# 20121277 김주호 HW3 보고서

이번 과제에서 구현한 ADT는 크게 2가지이다. Stack과 원형 Deque가 그것인데, 이 자료형들에 대한 연산들을 먼저 설명한다.

- Stack

① void init_stack(StackType * s)
스택 초기화 함수이다. 스택의 top을 -1로 설정한다.
② int is_empty(StackType * s)
스택의 공백 여부를 알려주는 함수이다.
③ int is_full(StackType * s)
스택의 포화상태 여부를 알려주는 함수이다.
④ void push(StackType * s, element item)
스택에 item 원소를 삽입하는 함수이다. top의 값이 하나 증가한다.
⑤ element pop(StackType * s)
스택의 top 원소를 삭제하는 함수이다. 삭제 대상인 원소의 값을 return한다.
⑥ element peek(StackType* s)
스택의 top 원소를 조회하는 함수이다. pop()과 peek()은 동일한 top원소 값을 return 한다. 그러나 pop()은 스택에서 top 원소를 제거한다는 차이가 있다.
⑦ int get_size(StackType * s)
스택에 담긴 원소의 개수를 알려준다.

- 원형 Deque

① void init_deque(DequeType * q)
덱을 초기화하는 함수이다. front와 rear의 값을 0으로 설정한다.
② int is_empty(DequeType * q)
덱의 공백 여부를 알려주는 함수이다.
③ int is_full(DequeType * q)
덱의 포화상태 여부를 알려주는 함수이다.
④ void add_front(DequeType * q, element item)
덱의 앞 쪽에 해당 원소를 삽입하는 함수이다.
⑤ element get_front(DequeType* q)
덱의 맨 앞에 저장된 원소를 조회하는 함수이다.
⑥ element delete_front(DequeType* q)
덱의 맨 앞의 원소를 제거하는 함수이다. front의 값을 하나 증가 시킨다. (원형 덱이므로, %연산을 통해 index 오류가 없도록 조정한다.)
⑦ void add_rear(DequeType * q, element item)

덱의 뒤 쪽에 해당 원소를 삽입하는 함수이다.

⑧ element get_rear(DequeType* q)

덱의 맨 뒤에 저장된 원소를 조회하는 함수이다.

⑨ element delete_rear(DequeType* q)

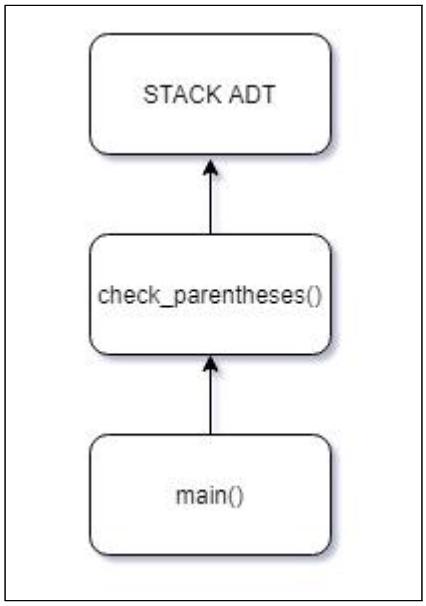
덱의 맨 뒤의 원소를 제거하는 함수이다. rear의 값을 하나 줄인다. (원형 덱이므로, %연산을 통해 index 오류가 없도록 조정한다.)

⑩ int get_size(DequeType * q)

덱에 담긴 원소의 개수를 알려준다.

**1번.**

**- 함수 구성 다이어그램**




**- 함수 설명**

① int check_parentheses()

괄호를 검사하는 함수이다. string의 원소들을 차례로 방문하면서, 여는 괄호들을 방문하게 되면, stack에 삽입하고, 닫는 괄호를 만나게 되면, 여는 괄호를 stack에서 꺼내어 pair가 맞는지 검사하게 된다. string의 문자들을 모두 방문하고도, stack에 원소가 남아있게 되면, 여는 괄호가 남는 상황이므로, 적절하지 않은 괄호가 된다.

