『제 24기 청학동』Tutoring 5주 주간 학습 보고서

학습활동 참여 점검			
팀명	2+3=5		
1회차 일시	2022.04.19 18:00 ~ 21:00	1회차 장소	Discord
1회차 참석자	신수빈,변장무,이준호,이동현,권재현		
1회차 불참자			
2회차 일시		2회차 장소	
2회차 참석자			
2회차 불참자			
3회차 일시		3회차 장소	
3회차 참석자			
3회차 불참자			
4회차 일시		4회차 장소	
4회차 참석자			
4회차 불참자			
5회차 일시		5회차 장소	
5회차 참석자			
5회차 불참자			
학습목표	1. 스택 // 2. 큐		
과제물	시험기간이므로 과제는 생략한다.		

『제 24기 청학동』Tutoring 5주 주간 학습 보고서

학습활동 참여 점검	
팀명	2+3=5

1. 스택(Stack)

스택은 쌓아놓은 더미라는 의미로 특수하게 변형하여 사용하는 리스트의 한 종류이다.
한쪽으로만 입력과 출력을 할 수 있는 구조로 먼저 입력된 것이 아래에 저장되고, 나중에 입력된 것이 위에 저장되는 후입선출(LIFO : Last-In First-Out)의 형태를 갖는다.

스택의 추상 데이터 타입 (ADT)

객체 : n개의 element형의 요소들의 선형리스트
연산
- create()
  - : 스택 s를 생성한다.
  - : 스택 선언 및 생성(배열 리스트 또는 연결 리스트)
- is_empty(s)
  - : 스택 s가 비어있는지를 검사한다.
  - : 배열로 스택 s를 구성한 경우, s의 length 검사
- is_full(s)
  - : 스택 s가 가득 찼는가를 검사한다.
  - : 배열로 s를 구성한 경우, s의 length 검사
- push(s, e)
  - : 스택 s에 데이터 e를 추가한다.
  - : s에 추가(배열 리스트 또는 연결 리스트)
- pop(s)
  - : 스택 s의 top에 있는 데이터를 삭제한다.
  - : s의 top 데이터를 삭제(배열 리스트 또는 연결 리스트)
- peek(s)
  - : 스택 s의 top에 있는 데이터를 삭제하지 않고 반환
  - : s의 top 데이터를 반환(배열 리스트 또는 연결 리스트)

배열로 구현하는 스택

구조체 선언 : 1차원 배열 stack[개수], top = -1

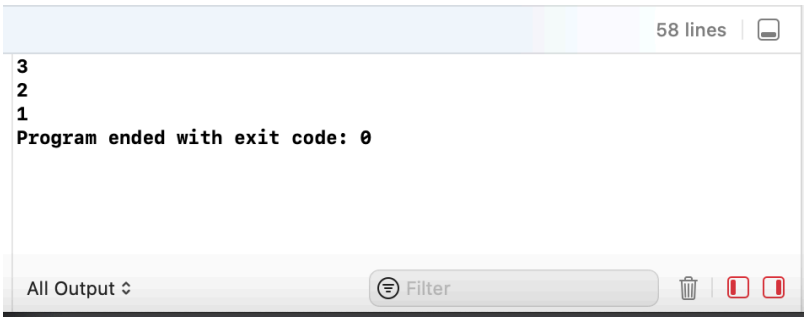
top 변수 : 최근에 입력된 자료의 첨자(top = 스택의 데이터 개수 - 1)

첫 데이터는 stack[0]에, 마지막 데이터는 stack[top]에 저장한다.

정수 배열 스택 프로그램

```
#include
#include

#define MAX_STACK_SIZE 100
typedefintelement;
element stack[MAX_STACK_SIZE];
int top = -1;

intis_empty()
{
    return (top == -1);
}

intis_full()
{
    return (top == (MAX_STACK_SIZE - 1));
}
```

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

elementpop()
{
    if (is_empty())
    {
        fprintf(stderr, "스택 공백 에러₩n");
        exit(1);
    }
    else return stack[top--];
}

elementpeek()
{
    if(is_empty())
    {
        fprintf(stderr, "스택 공백 에러₩n");
        exit(1);
    }
    elsereturnstack[top];
}

intmain(void)
{
    push(1);
    push(2);
    push(3);
    printf("%d₩n", pop());
    printf("%d₩n", pop());
    printf("%d₩n", pop());
    return 0;
}
```

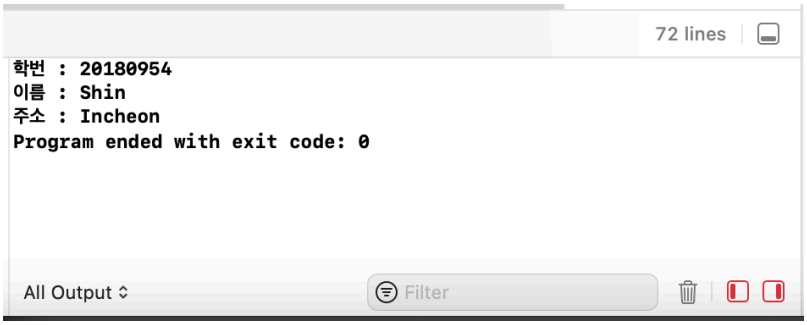


구조체 배열 스택 프로그램

```
#include

#include

#define MAX_STACK_SIZE 100

#define MAX_STRING 100


typedefstruct

{

    int student_no;

    char name[MAX_STRING];
```

```
    char address[MAX_STRING];
} element;

element stack[MAX_STACK_SIZE];
int top = -1;

intis_empty()
{
    return (top == -1);
}

intis_full()
{
    return (top == (MAX_STACK_SIZE - 1));
}

void push(element item)
{
    if (is_full())
    {
        fprintf(stderr, "스택 포화 에러₩n");
        return;
    }

