit(1);
    }else return s→data[(s→top)];
}
int prec(char op){
    switch(op){
        case '(': case ')': return 0;
        case '+': case '-': return 1;
        case '*': case '/': return 2;
    }
    return -1;
}
void infix_to_postfix(char exp[]){
    char ch, top_op;
    int len = strlen(exp);
    StackType s;
    init_stack(&s);
    for(int i = 0; i < len; i++){
        ch = exp[i];
        switch(ch){
            case '+': case '-': case '*': case '/':
                while(!is_empty(&s) && prec(ch) <= prec(peek(&s)))
                    printf("%c", pop(&s));
                push(&s, ch);
                break;
```

```
            case '(':
                push(&s, ch);
                break;
            case ')':
                top_op = pop(&s);
                while(top_op!='('){
                    printf("%c", top_op);
                    top_op = pop(&s);
                }break;
            default:
                printf("%c", ch);
                break;
        }
    }while(!is_empty(&s))
        printf("%c", pop(&s));
}

int main(){
    char *s = "(2+3)*4+9";
    printf("중위 표기식 %s\n", s);

    printf("후위표기식 :");
    infix_to_postfix(s);

    return 0;
}
```