컴퓨터학부 20211527 박창현
AAAAAA

1. **과제개요**

ext2 파일 시스템 이미지(ext2disk.img)를 C 언어로 직접 파싱해 슈퍼블록·그룹 디스크립터·아이노드를 읽어 경로 해석(resolve_path), 디렉터리 구조 출력(tree), 파일 내용 출력(print) 등 내장 명령어를 제공하는 대화형 쉘(ssu_ext2)을 구현하는 과제입니다.

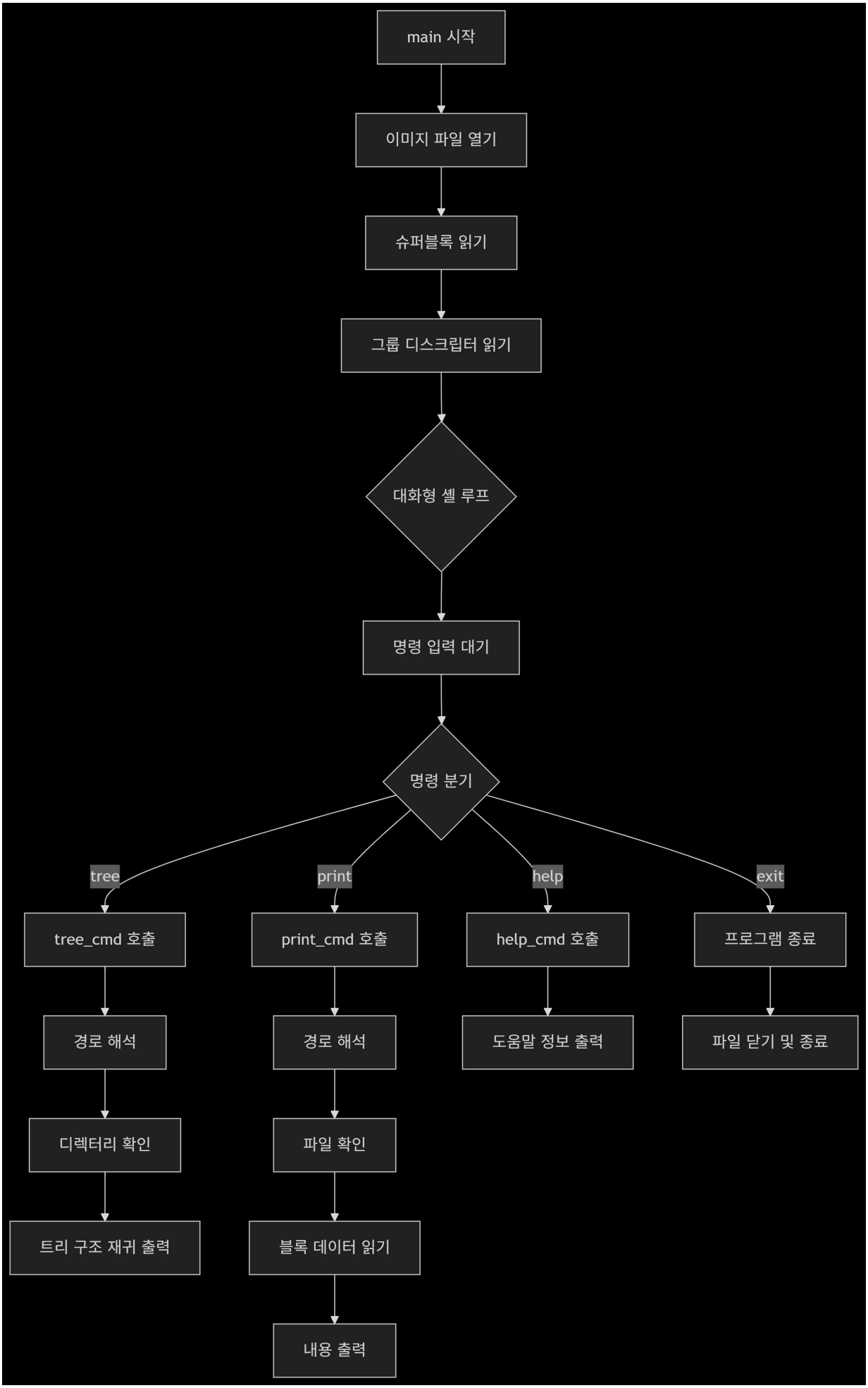
//

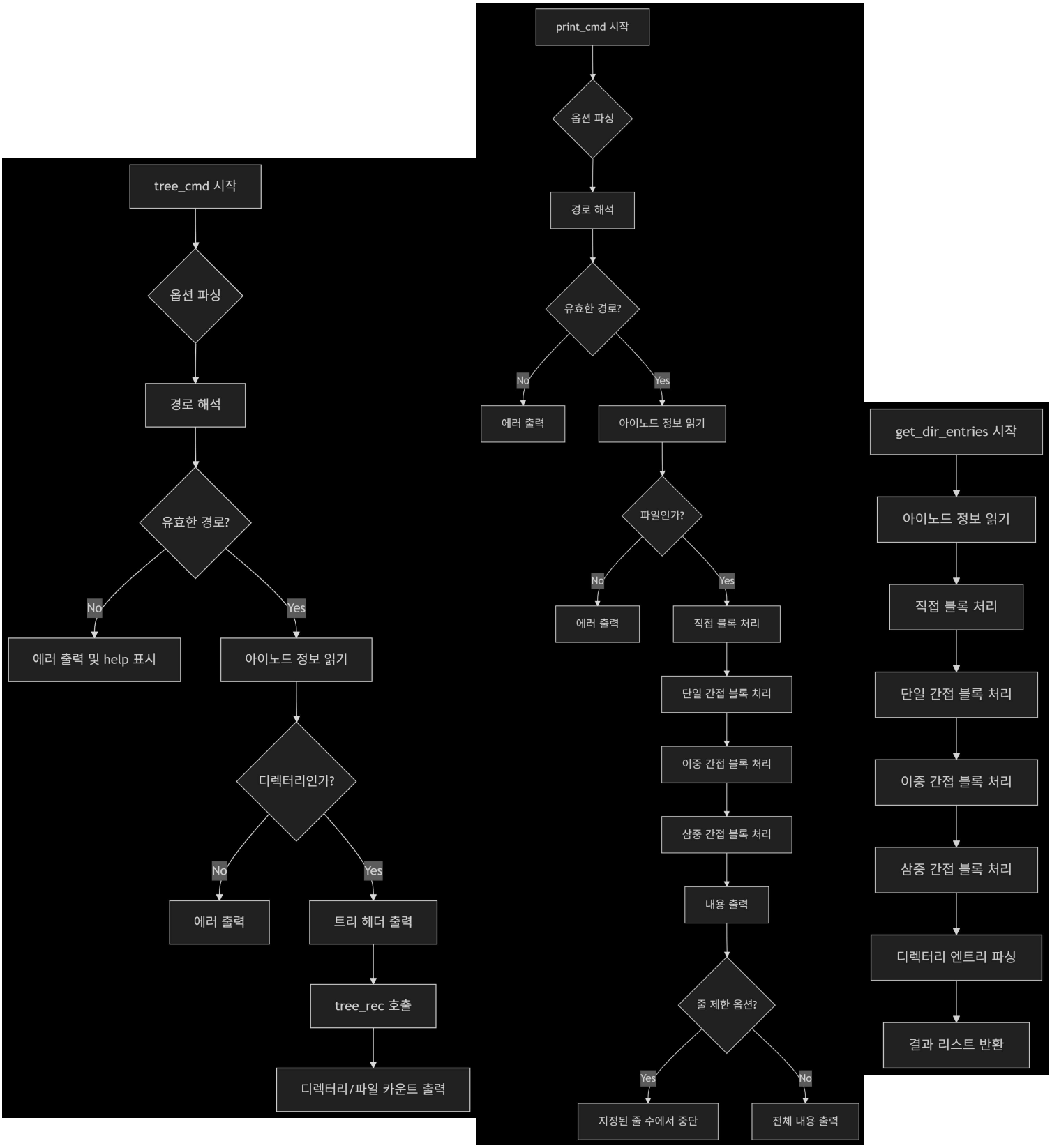
2. 구현기능

```
// 함수 프로토타입 선언
void read_superblock(void);                                      // 슈퍼블록 읽기 함수

void read_group_descs(void);                                     // 그룹 디스크립터 배열 읽기 함수

void read_block(uint32_t blk, void *buf);                        // 주어진 블록 번호의 전체 블록을 buf에 읽음

void get_inode(uint32_t idx, struct ext2_inode *inode);          // 아이노드 정보 읽기

static void parse_dir_block(uint8_t *buf, DirEntry **head, DirEntry **tail);    // 디렉터리 엔트리 파싱 헬퍼

DirEntry *get_dir_entries(uint32_t inode_no);                    // 디렉터리 엔트리 목록 가져오기

void free_entries(DirEntry *head);                               // 디렉터리 엔트리 메모리 해제

void print_permissions(mode_t mode);                             // 권한 정보 출력

void tree_rec(uint32_t inode_no, int depth, int flag_r, int flag_s, int flag_p, int has_next[]); // 재귀적 트리 출력

uint32_t resolve_path(const char *path);                         // 경로 해석 함수

void tree_cmd(int argc, char **argv);                            // tree 명령 처리 함수

static void append_block(BufferNode **head, BufferNode **tail, uint8_t *buf, size_t len);        // 리스트에 새 노드 추가

static void free_blocks(BufferNode *head);                       // 리스트 메모리 해제

void print_cmd(int argc, char **argv);                           // print 명령 처리 함수

void help_cmd(int argc, char **argv);                            // help 명령 처리 함수
```