input : char *str

output : int 0 또는 1

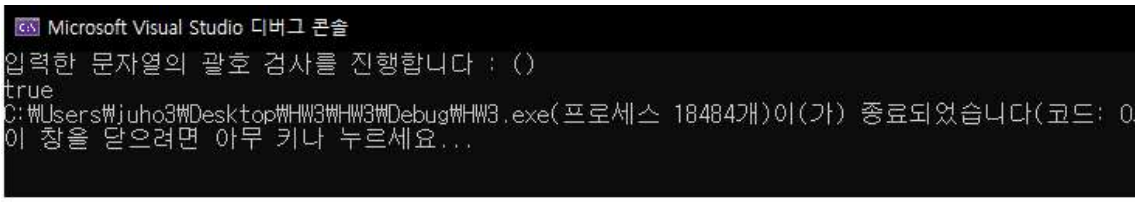
- 코드

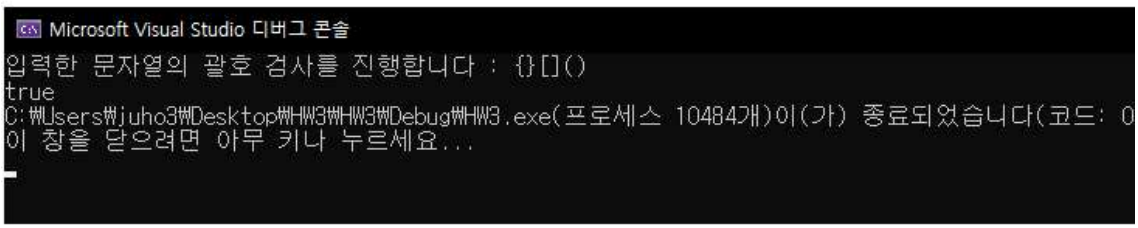
```
#include <string.h >
#define MAX_STRING_SIZE 21
#define ARR_SIZE 4
//==================stack ADT 구현==================//
#include <stdio.h >
#include <stdlib.h >
#define MAX_STACK_SIZE 50

typedef char element;
typedef struct {
        element data[MAX_STACK_SIZE];
        int top;
}StackType;
// 스택 초기화 함수
void init_stack(StackType * s) {
        s->top =-1;
}
// 공백 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)) {
                printf("스택이 포화상태 입니다.");
                return;
        }
        else {
                s->data[++(s ->top)] = item;
        }
}
// 삭제 함수
element pop(StackType * s) {
        if (is_empty(s)) {
                printf("스택이 비어있습니다.");
                exit(1);
        }
        else {
                return s ->data[(s ->top)--];
        }
}
// top원소 조회 함수
element peek(StackType* s) {
        if (is_empty(s)) {
                printf("스택이 비어있습니다.");
                exit(1);
        }
        else {
```

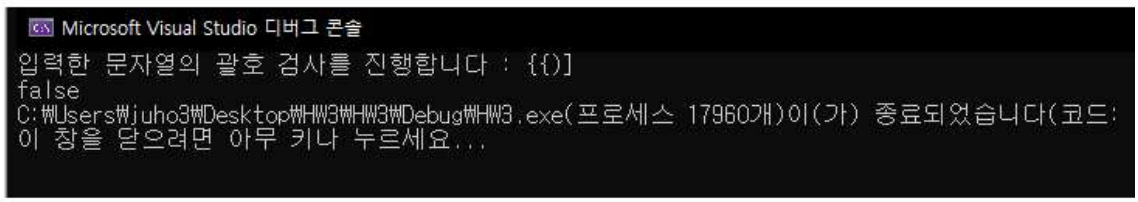
```
                return s ->data[s ->top];
        }
}
int get_size(StackType * s) {
        return (s ->top) +1;
}
////////////////////////////////////////////////////
// 전역 배열로 선언
char opening[ARR_SIZE] = {'(', '[', '{'};
char closing[ARR_SIZE] = {')', ']', '}'};

int check_parentheses(char *str){
        StackType s;
        init_stack(&s);

        char ch, open_ch;
        int n = strlen(str);

        for (int i =0; i < n; i ++) {
                ch = str[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_ch =='{'&& ch !='}')) {
                                        return 0;
                                }
                        }
                        break;
                default:
                        printf("err : 괄호가 아닌 문자가 입력되었습니다.");
                        exit(1);
                }
        }
        if (!is_empty(&s)) {
                return 0;
        }
        return 1;
}

int main() {
        char str[MAX_STRING_SIZE];
        printf("입력한 문자열의 괄호 검사를 진행합니다 : ");
        if (scanf("%s", str) ==-1) {
                return 0;
        }
        if (check_parentheses(str)) {
```