    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];
}

intmain(void)
{
    element ie = {20180954, "Shin", "Incheon"};
    element oe;

    push(ie);
    oe = pop();

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



일반적인 배열 스택 프로그램

```
#include
#include

#define MAX_STACK_SIZE 100

typedefintelement;
typedefstruct
{
    element data[MAX_STACK_SIZE];
    int top;
} StackType;

voidinit_stack(StackType *s)
```

```
{
    s->top = -1;
}

intis_empty(StackType *s)
{
    return (s->top == -1);
}

intis_full(StackType *s)
{
    return (s->top == (MAX_STACK_SIZE - 1));
}

voidpush(StackType *s, element item)
{
    if (is_full(s))
    {
        fprintf(stderr, "스택 포화 에러₩n");
        return;
    }
    else s->data[++(s->top)] = item;
}

elementpop(StackType *s)
{
    if (is_empty(s))
    {
        fprintf(stderr, "스택 공백 에러₩n");
        exit(1);
    }
    else return s->data[(s->top)--];
}

elementpeek(StackType *s)
{
    if (is_empty(s))
    {
        fprintf(stderr, "스택 공백 에러₩n");
        exit(1);
    }
    else return s->data[s->top];
}

intmain(void)
{
    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));
}
```

동적 스택 프로그램

```c
#include
#include

#define MAX_STACK_SIZE 100

typedefintelement;
typedefstruct
{
    element data[MAX_STACK_SIZE];
    int top;
} StackType;

voidinit_stack(StackType *s)
{
    s->top = -1;
}

intis_empty(StackType *s)
{
    return (s->top == -1);
}

intis_full(StackType *s)
{
    return (s->top == (MAX_STACK_SIZE - 1));
}

voidpush(StackType *s, element item)
{
    if (is_full(s))
    {
        fprintf(stderr, "스택 포화 에러₩n");
        return;
    }
    else s->data[++(s->top)] = item;
}

elementpop(StackType *s)
{
    if (is_empty(s)) {
        fprintf(stderr, "스택 공백 에러₩n");
        exit(1);
    }
    else return s->data[(s->top)--];
}

elementpeek(StackType *s)
{
    if (is_empty(s)) {
        fprintf(stderr, "스택 공백 에러₩n");
        exit(1);
    }
    else return s->data[s->top];
}

intmain(void)
{
    StackType *s;

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

    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);
}
```

동적 배열 스택 프로그램

```
#include
#include
#define MAX_STACK_SIZE 100

typedefintelement;
typedefstruct
{
    element *data;
    int capacity;
    int top;
} StackType;

voidinit_stack(StackType *s)
{
    s->top = -1;
    s->capacity = 1;
    s->data = (element *)malloc(s->capacity * sizeof(element));
}

intis_empty(StackType *s)
{
    return (s->top == -1);
}

intis_full(StackType *s)
{
    return (s->top == (MAX_STACK_SIZE - 1));
}

voidpush(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;
}

elementpop(StackType *s)
{
    if (is_empty(s)) {
        fprintf(stderr, "스택 공백 에러₩n");
        exit(1);
    }
    else return s->data[(s->top)--];
}

intmain(void)
{
    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;
}
```

연결 리스트로 구현하는 스택

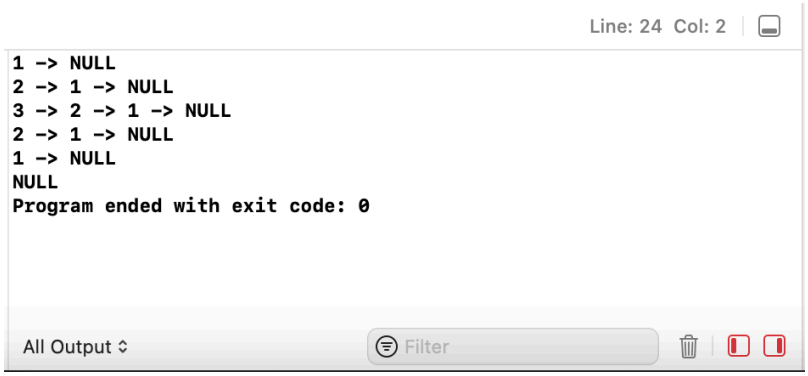
장점 : 배열 구조와 반대로 스택 크기가 제한되지 않는다.
단점 : 구현 과정이 더 복잡하고, push/pop 처리 시간이 더 걸린다.

연결된 스택 프로그램

```
#include

#include


typedefintelement;

typedefstructStackNode

{

    element data;

    struct StackNode *link;

} StackNode;


typedefstruct

{

    StackNode *top;

} LinkedStackType;


void init(LinkedStackType *s)

{

    s -> top = NULL;

}


int is_empty(LinkedStackType *s)

{

    return (s -> top == NULL);
```

```
}

int is_full(LinkedStackType *s)
{
    return 0;
}

void push(LinkedStackType *s, element item)
{
    StackNode *tmp = (StackNode *)malloc(sizeof(StackNode));
    tmp -> data = item;
    tmp -> link = s -> top;
    s -> top = tmp;
}

void print_stack(LinkedStackType *s)
{
    for(StackNode *p=s->top;p!=NULL;p=p->link)
        printf("%d -> ", p->data);
    printf("NULL ₩n");
}

element pop(LinkedStackType *s)
{
    if(is_empty(s))
    {
        fprintf(stderr, "스택이 비어있음 ₩n");
        exit(1);
    }

    else
    {
        StackNode *tmp = s -> top;
        int data = tmp -> data;
        s -> top = s -> top -> link;
        free(tmp);

        return data;
    }
```

```
}

element peek(LinkedStackType *s)
{
    if(is_empty(s))
    {
        fprintf(stderr, "스택이 비어있음₩n");
        exit(1);
    }
    else
    {
        return s -> top -> data;
    }
}

intmain(void)
{
    LinkedStackType s;
    init(&s);

    push(&s, 1);
    print_stack(&s);
    push(&s, 2);
    print_stack(&s);
    push(&s, 3);
    print_stack(&s);
    pop(&s);
    print_stack(&s);
    pop(&s);
    print_stack(&s);
    pop(&s);
    print_stack(&s);

    return 0;
}
```



