자료구조(F012) - 5장 실습 과제

G5_쓴맛불좀도둑들(송유민, 김성준, 김현민, 김진형, 김다원)

01번 - 정답 : 3번, **02번** - 정답 : 4번, **03번** - 정답 : 1번, **04번** - 정답 : 3번,

05번 - 정답 : 2번, **06번** - 정답 : 1번, **07번** - 정답 : 1번, **08번** - 정답 : 1번,

09번 - 정답 : 1번, **10번** - 정답 : 3번, **11번** - 정답 : 3번, **12번** - 정답 : 2번,

13번 - 정답 : 2번, **14번** - 정답 : 3번, **15번** - 정답 : 2번, **16번** - 정답 : 1번,

17번 - 정답 : 1번, **18번** - 정답 : 4번, **19번** - 정답 : 3번, **20번** - 정답 : 1번,

21번 - 정답 : 3번, **22번** - 정답 : 2번, **23번** - 정답 : 4번, **24번** - 정답 : 2번,

25번 - 정답 : 3번, **26번** - 정답 : 1번, **27번** - 정답 : 4번, **28번** - 정답 : 5번,

29번 - 정답 : 4번, **30번** - 정답 : 2번, **31번** - 정답 : 1번, **32번** - 정답 : 2번,

33번 - 정답 : 3번, **34번** - 정답 : 2번, **36번** - 정답 : 3번,

35번

<정답>

재귀 함수를 호출할 때마다 현재 상태를 스택에 저장한 뒤에, 가장 나중에 호출한 함수가 먼저 실행되고 완료되면 반환된다. 이때 모든 재귀 호출이 끝나면 가장 마지막에 호출된 함수부터 차례대로 반환된다.

37번

<정답>

3, 4를 스택에 push -> 스택 : [3, 4]

* 연산자를 만나 스택의 두 값을 pop 해주고 계산값을 다시 스택에 push -> 스택 : [12]

연산자를 만날 때까지 숫자를 스택에 push 해주고 -> 스택 : [12, 5, 6]

* 연산자를 만나 스택의 두 값을 pop 해주고 연산 후 스택에 push -> 스택 : [12, 30]

+ 연산자를 만나 스택의 두 값을 pop 해주고 연산 후 스택에 push -> 스택 : [42]

정답 : 42

38번

<정답>

배열의 범위보다 더 많은 데이터를 저장하려고 할 때 스택에 오버플로우가 발생하게 된다. 해결 방법으로는 순차 자료구조가 아닌 연결 자료구조로 linked list 를 사용하면 해결할 수 있다.

39번

<정답>

(가) A B + C - D -

(나) A B + C * D +

(다) A B / C / D /

40번

<정답>

A와 B를 스택에 push 스택 : [A, B]

'-' 연산자를 만나 스택의 두 값을 pop 해주고 계산, A - B 계산 값을 스택에 push 스택 : [(A - B)]

C를 스택에 push 스택 : (A - B) ,C

'-' 연산자를 만나 스택의 두 값을 pop 해주고 계산, (A - B) - C 계산 값을 스택에 push 스택 : {(A - B) - C}

D를 스택에 push 스택 : {(A - B) - C}, D

'-' 연산자를 만나 스택의 두 값을 pop 해주고, {(A - B) - C} - D 계산 값을 스택에 push 스택 : [{(A - B) - C} - D]

41번

<정답>

A를 출력 스택 : NULL

B를 스택에 push 스택 : B

C를 출력 스택 : B

B를 pop 스택 : NULL

D를 출력 스택 : NULL

42번

<정답>

나중에 호출한 함수부터 값이 return 되니 LIFO 후입선출 스택 구조가 적당하다

43번

<코드>

```c
// 스택 초기화 함수
void inStack(Stack* s) {
    s->top = -1;
}

// 스택이 비었는지 확인하는 함수
int Empty(Stack* s) {
    return s->top == -1;
}

// 스택이 가득 찼는지 확인하는 함수
int Full(Stack* s) {
    return s->top == 99;
}

// 스택에 노드를 push하는 함수
void push(Stack* s, listNode* node) {
    if (!Full(s)) {
        s->nodes[++(s->top)] = node;
    }
}

// 스택에서 노드를 pop하는 함수
listNode* pop(Stack* s) {
    if (!Empty(s)) {
        return s->nodes[(s->top)--];
    }
    return NULL;
}
```

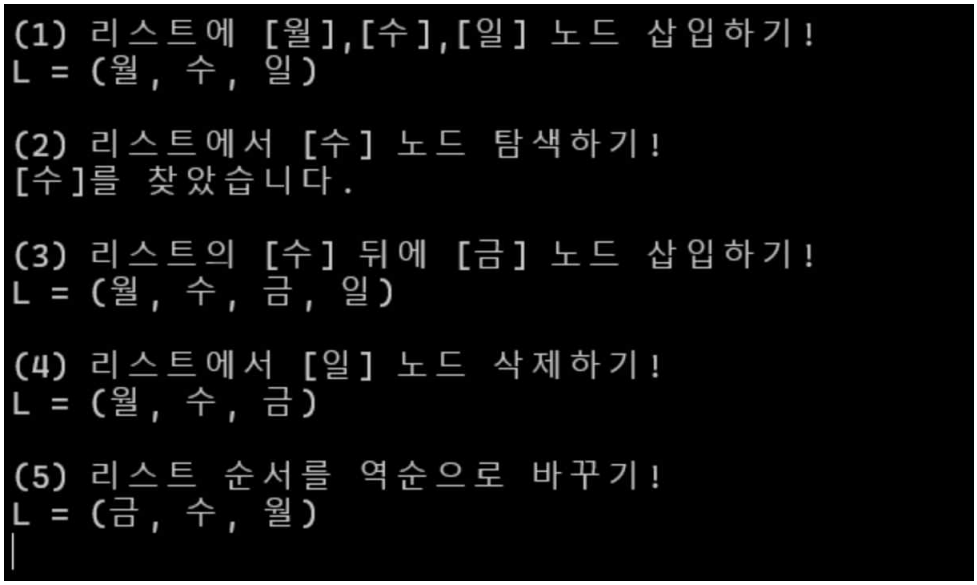
```c
// 리스트의 노드 순서를 역순으로 바꾸는 연산
void reverse(linkedList_h* L) {
    Stack s;
    inStack(&s);
    listNode* current = L->head;

    // 리스트의 모든 노드를 스택에 push
    while (current != NULL) {
        push(&s, current);
        current = current->link;
    }

    // 스택에서 pop하면서 리스트를 재구성
    if (!Empty(&s)) {
        L->head = pop(&s);  // 첫 번째 노드를 스택에서 꺼내고 헤드로 설정
        current = L->head;
    }

    while (!Empty(&s)) {
        current->link = pop(&s);  // 스택에서 노드를 꺼내서 리스트를 재구성
        current = current->link;
    }
    current->link = NULL;  // 마지막 노드의 link를 NULL로 설정
}
```

<실행 결과>


(1) 리스트에 [월],[수],[일] 노드 삽입하기!
L = (월, 수, 일)
(2) 리스트에서 [수] 노드 탐색하기!
[수]를 찾았습니다.
(3) 리스트의 [수] 뒤에 [금] 노드 삽입하기!
L = (월, 수, 금, 일)
(4) 리스트에서 [일] 노드 삭제하기!
L = (월, 수, 금)
(5) 리스트 순서를 역순으로 바꾸기!
L = (금, 수, 월)