/\*

240325 스택

∙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.

\*/

// 1. 전역변수로 구현하는 스택

#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 - 스택의 가장 윗부분

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()); // 출력하고 pop을 수행하여, 4를 출력하고 4를 스택에서 없앤다.

printf("%d\n", pop()); // 출력하고 pop을 수행하여, 3를 출력하고 3을 스택에서 없앤다.

printf("%d\n", pop()); // 출력하고 pop을 수행하여, 2를 출력하고 2을 스택에서 없앤다.

printf("%d\n", pop()); // 출력하고 pop을 수행하여, 1를 출력하고 1을 스택에서 없앤다.

printf("%d\n", pop()); // 마지막으로 입력된 1까지 이전에서 없어졌으므로, 에러 출력

return 0;

}

// 2. 구조체 배열 스택

#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--];

}

// 피크함수

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;

}

// 3. 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));

}

// 삽입함수

void push(StackType\* s, element item)

{

if (is\_full(s)) {

fprintf(stderr, "스택 포화 에러\n");

return;

}

else s->data[++(s->top)] = item;

}

// 삭제함수

element pop(StackType\* s)

{

if (is\_empty(s)) {

fprintf(stderr, "스택 공백 에러\n");

exit(1);

}

else return s->data[(s->top)--];

}

// 피크함수

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));

}

// 4. 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));

}

// 삽입함수

void push(StackType\* s, element item)

{

if (is\_full(s)) {

fprintf(stderr, "스택 포화 에러\n");

return;

}

else s->data[++(s->top)] = item;

}

// 삭제함수

element pop(StackType\* s)

{

if (is\_empty(s)) {

fprintf(stderr, "스택 공백 에러\n");

exit(1);

}

else return s->data[(s->top)--];

}

// 피크함수

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);

}

// 5. 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;

}

// 삭제함수

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;

}

// 괄호 검사 프로그램

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_STACK\_SIZE 100

typedef char element; // 교체!

// 차후에 스택이 필요하면 여기만 복사하여 붙인다.

// ===== 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));

}

// 삽입함수

void push(StackType\* s, element item)

{

if (is\_full(s)) {

fprintf(stderr, "스택 포화 에러\n");

return;

}

else s->data[++(s->top)] = item;

}

// 삭제함수

element pop(StackType\* s)

{

if (is\_empty(s)) {

fprintf(stderr, "스택 공백 에러\n");

exit(1);

}

else return s->data[(s->top)--];

}

// 피크함수

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 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;

}

int main(void)

{

const char\* p = "{ A[(i+1)]=0; }";

if (check\_matching(p) == 1)

printf("%s 괄호검사성공\n", p);

else

printf("%s 괄호검사실패\n", p);

return 0;

}