



Dash - not\_git

ex 00

*Summary:* 이 과제는 git과 전혀 상관없는 과제입니다.

# Contents

I. Forward

II. Instructions

III. Exercise 00 : 앨런 튜닝샵

IV. Exercise 01 : 덤앤더머

# Chapter 1

## Forward



"튜닝의 끝은 순정이 아니라 순정 튜닝이다."

*(The perfect form of tuning is not genuine, but genuine tuning.)*


# Chapter 2

## Instructions

- 과제 중에 절대 `git`을 떠올리지 마세요! (진지)
- `compile`은 `cc` 컴파일러를 사용하며 `-Wall -Wextra -Werror` 옵션을 지정해야 합니다.
- 노미네이트는 없습니다. 그냥...없어요.

## Chapter 3

### Exercise 00 :

	Exersize 00
앨런 튜닝샵	
Turn-in directory : ex00/	
Files to turn in : *.c (without main), ex00.h	
Allowed functions : Anything you want	

튜닝샵을 운영 중인 Alan은 마을에 자주 나타나는 변덕스런 손님 Jerry를 대비하여 간단한 변경사항 저장 툴을 적용하려 한다.

코딩을 잘 알지 못하는 Alan은 우리에게 “요청 추가”, “요청 내역 n번 전으로 되돌리기”, “변경사항 작성 완료”, 세 가지의 기능이 들어간 프로그램을 의뢰하였다.

변덕스런 손님 Jerry가 수 많은 요청 추가와 되돌리기로 Alan을 괴롭히기 전에 얼른 프로그램을 완료해보자.

- 이 프로그램이 정상적으로 종료되면, 함께 제공되는 car.txt 파일 내용 중 변경된 사항을 적용하여 새로운 파일인 car.replace을 생성하여야 한다.
- car.txt 파일 내에는 Jerry가 소유하고 있는 자동차의 현재 상태를 나타내는 정보를 가지고 있으며, “key:value”형식으로 구성되어 있다.
- 이 서버젝트의 모든 key 입력값은, 예외처리가 없는 valid한 key값만 주어지는 것이 전제되어 있다. (car.txt에 없는 key값을 예외처리할 필요가 없다.)
- 아래는 car.txt파일 예시

```
bumper:red
door:white
trunk:black
wheel:silver
```

- 헤더파일에는 아래의 구조체가 포함되어야 하며, 다른 구조체는 포함할 수 없다.

```
typedef struct s_node{
    int number;
    char *key;
    char *value;
    struct s_node *next;
    struct s_node *prev;
} t_node;
```

- number: 0부터 시작하는 각 노드의 일련번호
- key: 변경되어야 할 항목의 key
- value: 변경되어야 할 항목의 value
- next, prev: 연결리스트 주소

아래의 메인문이 정상적으로 동작하여야 한다.

```
t_node *get_new_node(int i){
    t_node *new;

    new = (t_node *)malloc(sizeof(t_node));
    new->number = i;
    new->value = NULL;
    new->key = NULL;
    new->next = NULL;
    new->prev = NULL;
    return new;
}

int main()
{
    t_node *list;
    t_node *head;

    list = get_new_node(0);
    head = list;

    commit(list, "bumper", "black");
    commit(list, "bumper", "red");
    commit(list, "trunk", "white");

    print(list);
    restore(list, 2);
    print(list);
    commit(list, "trunk", "yellow");
    print(list);
    push(list);
    return (0);
}
```

### Commit

- 매개변수는 왼쪽부터 연결리스트 header의 주소, 변경될 항목의 key, 변경될 항목의 value
- 연결리스트에 변경될 항목들을 순차적으로 저장하여야 한다.

### Restore

- 매개변수는 왼쪽부터 연결리스트 header의 주소, 되돌리고픈 요청사항의 개수
- 두번째 매개변수로 2가 들어왔다면, 가장 마지막에 저장된 2개의 변경사항이 정상적으로 삭제되어야 하며, 누수가 일어나지 않아야 한다.
- 4개의 요청사항이 있는 경우 두번째 매개변수로 5가 들어오는 등, 비정상적인 입력을 들어오지 않는다고 가정한다.

### Print

- 현재까지의 변경사항들을 출력하여야 한다.

### Push

- car.txt 파일 내에 있는 내용들을 현재까지의 변경사항들을 적용하여 car.replace파일 내에 저장하여야 한다.
- car.txt파일과 car.replace의 key 정렬은 같지 않아도 된다.
- key가 중복되지 않아야 한다.

아래 예시는, 위 main문 예제가 실행될 때 출력 예시

```
$ ./ex00 | cat -e
000 "bumper:black" commit!!$
001 "bumper:red" commit!!$
002 "trunk:white" commit!!$
log 000 "bumper:black"$
log 001 "bumper:red"$
log 002 "trunk:white"$
restore 000$
log 000 "bumper:black"$
003 "trunk:yellow" commit!!$
log 000 "bumper:black"$
log 003 "trunk:yellow"$
pushed!!$
```

아래 예시는, 위 main문 예제가 실행될 때 car.replace 예시

```
bumper:black
door:white
trunk:yellow
wheel:silver
```



git commit시 변경내용이 저장되는 방식과 이 문제에서의 방식이 다르다.  
실제 git commit 시 어떤 내용들이 어떻게 저장되고 관리되는지 알아보자.



제공되는 ex01(.c/.h) 파일은 수정되어선 안 된다.