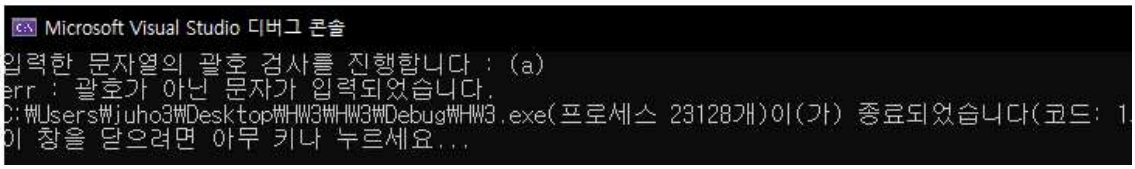
```
                printf("true");
        }
        else {
                printf("false");
        }
        return 0;
}
```

- 실행 결과



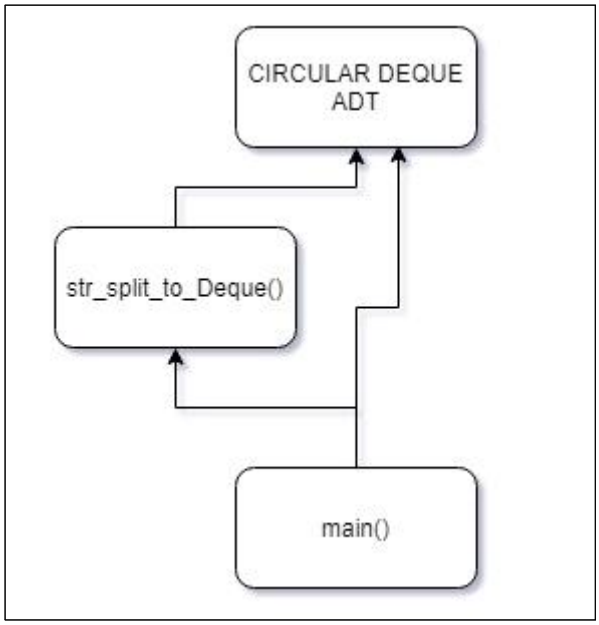



2번.

- 함수 구성 다이어그램




- 함수 설명

① void rotate(DequeType * q)

덱에 저장되어 있는 원소들을 회전시키는 함수이다. 예를 들어, 덱에 원소가 1, 2, 3, 4 순서로 저장되어 있다면, rotate(덱)을 실행한 결과는 2, 3, 4, 1이 된다. 인쇄열의 맨 앞의 것을 인쇄하되, 인쇄 대기 목록에 있는 다른 원소들 중에 중요도가 더 높은 것이 있다면, 맨 뒤로 보내기(rotate) 때문에 이 함수를 잘 활용해볼 수 있다. 회전이라 함은 다름아닌, delete_front()와 add_rear()를 차례로 실행하는 것이다.

input : DequeType * q

return : 없음

② int str_split_to_Deque()

입력이 “1 -2 3 4 51”처럼 띄어쓰기하여 들어오기 때문에, 각 정수에 접근하기 위해서는 입력을 공백(띄어쓰기) 기준으로 parsing하는 작업이 필요하다. 함수의 인자로 해당 string이 들어오게 되면, 이를 잘 split하여 정수 각각을 deque의 각 원소로 저장시키는 함수이다. 덱에 저장된 원소의 개수를 return값으로 주도록 하여, 이를 main함수에서 활용할 수 있도록 하였다.

input : DequeType * dq, int max_arr_size, char * str

return : deque의 사이즈 get_size(dq)