괄호 매칭 알고리즘

<괄호 구성/검사 조건>

1. 왼쪽 괄호의 개수와 오른쪽 괄호의 개수가 같아야 한다.
2. 같은 괄호에서 왼쪽 괄호는 오른쪽 괄호보다 먼저 나와야 한다.
3. 괄호 사이에는 포함 관계만 존재한다.

문자열에 있는 괄호를 차례대로 조사하면서 왼쪽 괄호를 만나면 스택에 삽입하고, 오른쪽 괄호를 만나면 스택에서 top 괄호를 삭제한 후 오른쪽 괄호와 짝이 맞는지를 검사한다.
이때 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

```
#include

#include

#include

#define MAX_STACK_SIZE 100


typedefcharelement;

typedefstruct

{

    element data[MAX_STACK_SIZE];

    int top;

} StackType;


voidinit_stack(StackType *s)

{

    s->top = -1;

}


intis_empty(StackType *s)

{

    return (s->top == -1);

}
```

```
intis_full(StackType *s)
{
    return (s->top == (MAX_STACK_SIZE - 1));
}

voidpush(StackType *s, element item)
{
    if (is_full(s))
    {
        fprintf(stderr, "스택 포화 에러₩n");
        return;
    }
    else s->data[++(s->top)] = item;
}

elementpop(StackType *s)
{
    if (is_empty(s))
    {
        fprintf(stderr, "스택 공백 에러₩n");
        exit(1);
    }
    else return s->data[(s->top)--];
}

elementpeek(StackType *s)
{
    if (is_empty(s))
    {
        fprintf(stderr, "스택 공백 에러₩n");
        exit(1);
    }
    else return s->data[s->top];
}

intcheck_matching(constchar *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_ch == '{' && ch != '}'))
                    {
                        return 0;
                    }
                    break;
                }
        }
    }
    if (!is_empty(&s)) return 0;
    return 1;
}

intmain(void)
{
    char *p = "{ A[(i+1)]=0; }";
    if (check_matching(p) == 1)
        printf("%s 괄호 검사 성공₩n", p);
    else
        printf("%s 괄호 검사 실패₩n", p);
    return 0;
```

```
}
```

수식의 표기 방법 : 전위(Prefix), 중위(Infix), 후위(Postfix)

전위 표기법	중위 표기법	후위 표기법
+2*34	2+3*4	234*+
*ab+5	a*b+5	ab*5+
+12+7	(1+2)+7	12+7+

사람

수식을 중위 표기식으로 표현한다.
연산자 우선순위를 사용하므로 괄호를 생략하기도 한다.

컴퓨터

스택을 활용하여 수식을 후위 표기식으로 변환하여 표현한다.
후위 표기식에서의 연산자 우선순위에 따라 계산을 진행한다.

후위 표기식의 계산 과정

1. 입력된 후위 표기식을 왼쪽부터 스캔한다.
2. 피연산자를 만나면 무조건 스택에 push한다
3. 연산자를 만나면 연산자 종류에 따라 스택에 저장된 피연산자를 pop하고, 계산한 수 다시 스택에 push한다.
4. 후위 표기식 스캔을 완료했다면 스택에 유일하게 남은 결과를 pop하여 리턴해준다.

자연어 알고리즘

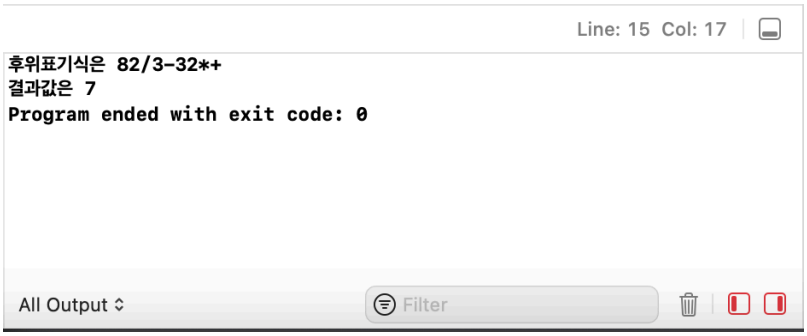
피연산자를 만나면 그대로 출력한다.
연산자를 만나면 스택에 저장한다.
(스택 top에 저장되어 있는 연산자보다 우선순위가 낮은 연산자가 나오면 pop, 우선순위가 높은 연산자가 나오면 push)
왼쪽 괄호는 우선순위가 가장 낮은 연산자로 취급한다.
오른쪽 괄호가 나오면 스택에서 왼쪽 괄호 다음 순서로 쌓여있는 모든 연산자를 pop한다.
입력된 중위 표기식이 끝날 때까지 위 과정을 반복한다.