//

3. 상세설계 – 함수 순서도


main 시작
이미지 파일 열기
슈퍼블록 읽기
그룹 디스크립터 읽기
대화형 셸 루프
명령 입력 대기
명령 분기
tree
print
help
exit
tree_cmd 호출
print_cmd 호출
help_cmd 호출
프로그램 종료
경로 해석
경로 해석
도움말 정보 출력
파일 닫기 및 종료
디렉터리 확인
파일 확인
트리 구조 재귀 출력
블록 데이터 읽기
내용 출력



주요 데이터 구조

ext2_super_block: 파일 시스템 메타데이터 저장

ext2_group_desc: 블록 그룹 정보 저장

ext2_inode: 파일/디렉터리 메타데이터 저장

DirEntry: 디렉터리 엔트리 연결 리스트

BufferNode: 파일 내용 버퍼 연결 리스트

초기화 및 파일 시스템 구조 읽기 함수

read_superblock(): EXT2 파일 시스템의 슈퍼블록을 읽고 검증
read_group_descs(): 그룹 디스크립터 테이블 읽기
read_block(): 지정된 블록 번호의 데이터 읽기

아이노드 관련 함수

get_inode(): 특정 아이노드 정보 읽기
resolve_path(): 경로 문자열을 아이노드 번호로 변환

디렉터리 처리 함수

parse_dir_block(): 디렉터리 블록 파싱
get_dir_entries(): 디렉터리 내 모든 엔트리 가져오기
free_entries(): 디렉터리 엔트리 메모리 해제

명령 처리 함수

tree_cmd(): 디렉터리 트리 구조 출력
print_cmd(): 파일 내용 출력
help_cmd(): 도움말 출력

유틸리티 함수

print_permissions(): 파일 권한 정보 출력
append_block(): 버퍼 리스트에 새 노드 추가
free_blocks(): 버퍼 리스트 메모리 해제

//

4. 실행결과(구현한 모든 기능 및 실행 결과 캡쳐)