- 코드

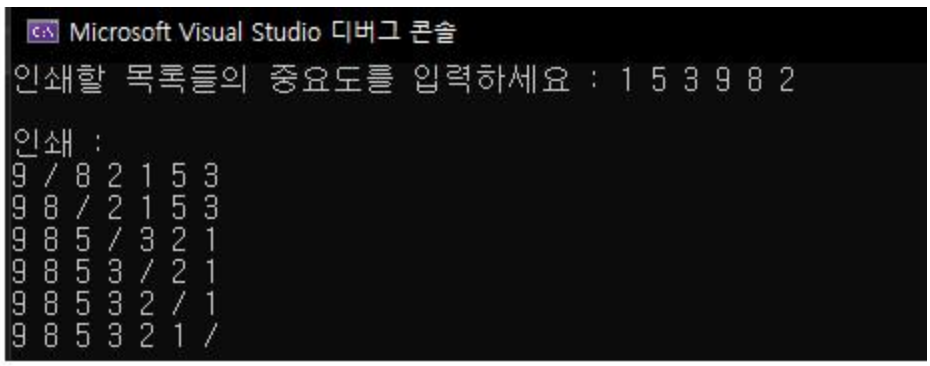
```
#include <string.h >
//======================원형 deque ADT 구현
==================================
#include <stdio.h >
#include <stdlib.h >
#define MAX_QUEUE_SIZE 300

/*원형 deque을 구현한다.*/
typedef int element;
typedef struct {
        element data[MAX_QUEUE_SIZE];
        int front, rear;
}DequeType;
// 오류 출력 함수
void error(const char * message) {
        fprintf(stderr, "%s₩n", message);
        exit(1);
}
// 초기화
void init_deque(DequeType * q) {
        q->front = q ->rear =0;
}
// 공백 상태 검출 함수
int is_empty(DequeType * q) {
        return (q ->front == q ->rear);
}
// 포화 상태 검출 함수
int is_full(DequeType * q) {
        return ((q ->rear +1) % MAX_QUEUE_SIZE == q ->front);
}
// 앞 삽입
void add_front(DequeType * q, element item) {
        if (is_full(q)) {
                error("큐가 포화상태입니다.");
        }
        q->data[q ->front] = item;
        q->front = (q ->front -1 + MAX_QUEUE_SIZE) % MAX_QUEUE_SIZE;
}
// 앞 원소 조회
element get_front(DequeType* q) {
        if (is_empty(q)) {
                error("큐가 공백상태입니다.");
        }
        return q ->data[(q ->front +1) % MAX_QUEUE_SIZE];
}
// 앞 원소 삭제
element delete_front(DequeType* q) {
        if (is_empty(q)) {
                error("큐가 공백상태입니다.");
        }
        q->front = (q ->front +1) % MAX_QUEUE_SIZE;
        return q ->data[q ->front];
```

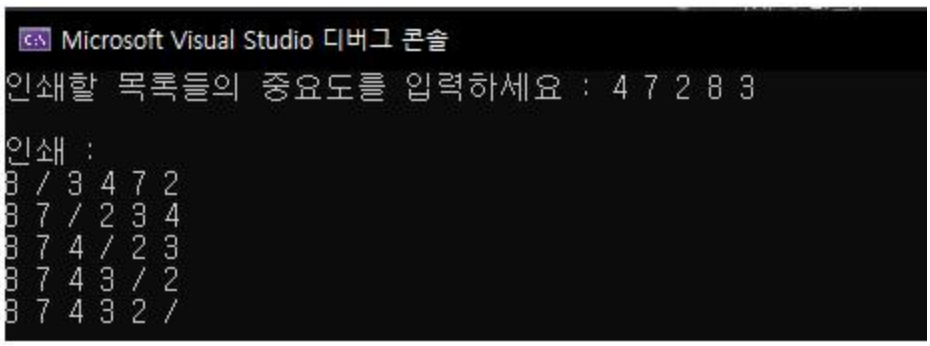
```
}
// 뒤 삽입
void add_rear(DequeType * q, element item) {
        if (is_full(q)) {
                error("큐가 포화상태입니다.");
        }
        q->rear = (q ->rear +1) % MAX_QUEUE_SIZE;
        q->data[q ->rear] = item;
}
// 뒤 조회
element get_rear(DequeType* q) {
        if (is_empty(q)) {
                error("큐가 공백상태 입니다.");
        }
        return q ->data[q ->rear];
}
// 뒤 삭제
element delete_rear(DequeType* q) {
        int pre = q ->rear;
        if (is_empty(q)) {
                error("큐가 공백 상태입니다.");
        }
        q->rear = (q ->rear -1 + MAX_QUEUE_SIZE) % MAX_QUEUE_SIZE;
        return q ->data[pre];
}
// 회전 함수(앞의 원소를 맨 뒤로 보내는 함수)
void rotate(DequeType * q) {
        element item = delete_front(q);
        add_rear(q, item);
}
// deque에 저장된 원소의 갯수를 알려주는 함수
int get_size(DequeType * q) {
        return (q ->rear - q ->front + MAX_QUEUE_SIZE) % MAX_QUEUE_SIZE;
}
//=================================================================
=====

#define MAX_STR_SIZE 300  // 문자열 최대 크기(공백 포함)
int str_split_to_Deque(DequeType * dq, int max_arr_size, char * str) {
        /*공백을 포함한 str을 입력받아 공백 기준 split하여 원형Deque에 저장하는 함
수이다.*/
        int * start = dq ->data;
        int num =0;
        // 숫자 분리 상태를 기억 (0이면 진행 안됨, -1이면 음수 진행, 1이면 양수 진
행)
        int state =0;
        while (*str ==' ') str ++;;  // 앞쪽 공백 제거
        for (; *str; str ++) {
                if (*str !=' ') {
                        if (state ==0) {  // 숫자 분리가 진행되지 않았다면,
                                if (*str =='-') {
                                        state =-1;      // 음수라는 표시,
                                        str++;
                                }
```