후위 표기 수식 계산 프로그램

```
#include
#include
#include
#define MAX_STACK_SIZE 100

typedefcharelement;
typedefstruct
{
    element data[MAX_STACK_SIZE];
    int top;
} StackType;
```

```
voidinit_stack(StackType *s)
{
    s->top = -1;
}

intis_empty(StackType *s)
{
    return (s->top == -1);
}

intis_full(StackType *s)
{
    return (s->top == (MAX_STACK_SIZE - 1));
}

voidpush(StackType *s, element item)
{
    if (is_full(s))
    {
        fprintf(stderr, "스택 포화 에러₩n");
        return;
    }
    else s->data[++(s->top)] = item;
}

elementpop(StackType *s)
{
    if (is_empty(s))
    {
        fprintf(stderr, "스택 공백 에러₩n");
        exit(1);
    }
    else return s->data[(s->top)--];
}

elementpeek(StackType *s)
{
    if (is_empty(s))
    {
        fprintf(stderr, "스택 공백 에러₩n");
        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;

    init_stack(&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);
}

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

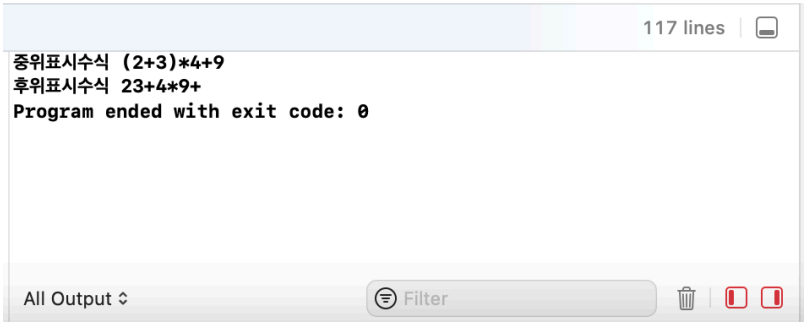


중위 표기 수식을 후위 표기 수식으로 변환하는 프로그램

```
#include
#include
#include
#define MAX_STACK_SIZE 100

typedefcharelement;
typedefstruct
{
    element data[MAX_STACK_SIZE];
    int top;
} StackType;

voidinit_stack(StackType *s)
{
    s->top = -1;
}

intis_empty(StackType *s)
{
    return (s->top == -1);
}

intis_full(StackType *s)
{
    return (s->top == (MAX_STACK_SIZE - 1));
}

voidpush(StackType *s, element item)
{
    if (is_full(s))
    {
        fprintf(stderr, "스택 포화 에러");
        return;
    }
    else s->data[++(s->top)] = item;
```

```
}

elementpop(StackType *s)
{
    if (is_empty(s))
    {
        fprintf(stderr, "스택 공백 에러₩n");
        exit(1);
    }
    else return s->data[(s->top)--];
}

elementpeek(StackType *s)
{
    if (is_empty(s))
    {
        fprintf(stderr, "스택 공백 에러₩n");
        exit(1);
    }
    else return s->data[s->top];
}

intprec(char op)
{
    switch (op)
    {
        case '(': case ')': return 0;
        case '+': case '-': return 1;
        case '*': case '/': return 2;
    }
    return -1;
}

voidinfix_to_postfix(char exp[])
{
    int i = 0;
    char ch, top_op;
    int len = strlen(exp);
    StackType s;

    init_stack(&s);                    // Ω∫√ √ ±,»≠
    for (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));
}

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



미로탐색 프로그램

```
#include
#include
#include
#define MAX_STACK_SIZE 100
#define MAZE_SIZE 6

typedefstruct
{
    short r;
    short c;
} element;

typedefstruct
{
    element data[MAX_STACK_SIZE];
    int top;
} StackType;

voidinit_stack(StackType *s)
{
    s->top = -1;
}

intis_empty(StackType *s)
{
    return (s->top == -1);
}

intis_full(StackType *s)
{
    return (s->top == (MAX_STACK_SIZE - 1));
}

voidpush(StackType *s, element item)
{
    if (is_full(s))
    {
        fprintf(stderr, "스택 포화 에러₩n");
        return;
    }
    else s->data[++(s->top)] = item;
}
```

학습 내용

```
elementpop(StackType *s)
{
    if (is_empty(s))
    {
        fprintf(stderr, "스택 공백 에러₩n");
        exit(1);
    }
    else return s->data[(s->top)--];
}

elementpeek(StackType *s)
{
    if (is_empty(s))
    {
        fprintf(stderr, "스택 공백 에러₩n");
        exit(1);
    }
    else return s->data[s->top];
}

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' },};

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);
    }
}

voidmaze_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");
    }
}

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

```
111111
.01001
100011
101011
10100x
111111

111111
..1001
100011
101011
10100x
111111

111111
..1001
1.0011
101011
10100x
111111

111111
..1001
1..011
101011
10100x
111111

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

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

111111
..1001
1...11
101.11
101.0x
111111
```

```
111111
..1001
1...11
101.11
101..x
111111
성공
Program ended with exit code: 0
All Output ⌃  Filter
```

2. 큐(Queue)

큐는 먼저 들어온 데이터가 먼저 나가는 선입선출(FIFO : First-In First-Out) 형태의 자료구조이다.

큐 ADT

객체 : n개의 element형으로 구성된 요소들의 순서있는 모임
연산
create() : 큐를 생성한다.
init(q) : 큐를 초기화한다.
is_empty(q) : 큐가 비어있는지를 검사한다.
is_full(q) : 큐가 가득 찼는가를 검사한다.
enqueue(q, e) : 큐의 rear에 요소를 추가한다.
dequeue(q) : 큐의 front에 있는 요소를 반환한 다음 삭제한다.
peek(q) : 큐에서 삭제하지 않고, front의 요소를 반환한다.

큐의 응용

직접적인 응용
순서화 시뮬레이션의 대기열
ex) 공항에서의 비행기들, 은행에서의 대기열 등...
통신에서 데이터 패킷들의 전송패턴 모델링
프린터와 컴퓨터 사이의 버퍼링

간접적인 응용
스택과 마찬가지로 프로그래머의 자료 저장/검색/조회 등의 도구이다.
많은 알고리즘에서 일반적 형태 또는 응용 형태로 사용한다.

배열을 이용한 큐

선형 큐 (Linear Queue)
1차원 배열로 큐를 구현하는 경우이다.
삽입/삭제 동작이 빈번하면 front와 rear값이 배열의 뒤쪽으로 이동한다.
앞부분의 빈 공간을 활용하려면 큐의 값들을 모두 앞부분으로 이동시켜야 한다.
-> 불편하고 재구성 시간이 소모된다.
-> 선형 큐는 많이 사용되지 않는다.

선형 큐 프로그램

