✓ 미션 1 (난이도 : 최하)

1. 미션 제목

출근 (등교) 를 해보자!

2. 지시문

본인이 생각하는 최적의 출근 (등교) 길을 표현해보자.

1. 의사코드 (Pseudo code) 로 표현하라.
2. 1에서 작성한 의사코드를 어떤식으로 개발할지 고민해보자.
   (완벽한 구현이 아닌 필수 분기, 혹은 함수로 나누는 부분에 대한 고민)
   A. 변수?
   B. 함수?
   C. 분기?
3. 생각해보기
   A. 의사코드가 필요한 이유?
   B. 메인 함수 외에 다른 helper 함수를 사용하는 이유?
4. 더 나아가 **(Optional - 안하셔도 평가에는 따로 영향이 없습니다.)**
   A. 의사코드를 프로그램으로 만들어보기
   (아래 예시처럼, 조건이 있는 경우들은 조건들을 매개변수로 받아서 구현, 혹은 콘솔 입력을 받아서 구현)

3. 예시

의사코드

- 어제 술을 마셨나?
  - Yes: 9시 30분까지 잔다
    - 대중교통을 이용하기엔 너무 피곤한가?
      - Yes: 자가용 이용
  - No: 7시 30분 기상
- 자가용 이용
  - 네비게이션 목적지 입력 후, 무료도로로 안내
- 대중교통 이용
  - 근처 역 방향 버스를 아무거나 탄다 (집앞 정류장에서 모든 버스가 역을 지남)
  - 분당선 왕십리 급행 혹은 완행 열차를 탄다
  - 신분당선으로 환승
    - 미금역을 놓쳤는가?
      - 정신차리고 정자역에서 환승한다
    - 미금역에서 환승한다
  - 신분당선 강남행 탑승
  - 판교역 하차
- 회사도착

개발 고민 예시 (Optional)

```
{
   ...

   if (drink_yesterday)
   {
      use_car();
   }
   else
   {
```

```
        use_public_transit();
    }

}

void
use_public_transit()
{
    ...
    use_bus("home", "subway");
    use_subway("망포역", "판교역");
    ...
}
```

4. 핵심 개념 (키워드 제시)
#의사코드 #코드스타일 #함수 #분기문

✓ 미션 2 (난이도 : 하)

1. 미션 제목
N 의 약수들

2. 지시문
양수 A가 N의 진짜 약수가 되려면, N이 A의 배수이고, A가 1과 N이 아니어야 한다. 어떤 수 N의 진짜 약수가 모두 주어질 때, N을 구하는 프로그램을 아래와 같이 구현했다.

***아래 주석 부분을 채워 주세요.**

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

int *divisor;

int
compare(const void * a, const void * b)
{
    return (*(int*)a - *(int*)b);
}

int
main()
{
    int N;
```

```
    scanf("%d", &N);

    divisor = malloc(sizeof(int) * N);

    for(int i = 0; i < N; i++){
        int tmp;
        scanf("%d", &tmp);
        divisor[i] = tmp;
    }

    qsort(divisor, N, sizeof(int), compare);

    int answer = /**여기를 채우세요.**/;
    printf("%d\n", answer);