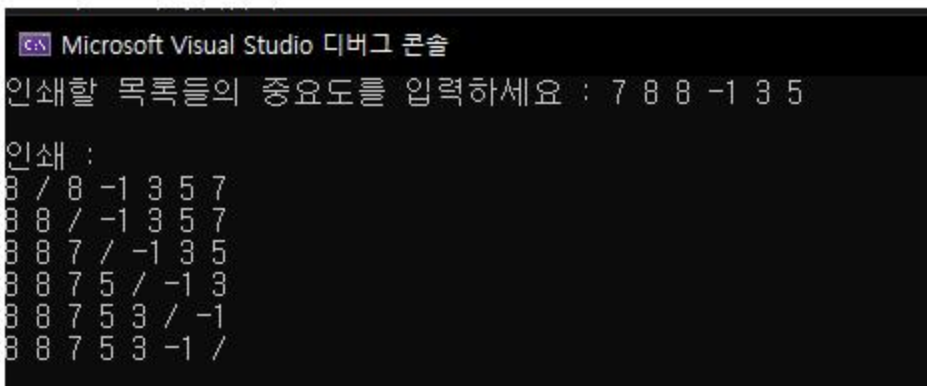
```
                                else state =1;  // 양수라는 표시
                        }
                        // 문자를 숫자로 변경
                        num = num *10 +*str -'0';
                }
                else {
                        if ((dq ->data - start) < max_arr_size) {
                                add_rear(dq, num * state);
                                num =0;
                                state =0;
                                // 공백 문자 제거
                                while (*(str +1) ==' ') str ++;
                        }
                        else return get_size(dq);
                }
        }
        // 숫자를 분리중에 반복문이 중단되었으면 해당 숫자를 deque에 추가한다.
        if (state) add_rear(dq, num * state);
        return get_size(dq);
}

int main() {
        char str[MAX_STR_SIZE];
        int size;
        int max_val;
        int printed_num;
        DequeType wait_to_print;
        init_deque(&wait_to_print);
        DequeType printed;
        init_deque(&printed);
        printf("인쇄할 목록들의 중요도를 입력하세요 : ");
        gets_s(str, MAX_STR_SIZE); // 입력을 문자열로 받는다.
        size = str_split_to_Deque(&wait_to_print, MAX_QUEUE_SIZE, str); // 문자열
입력을 정수 배열 wait_to_print에 저장
        printf("₩n인쇄 :₩n");
        while (!is_empty(&wait_to_print)) {
                while (1) {
                        max_val = get_front(&wait_to_print);
                        for (int i = wait_to_print.front +2; i <= wait_to_print.rear; i
++) {
                                if (wait_to_print.data[i] > max_val) {
                                        max_val = wait_to_print.data[i];
                                }
                        }
                        if (max_val == get_front(&wait_to_print)) {
                                break;
                        }
                        else {
                                rotate(&wait_to_print);
                        }
                }
                printed_num = delete_front(&wait_to_print);
                add_rear(&printed, printed_num);
                for (int i = printed.front +1; i <= printed.rear; i ++) {
```

```
                        printf("%d ", printed.data[i]);
                }
                printf("/ ");
                for (int i = wait_to_print.front +1; i <= wait_to_print.rear; i ++) {
                        printf("%d ", wait_to_print.data[i]);
                }
                printf("₩n");
        }
        return 0;
}
```