```
#include

#include

#define MAX_QUEUE_SIZE 5


typedefintelement;

typedefstruct

{

    int  front;

    int rear;

    element  data[MAX_QUEUE_SIZE];

} QueueType;


void error(char *message)

{

    fprintf(stderr, "%s₩n", message);

    exit(1);

}
```

```
voidinit_queue(QueueType *q)
{
    q->rear = -1;
    q->front = -1;
}

voidqueue_print(QueueType *q)
{
    for (int i = 0; i<MAX_QUEUE_SIZE; i++)
    {
        if (i <= q->front || i> q->rear)
            printf("    | ");
        else
            printf("%d | ", q->data[i]);
    }
    printf("₩n");
}

intis_full(QueueType *q)
{
    if (q->rear == MAX_QUEUE_SIZE - 1)
        return 1;
    else
        return 0;
}

intis_empty(QueueType *q)
{
    if (q->front == q->rear)
        return 1;
    else
        return 0;
}

void enqueue(QueueType *q, int item)
{
    if (is_full(q))
    {
        error("큐가 포화상태 입니다.");
```

```
        return;
    }
    q->data[++(q->rear)] = item;
}

intdequeue(QueueType *q)
{
    if (is_empty(q))
    {
        error("큐가 공백상태 입니다.");
        return -1;
    }
    int item = q->data[++(q->front)];
    return item;
}

intmain(void)
{
    int item = 0;
    QueueType q;

    init_queue(&q);

    enqueue(&q, 10); queue_print(&q);
    enqueue(&q, 20); queue_print(&q);
    enqueue(&q, 30); queue_print(&q);

    item = dequeue(&q); queue_print(&q);
    item = dequeue(&q); queue_print(&q);
    item = dequeue(&q); queue_print(&q);
    return 0;
}
```

```
Line: 25  Col: 31
10 |    |    |    |    |
10 | 20 |    |    |    |
10 | 20 | 30 |    |    |
   | 20 | 30 |    |    |
   |    | 30 |    |    |
   |    |    |    |    |
Program ended with exit code: 0

All Output ⌃                Filter
```

선형 큐는 구현이 쉬우나, 계속 사용하려면 큐의 값을 이동시키는 시간이 필요하기 때문에 불편하고 시간이 걸려 많이 사용되지 않으며, 이동시간이 없는 원형 큐로 대체된다.

원형 큐 (Circular Queue)

1차원 배열을 원형으로 사용하여 큐를 구현한다.
배열을 원형 구조로 활용하도록 front와 rear 값의 계산 방식을 변형한다.
재구성 시간이 필요없다.
-> 일반적인 큐 사용 패턴

원형 큐 프로그램

```
#include
#include

#define MAX_QUEUE_SIZE 5

typedefintelement;
typedefstruct
{
    element  data[MAX_QUEUE_SIZE];
    int  front, rear;
} QueueType;

void error(char *message)
{
    fprintf(stderr, "%s₩n", message);
    exit(1);
}

voidinit_queue(QueueType *q)
{
    q->front = q->rear = 0;
}

intis_empty(QueueType *q)
{
    return (q->front == q->rear);
}

intis_full(QueueType *q)
{
    return ((q->rear + 1) % MAX_QUEUE_SIZE == q->front);
}
```

```
voidqueue_print(QueueType *q)
{
    printf("QUEUE(front=%d rear=%d) = ", q->front, q->rear);
    if (!is_empty(q))
    {
        int i = q->front;
        do
        {
            i = (i + 1) % (MAX_QUEUE_SIZE);
            printf("%d | ", q->data[i]);
            if (i == q->rear)
                break;
        } while (i != q->front);
    }
    printf("₩n");
}

voidenqueue(QueueType *q, element item)
{
    if (is_full(q))
        error("큐가 포화상태 입니다.");
    q->rear = (q->rear + 1) % MAX_QUEUE_SIZE;
    q->data[q->rear] = item;
}

elementdequeue(QueueType *q)
{
    if (is_empty(q))
        error("큐가 공백상태 입니다.");
    q->front = (q->front + 1) % MAX_QUEUE_SIZE;
    return q->data[q->front];
}

elementpeek(QueueType *q)
{
    if (is_empty(q))
        error("큐가 공백상태 입니다.");
    return q->data[(q->front + 1) % MAX_QUEUE_SIZE];
```

```
}

intmain(void)
{
    QueueType queue;
    int element;

    init_queue(&queue);
    printf("-- 데이터 추가 단계 --₩n");
    while (!is_full(&queue))
    {
        printf("정수를 입력하세요 : ");
        scanf("%d", &element);
        enqueue(&queue, element);
        queue_print(&queue);
    }
    printf("큐는 포화상태 입니다.₩n₩n");

    printf("-- 데이터 삭제 단계 --₩n");
    while (!is_empty(&queue))
    {
        element = dequeue(&queue);
        printf("꺼내진 정수 : %d ₩n", element);
        queue_print(&queue);
    }
    printf("큐는 공백상태 입니다.₩n");
    return 0;
}
```

```
Line: 32  Col: 2
-- 데이터 추가 단계 --
정수를 입력하세요 : 10
QUEUE(front=0 rear=1) = 10 |
정수를 입력하세요 : 20
QUEUE(front=0 rear=2) = 10 | 20 |
정수를 입력하세요 : 30
QUEUE(front=0 rear=3) = 10 | 20 | 30 |
정수를 입력하세요 : 40
QUEUE(front=0 rear=4) = 10 | 20 | 30 | 40 |
큐는 포화상태 입니다.