    return 0;
}
```

3. 핵심 개념

#약수 #정렬

4. 부가 설명

- 약수: https://en.wikipedia.org/wiki/Divisor
- 정렬: http://www.cplusplus.com/reference/cstdlib/qsort/

✓ 미션 3 (난이도 : 상)

1. 미션 제목

Infix to postfix (https://www.geeksforgeeks.org/stack-set-2-infix-to-postfix/)

2. 지시문

스택을 사용한 기본 문제중 하나로, 중위 표기법을 후위 표기법으로 변환하는 프로그램을 만드는 것 입니다.

- 중위 표기법: 2 + 2
- 후위 표기법: 2 2 +

왜 후위 표기법으로 변환하는게 필요한가?

- 컴파일러는 오른쪽에서 왼쪽 혹은 왼쪽에서 오른쪽으로 표기법을 읽기 때문
- 혹은 추가로 Use case 들을 찾아보아요~

알고리즘

1. 중위 표기법을 왼쪽에서 오른쪽으로 읽는다.
2. 문자가 피연산자 (operand) 라면 출력결과에 저장
3. 연산자 (operator) 라면
    3.1. 연산자가 들어오면 자기보다 우선순위가 높거나 같은 것들을 Pop 하고 출력결과에 저장 (Pop 도중에 괄호가 나타나면 pop 중지), 자신을 Stack 에 Push 하시오.
4. 여는 괄호 (“(“) 를 만나면 무조건 Stack 에 Push 하시오.
5. 닫는 괄호 (“)”) 를 만나면 여는 괄호 (“(“)를 만날 때까지 Stack 에서 Pop 하여 출력결과에 저장
6. 2 ~ 5 번 과정을 반복
7. 출력결과 출력
8. Stack 이 Empty 가 될 때까지, Pop 하고 출력

*** 6주차 미션에서 구현한 스택을 이용하면 편하겠네요.**

3. 핵심 개념

#infix #postfix #중위표기법 #후위표기법

4. 부가 설명

- 중위표기법: https://en.wikipedia.org/wiki/Infix_notation
- 후위표기법: https://en.wikipedia.org/wiki/Reverse_Polish_notation
- Stack: https://en.wikipedia.org/wiki/Stack_(abstract_data_type)

✓ 미션 4 (난이도 : 상)

1. 미션 제목

중복 빼고 정렬하기

2. 지시문

N개의 정수가 주어진다. 이때, N개의 정수를 오름차순으로 정렬하는 프로그램을 작성하시오. 같은 정수는 한번만 출력한다.

조건 및 질문

- 구현체를 위 문제은행에 제출하여 성공하시오
- 본인의 구현체의 시간복잡도를 구하시요
- 해당 문제에서 본인의 구현체보다 빠른 답이 있을까요?

3. 핵심 개념

#정렬 #중복 #빅오표기법 #big O notation

4. 부가 설명

- 여러 정렬 방법: https://www.geeksforgeeks.org/sorting-algorithms/

- Big O notation: https://en.wikipedia.org/wiki/Big_O_notation

::답지::

미션1

답안 없음

미션2

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

int *divisor;

int
compare(const void * a, const void * b)
{
   return (*(int*)a - *(int*)b);
}

int
main()
{
        int N;
        scanf("%d", &N);

   divisor = malloc(sizeof(int) * N);

        for(int i = 0; i < N; i++){
                int tmp;
                scanf("%d", &tmp);
                divisor[i] = tmp;
        }

        qsort(divisor, N, sizeof(int), compare);

        int answer = /**여기를 채우세요.**/divisor[0] * divisor[N - 1];
        printf("%d\n", answer);

        return 0;
}
```

미션3

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

struct Stack
{
   int top;
```

```
    unsigned capacity;
    int* array;
};

struct Stack* create_stack(unsigned capacity)
{
    struct Stack* stack = (struct Stack*) malloc(sizeof(struct Stack));

    if (!stack)
        return NULL;

    stack->top = -1;
    stack->capacity = capacity;

    stack->array = (int*) malloc(stack->capacity * sizeof(int));

    return stack;
}

int
is_empty(struct Stack* stack)
{
    return stack->top == -1 ;
}

char
peek(struct Stack* stack)
{
    return stack->array[stack->top];
}

char
pop(struct Stack* stack)
{
    if (!is_empty(stack))
        return stack->array[stack->top--] ;
    return '$';
}

void
push(struct Stack* stack, char op)
{
    stack->array[++stack->top] = op;
}

int
is_operand(char ch)
{
    return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');
}
```

```
int
prec(char ch)
{
    switch (ch)
    {
    case '+':
    case '-':
        return 1;

    case '*':
    case '/':
        return 2;

    case '^':
        return 3;
    }
    return -1;
}

int
infix_to_postfix(char* exp)
{
    int i, k;

    struct Stack* stack = create_stack(strlen(exp));
    if(!stack)
        return -1;

    for (i = 0, k = -1; exp[i]; ++i)
    {
        if (is_operand(exp[i]))
        {
            exp[++k] = exp[i];
        }
        else if (exp[i] == '(')
        {
            push(stack, exp[i]);
        }
        else if (exp[i] == ')')
        {
            while (!is_empty(stack) && peek(stack) != '(')
                exp[++k] = pop(stack);

            if (!is_empty(stack) && peek(stack) != '(')
                return -1;
            else
                pop(stack);
        }
        else
```

```
        {
            while (!is_empty(stack) && prec(exp[i]) <= prec(peek(stack)))
                exp[++k] = pop(stack);

            push(stack, exp[i]);
        }

    }

    while (!is_empty(stack))
        exp[++k] = pop(stack );

    exp[++k] = '\0';
    printf("%s", exp);

    return 1;
}

int
main()
{
    char exp[] = "a+b*(c^d-e)^(f+g*h)-i";

    infix_to_postfix(exp);

    return 0;
}
```

미션4

```
#include <stdio.h>

int
main(void)
{
        int N, i, input, flag[3000] = { 0, };
        scanf("%d", &N);

        for (i = 0; i < N; i++)
        {
                scanf("%d", &input);
                flag[input + 1000]++;
        }

        for (i = 0; i < 3000; i++)
                if (flag[i] != 0)
                        printf("%d ", i - 1000);

        printf("\n");
}
```