- 실행 결과

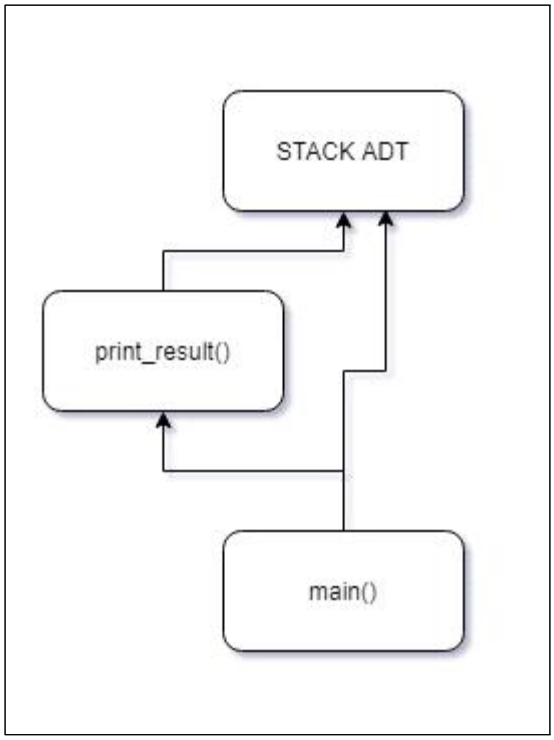


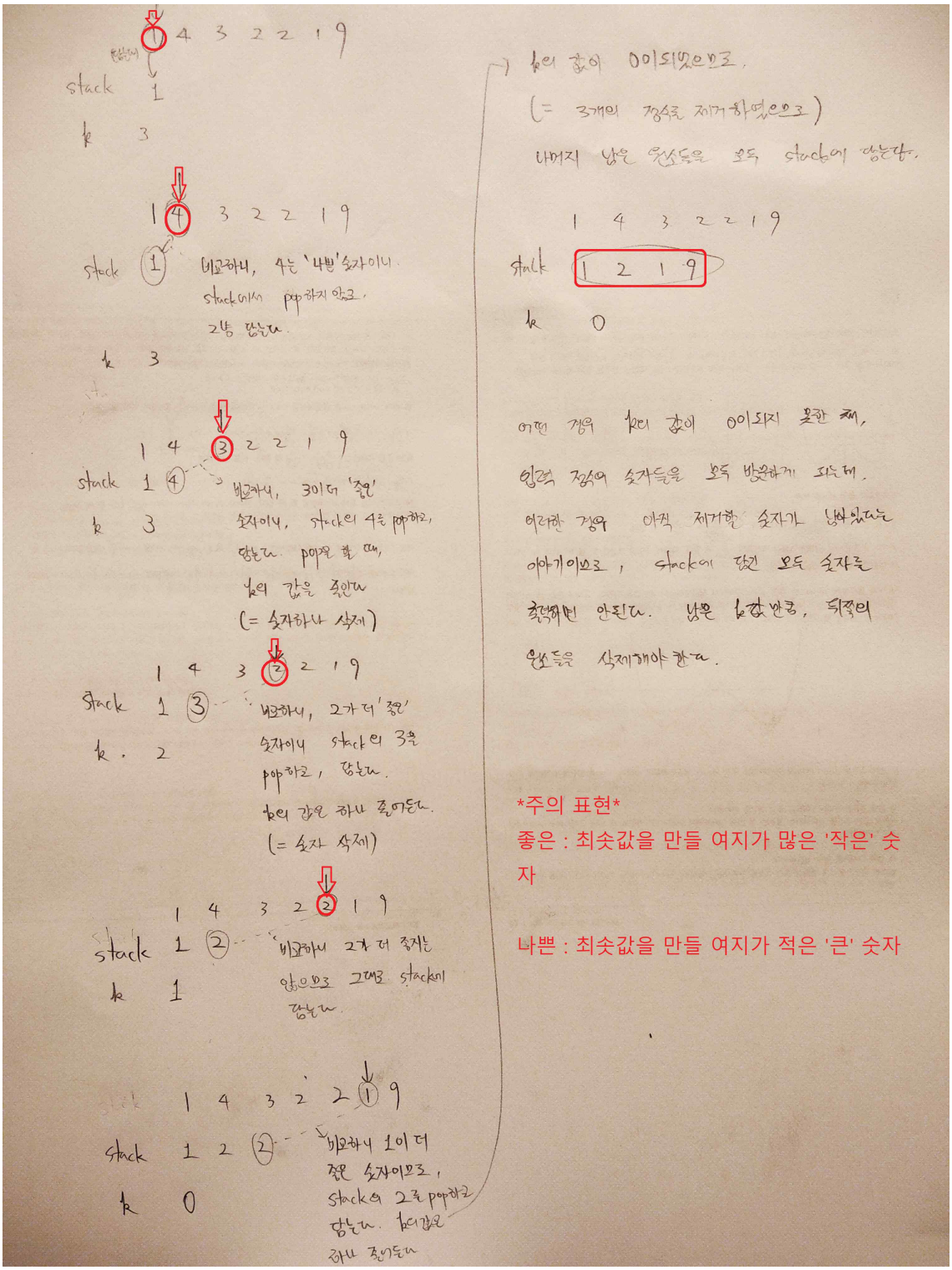
3번

- 함수 구성 다이어그램




- 로직 설명

입력으로 들어온 정수가 양수인 상황을 예로 들어 설명한다. 입력으로 들어온 원소를 앞에서부터 차례로 방문하면서 좋은 (최솟값을 만들어주는 작은 숫자) 숫자가 나오면, stack에 담겨 있는 나쁜 (방문한 숫자보다 커서 최솟값을 만들기에 적절하지 않은 숫자) 숫자를 pop하고 방문한 원소를 push하는 방식으로 로직을 작성한다. 예를 들어, 1432219 라는 정수에서 3개의 원소를 제거하여 최솟값을 만든다고 한다면, 답은 4, 3, 2를 지운 1219가 답이 되는데, 이를 위에서 설명한 대로, logic을 따라가본다.




만약, 입력 정수가 음수인 경우라면, - 부호를 제거하고 최댓값을 구한다고 생각하면서, logic을 짜면 된다. 더 '좋은'의 숫자의 기준이 '큰' 숫자가 되면 된다. 단순히 부등호의 방향만 바뀌면 되는데, 이를 state 변수로 조정하였다.

- **함수 설명**

① void print_result()

Step 1으로 (답에 근접한) stack을 얻어내었다면, 이를 원하는 만큼, 원하는 형태대로 출력해내는 함수가 필요하다. 답에 근접하다는 표현을 한 이유는 stack의 원소들을 모두 붙여서 출력하면, 답이 안되는 경우가 있기 때문이다. k가 0까지 줄어들지 못한 채로 입력 정수의 숫자들을 모두 방문하게 되었다면, stack에 있는 원소를 모두 출력하면 안되고, end = get_size(st) - k로 출력의 끝 지점을 조정해야 한다. 그리고 음수인 경우, 맨 앞에 - 기호를 붙여 출력해야 한다. 음수인지 양수인지에 따라 출력의 형태가 달라지기 때문에, 적절한 조건문에 진입할 수 있도록 flag 변수를 활용하였다. 또한, 10200 같은 숫자에서 1을 제거하게 되면, 답이 0200이 되는데, 맨 앞의 0을 출력하지 않기를 원하므로, 이러한 관리도 해주어야 한다.