-- 데이터 삭제 단계 --
꺼내진 정수 : 10
QUEUE(front=1 rear=4) = 20 | 30 | 40 |
꺼내진 정수 : 20
QUEUE(front=2 rear=4) = 30 | 40 |
꺼내진 정수 : 30
QUEUE(front=3 rear=4) = 40 |
꺼내진 정수 : 40
QUEUE(front=4 rear=4) =
큐는 공백상태 입니다.
Program ended with exit code: 0

All Output ⌃   Filter
```

큐의 응용

버퍼

큐는 서로 다른 속도로 실행되는 2개의 프로세스 간의 상호 작용을 조화 시키는 버퍼 역할을 담당한다.
대게 데이터를 생산하는 생산자 프로세스가 있고, 데이터를 소비하는 소비자 프로세스가 있다. 이 사이에 큐로 구성되는 버퍼가 존재한다.

버퍼 프로그램

```
#include

#include

#include


#define MAX_QUEUE_SIZE 5


typedefintelement;

typedefstruct

{

    element  data[MAX_QUEUE_SIZE];

    int  front, rear;

} QueueType;


void error(char *message)

{

    fprintf(stderr, "%s₩n", message);

    exit(1);

}


voidinit_queue(QueueType *q)
```

```
{
    q->front = q->rear = 0;
}

intis_empty(QueueType *q)
{
    return (q->front == q->rear);
}

intis_full(QueueType *q)
{
    return ((q->rear + 1) % MAX_QUEUE_SIZE == q->front);
}

voidqueue_print(QueueType *q)
{
    printf("QUEUE(front=%d rear=%d) = ", q->front, q->rear);
    if (!is_empty(q))
    {
        int i = q->front;
        do
        {
            i = (i + 1) % (MAX_QUEUE_SIZE);
            printf("%d | ", q->data[i]);
            if (i == q->rear)
                break;
        } while (i != q->front);
    }
    printf("₩n");
}

voidenqueue(QueueType *q, element item)
{
    if (is_full(q))
        error("큐가 포화상태 입니다.");
    q->rear = (q->rear + 1) % MAX_QUEUE_SIZE;
    q->data[q->rear] = item;
}
```

```
elementdequeue(QueueType *q)
{
    if (is_empty(q))
        error("큐가 공백상태 입니다.");
    q->front = (q->front + 1) % MAX_QUEUE_SIZE;
    return q->data[q->front];
}

elementpeek(QueueType *q)
{
    if (is_empty(q))
        error("큐가 공백상태 입니다.");
    return q->data[(q->front + 1) % MAX_QUEUE_SIZE];
}

intmain(void)
{
    QueueType queue;
    int element;

    init_queue(&queue);
    srand(time(NULL));

    for(int i=0;i<100; i++)
    {
        if (rand() % 5 == 0)
        {
            enqueue(&queue, rand()%100);
        }

        queue_print(&queue);
        if (rand() % 10 == 0)
        {
            int data = dequeue(&queue);
        }
        queue_print(&queue);
    }
    return 0;
}
```

덱(Deque)

double-ended queue의 줄임말로 큐의 전단과 후단에서 모두 삽입과 삭제가 가능한 큐이다.

덱 ADT

객체 : n개의 element형으로 구성된 요소들의 순서있는 모임

연산

create() : 덱을 생성한다.
init(dq) : 덱을 초기화한다.
is_empty(dq) : 덱이 공백상태인지를 검사한다.
is_full(dq) : 덱이 포화상태인지를 검사한다.
add_front(dq, e) : 덱의 앞에 요소를 추가한다.
add_rear(dq, e) : 덱의 뒤에 요소를 추가한다.
delete_front(dq, e) : 덱의 앞에 있는 요소를 반환한 다음 삭제한다.
delete_rear(dq, e) : 덱의 뒤에 있는 요소를 반환한 다음 삭제한다.
get_front(dq) : 덱의 앞에서 삭제하지 않고 앞에 있는 요소를 반환한다.
get_rear(dq) : 덱의 뒤에서 삭제하지 않고 뒤에 있는 요소를 반환한다.

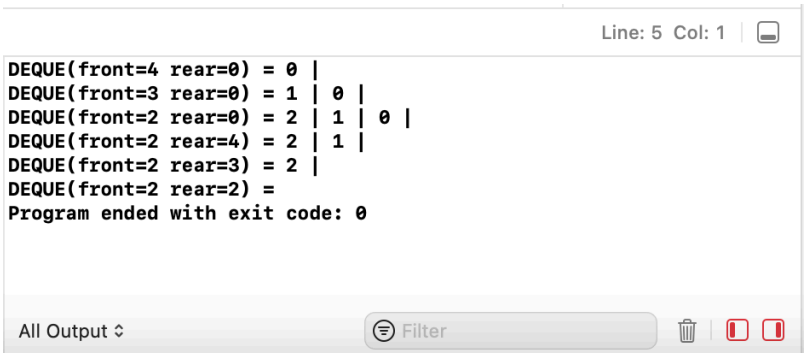
덱 프로그램

```
#include

#include


#define MAX_QUEUE_SIZE 5


typedefintelement;

typedefstruct

{

    element  data[MAX_QUEUE_SIZE];

    int  front, rear;

} DequeType;


void error(char *message)

{

    fprintf(stderr, "%s₩n", message);

    exit(1);

}


voidinit_deque(DequeType *q)

{
```

```
    q->front = q->rear = 0;
}

intis_empty(DequeType *q)
{
    return (q->front == q->rear);
}

intis_full(DequeType *q)
{
    return ((q->rear + 1) % MAX_QUEUE_SIZE == q->front);
}

voiddeque_print(DequeType *q)
{
    printf("DEQUE(front=%d rear=%d) = ", q->front, q->rear);
    if (!is_empty(q))
    {
        int i = q->front;
        do
        {
            i = (i + 1) % (MAX_QUEUE_SIZE);
            printf("%d | ", q->data[i]);
            if (i == q->rear)
                break;
        } while (i != q->front);
    }
    printf("₩n");
}

voidadd_rear(DequeType *q, element item)
{
    if (is_full(q))
        error("•∞° ˜»≠ªÛ¬¿'¥œ¥Ÿ");
    q->rear = (q->rear + 1) % MAX_QUEUE_SIZE;
    q->data[q->rear] = item;
}

elementdelete_front(DequeType *q)
```

```
{
    if (is_empty(q))
        error("•∞° ∞¯πÈªÛ¬¿'¥œ¥Ÿ");
    q->front = (q->front + 1) % MAX_QUEUE_SIZE;
    return q->data[q->front];
}

elementget_front(DequeType *q)
{
    if (is_empty(q))
        error("•∞° ∞¯πÈªÛ¬¿'¥œ¥Ÿ");
    return q->data[(q->front + 1) % MAX_QUEUE_SIZE];
}

voidadd_front(DequeType *q, element val)
{
    if (is_full(q))
        error("•∞° ˜»≠ªÛ¬¿'¥œ¥Ÿ");
    q->data[q->front] = val;
    q->front = (q->front - 1 + MAX_QUEUE_SIZE) % MAX_QUEUE_SIZE;
}

elementdelete_rear(DequeType *q)
{
    int prev = q->rear;
    if (is_empty(q))
        error("•∞° ∞¯πÈªÛ¬¿'¥œ¥Ÿ");
    q->rear = (q->rear - 1 + MAX_QUEUE_SIZE) % MAX_QUEUE_SIZE;
    return q->data[prev];
}

elementget_rear(DequeType *q)
{
    if (is_empty(q))
        error("•∞° ∞¯πÈªÛ¬¿'¥œ¥Ÿ");
    return q->data[q->rear];
}

intmain(void)
```

```
{
    DequeType queue;

    init_deque(&queue);
    for (int i = 0; i < 3; i++)
    {
        add_front(&queue, i);
        deque_print(&queue);
    }
    for (int i = 0; i < 3; i++)
    {
        delete_rear(&queue);
        deque_print(&queue);
    }
    return 0;
}
```



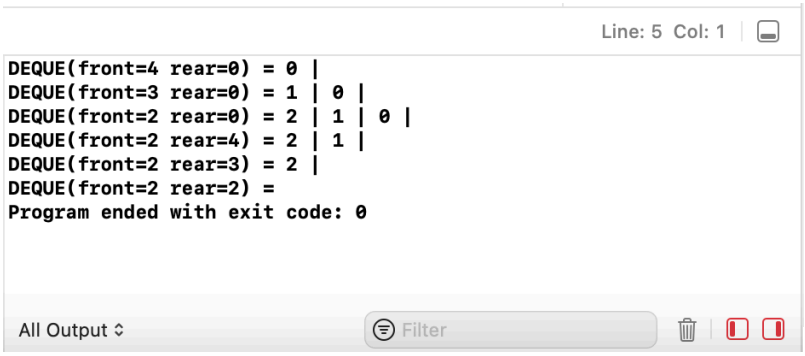
은행 서비스 시뮬레이션 프로그램

```
#include
#include
#include

#define MAX_QUEUE_SIZE 5

typedefstruct
{
    int id;
    int arrival_time;
    int service_time;
} element;

typedefstruct
{
```

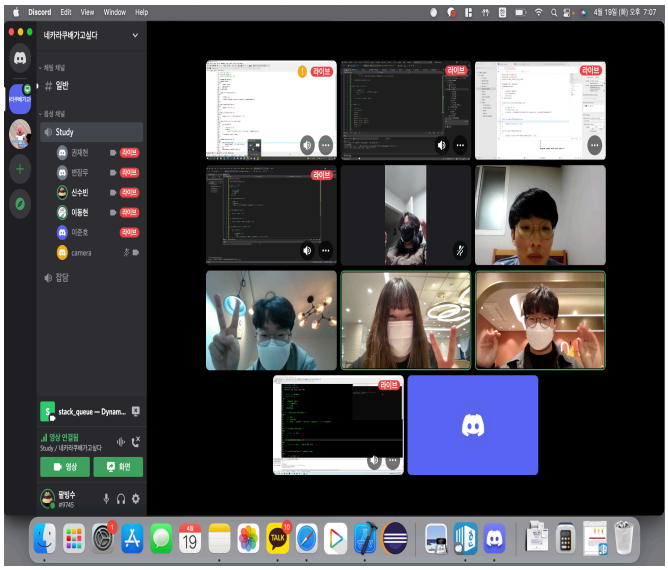
```
    element  data[MAX_QUEUE_SIZE];
    int  front, rear;
} QueueType;

void error(char *message)
{
    fprintf(stderr, "%s₩n", message);
    exit(1);
}

voidinit_queue(QueueType *q)
{
    q->front = q->rear = 0;
}

intis_empty(QueueType *q)
{
    return (q->front == q->rear);
}

intis_full(QueueType *q)
{
    return ((q->rear + 1) % MAX_QUEUE_SIZE == q->front);
}

voidqueue_print(QueueType *q)
{
    printf("QUEUE(front=%d rear=%d) = ", q->front, q->rear);
    if (!is_empty(q))
    {
        int i = q->front;
        do
        {
            i = (i + 1) % (MAX_QUEUE_SIZE);
            printf("%d | ", q->data[i]);
            if (i == q->rear)
                break;
        } while (i != q->front);
    }
```