input : char * str, StackType * st, int k

output : 없음

- **코드**

```
#include <string.h >
//==================stack ADT 구현====================//
#include <stdio.h >
#include <stdlib.h >
#define MAX_STACK_SIZE 50

typedef char element;
typedef struct {
        element data[MAX_STACK_SIZE];
        int top;
}StackType;
// 스택 초기화 함수
void init_stack(StackType * s) {
        s->top =-1;
}
// 공백 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)) {
                printf("스택이 포화상태 입니다.");
                return;
        }
        else {
```

```
                s->data[++(s ->top)] = item;
        }
}
// 삭제 함수
element pop(StackType * s) {
        if (is_empty(s)) {
                printf("스택이 비어있습니다.");
                exit(1);
        }
        else {
                return s ->data[(s ->top)--];
        }
}
// top원소 조회 함수
element peek(StackType* s) {
        if (is_empty(s)) {
                printf("스택이 비어있습니다.");
                exit(1);
        }
        else {
                return s ->data[s ->top];
        }
}
int get_size(StackType * s) {
        return (s ->top) +1;
}
////////////////////////////////////////////////

void print_result(char * str, StackType * st, int k) {
    int size = strlen(str);
        int flag;
        int end;
        int state =1;
        int start =0;
        int check =1;
        if (str[0] =='-') {
                state =-1;
                start =1;
        }
        for (int i = start; i <size; i ++) { // stack에 result를 담는다.
                // 음수가 입력으로 들어오는 상황을 고려하여, 부등호 방향을 state
변수로 조정한다.
                while (get_size(st) && k >0 && (state)*peek(st) > (state)*(str[i]
-'0')) {
                        pop(st);
                        k--;
                }
                push(st, str[i] -'0');
        }
        flag =0;
        end = get_size(st) - k; // k가 0이 아닌경우 stack에 담긴 원소 중 뒤쪽의 k
개를 지워야 함.
        if (state ==-1) {
                printf("-");
```

```
        }
        for (int i =0; i <end; i ++) { // print
                check =0;               // for문에 진입여부를 check하는 변수. for문에 진입하지 않으면 출력값은 0이다.
                if (st ->data[i] !=0) { // 출력이 0200이라면 맨 앞의 0을 출력하지 않고 200으로 출력하기 위함.
                        flag =1;
                }
                if (flag ==1) {
                        printf("%d", st ->data[i]);
                }
        }
        if (check) {
                printf("0");
        }
        return ;
}

int main() {
        char str[MAX_STACK_SIZE];
        int k;
        int size;
        int check =0;
        StackType st;
        init_stack(&st);
        printf("정수를 입력하세요 : ");
        if (scanf("%s", str) ==-1) {
                return 0;
        }
    str[MAX_STACK_SIZE -1] =0;
        size = strlen(str);
        printf("제거할 숫자의 갯수를 입력하세요. : ");
        if (scanf("%d", &k) ==-1) {
                return 0;
        }

        if (str[0] =='-') {
                check =1;
        }
        if (k >size - check || k <0) {
                printf("₩nerr: 제거할 숫자의 갯수가 잘못 입력되었습니다.₩n");
                exit(1);
        }
        print_result(str, &st, k);
        return 0;
}
```

- 실행 결과


Microsoft Visual Studio 디버그 콘솔
정수를 입력하세요 : 1432219
제거할 숫자의 갯수를 입력하세요. : 3
1219
C:₩Users₩juho3₩Desktop₩HW3₩HW3₩Debug₩HW3.exe(프로세스 26408개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
Microsoft Visual Studio 디버그 콘솔
정수를 입력하세요 : -1432219
제거할 숫자의 갯수를 입력하세요. : 3
-4329
C:₩Users₩juho3₩Desktop₩HW3₩HW3₩Debug₩HW3.exe(프로세스 8508개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
Microsoft Visual Studio 디버그 콘솔
정수를 입력하세요 : 10200
제거할 숫자의 갯수를 입력하세요. : 1
200
C:₩Users₩juho3₩Desktop₩HW3₩HW3₩Debug₩HW3.exe(프로세스 10812개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
Microsoft Visual Studio 디버그 콘솔
정수를 입력하세요 : 10
제거할 숫자의 갯수를 입력하세요. : 2
0
C:₩Users₩juho3₩Desktop₩HW3₩HW3₩Debug₩HW3.exe(프로세스 7164개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...