```
    printf("₩n");
}

voidenqueue(QueueType *q, element item)
{
    if (is_full(q))
        error("큐가 포화상태 입니다.");
    q->rear = (q->rear + 1) % MAX_QUEUE_SIZE;
    q->data[q->rear] = item;
}

elementdequeue(QueueType *q)
{
    if (is_empty(q))
        error("큐가 공백상태 입니다.");
    q->front = (q->front + 1) % MAX_QUEUE_SIZE;
    return q->data[q->front];
}

elementpeek(QueueType *q)
{
    if (is_empty(q))
        error("큐가 공백상태 입니다.");
    return q->data[(q->front + 1) % MAX_QUEUE_SIZE];
}

intmain(void)
{
    int minutes = 60;
    int total_wait = 0;
    int total_customers = 0;
    int service_time = 0;
    int service_customer;
    QueueType queue;
    init_queue(&queue);

    srand(time(NULL));
    for (int clock = 0; clock < minutes; clock++)
    {
```

```
        printf("현재시각 = %d₩n", clock);
        if ((rand()%10) < 3)
        {
            element customer;
            customer.id = total_customers++;
            customer.arrival_time = clock;
            customer.service_time = rand() % 3+1;
            enqueue(&queue, customer);
            printf("고객 %d이 %d분에 들어옵니다. 업무처리시간 = %d분₩n",
                customer.id, customer.arrival_time, customer.service_time);
        }

        if (service_time > 0)
        {
            printf("고객 %d 업무처리중입니다.₩n", service_customer);
            service_time--;
        }

        else
        {
            if (!is_empty(&queue))
            {
                element customer = dequeue(&queue);
                service_customer = customer.id;
                service_time = customer.service_time;
                    printf("고객 %d이 %d분에 업무를 시작합니다. 대기시간은
%d분이었습니다.₩n", customer.id, clock, clock - customer.arrival_time);
                total_wait += clock - customer.arrival_time;
            }
        }
    }

    printf("전체 대기 시간 = %d분 ₩n", total_wait);
    return 0;
}
```


Line: 5 Col: 1
DEQUE(front=4 rear=0) = 0 |
DEQUE(front=3 rear=0) = 1 | 0 |
DEQUE(front=2 rear=0) = 2 | 1 | 0 |
DEQUE(front=2 rear=4) = 2 | 1 |
DEQUE(front=2 rear=3) = 2 |
DEQUE(front=2 rear=2) =
Program ended with exit code: 0
All Output
Filter

『제 24기 청학동』Tutoring 5주 주간 학습 보고서

학습활동 참여 점검	
팀명	2+3=5
성찰활동 잘한 점	신수빈 : 다음 주에 튜티들이 해당 과목(자료구조) 중간고사를 보는데 그 간의 튜터링 활동을 통해 적게나마 도움이 되었다고 생각한다. 시험 이후에도 함께 머리를 맞대고 학습하며 집단지성 효과를 보고 싶다.변장무 : 시험기간이라 정신없는 상황에서도 불구하고 청학동 실습에서 집중하여 좋은 결과를 얻기 위해 노력하였다. 자료구조를 완벽히 이해하여 좋은 프로그래밍 결과와 시험 결과를 위해 노력하였다.이동현 : 코드 리뷰를 통한 의사소통 덕분에 팀원들 모두가 이해하고 넘어갈 수 있었던 거 같다. 지난 청학동 시간에서 전체적인 시간이 오래 걸렸던 것이 문제였으나 이번 주차부터는 조금씩 줄어들고 있는 것 같다.이준호 : 코드에서 찾지 못한 오류가 나는 부분을 도와주었다.권재현 : 수업 시간에 배웠던 내용을 떠올리고 청학동 시간에 응용해 보려고 시도해 보았다.
성찰활동 개선할 점	신수빈 : 중간평가에서 1~4주차 보고서에 문제가 없었던 것을 확인하고 다른 경쟁 팀들의 현황도 파악할 수 있었다. 생각보다 합격한 팀들이 많아서 수상을 하기 위해서는 앞으로 더 노력해야 할 것 같다.변장무 : 도저히 이해가 가지 않는 프로그램이 있어서 중간에 포기하려는 생각이 들 때가 종종 생긴다. 하지만 청학동을 같이하는 인원들이 잘 이끌어주어 끝까지 잘 마무리할 수 있었다. 이러한 성격을 고쳐 더욱 꾸준히 노력하는 사람이 되어야 할 것 같다. 이동현 : 지난 시간 문제점을 극복하고 있으나 팀원들이 전체적으로 복습을 많이 하지 않아 튜터링 시간에 복습이 이루어져 활동 시간이 많이 줄지는 않았다. 다음 주차부터는 팀원 모두가 복습을 해 와 시간을 줄일 수 있었으면 좋겠다.이준호 : 없다.권재현 : 코딩 실력이 부족해서 머리가 마음을 따라가주지 못하여 잘 안됐다.
구성원 학습활동 사진	
다음 학습할 내용	1. 트리 // 2. 이진 트리 // 3. 이진 탐색 트리