5-1-0 ssu_ext2 프로그램의 실행결과는 "학번>"이 표준 출력되고 내장명령어(tree, print, help, exit) 입력 대기, 첫 번째 인자 없이 ssu_ext2 프로그램 실행 시 Usage 출력

```
changhyeon@RSP:~$ ./ssu_ext2 ~/test_dir/ext2disk.img
20211527> exit
changhyeon@RSP:~$ ./ssu_ext2
Usage Error : ./ssu_ext2 <EXT2_IMAGE>
changhyeon@RSP:~$
```

5-1-1 프롬프트 상에서 지정한 내장명령어 외 기타 명령어 입력 시 help 명령어의 결과를 출력 후 프롬프트 재출력

```
changhyeon@RSP:~$ ./ssu_ext2 ~/test_dir/ext2disk.img
20211527> asd
Usage:
> tree <PATH> [OPTION]... : display the directory structure if <PATH> is a directory
  -r : display the directory structure recursively if <PATH> is a directory
  -s : display the directory structure if <PATH> is a directory, including the size of each file
  -p : display the directory structure if <PATH> is a directory, including the permissions of each directory and file
> print <PATH> [OPTION]... : print the contents on the standard output if <PATH> is file
  -n <line_number> : print only the first <line_number> lines of its contents on the standard output if <PATH> is file
> help [COMMAND] : show commands for program
> exit : exit program
20211527> exit
changhyeon@RSP:~$
```

5-1-2 프롬프트 상에서 엔터만 입력 시 프롬프트 재출력

```
changhyeon@RSP:~$ ./ssu_ext2 ~/test_dir/ext2disk.img
20211527>
20211527>
```

5-2-0 첫 번째 인자 <PATH>가 디렉토리일 시 해당 경로에 있는 파일과 디렉토리들의 이름을 출력(".", "..", "lost+found" 디렉토리는 출력 생략

```
changhyeon@RSP:~$ ./ssu_ext2 ~/test_dir/ext2disk.img
20211527> tree .
.
├── A
├── B
├── ssu_open.c
└── ssu_test.txt

3 directories, 2 files
20211527> tree B
B
├── BB
└── hello.py

2 directories, 1 files
20211527> tree ssu_open.c
Error: 'ssu_open.c' is not directory
20211527>
```

5-2-1 '-r' 옵션 입력 시에는 디렉토리의 하위 파일과 하위 디렉토리들의 이름도 추가로 출력

```
changhyeon@RSP:~$ ./ssu_ext2 ~/test_dir/ext2disk.img
20211527> tree . -r
.
├── A
│   ├── AA
│   └── hello.c
├── B
│   ├── BB
│   │   └── bye.cpp
│   └── hello.py
├── ssu_open.c
└── ssu_test.txt

5 directories, 5 files
20211527> tree B -r
B
├── BB
│   └── bye.cpp
└── hello.py

2 directories, 2 files
20211527>
```

5-2-2 '-s' 옵션 입력 시에는 파일 또는 디렉토리의 크기와 함께 출력

```
changhyeon@RSP:~$ ./ssu_ext2 ~/test_dir/ext2disk.img
20211527> tree . -s
[4096] .
├── [4096] A
├── [4096] B
├── [0] ssu_open.c
└── [74] ssu_test.txt

3 directories, 2 files
20211527> tree B -s
[4096] B
├── [4096] BB
└── [0] hello.py

2 directories, 1 files
20211527>
```

5-2-3 '-p'옵션 입력 시에는 파일 또는 디렉토리의 권한과 함께 출력

```
changhyeon@RSP:~$ ./ssu_ext2 ~/test_dir/ext2disk.img
20211527> tree . -p
[drwxr-xr-x ] .
├── [drwxr-xr-x ] A
├── [drwxr-xr-x ] B
├── [-rw-r--r-- ] ssu_open.c
└── [-rw-r--r-- ] ssu_test.txt

3 directories, 2 files
20211527> tree B -p
[drwxr-xr-x ] B
├── [drwxr-xr-x ] BB
└── [-rw-r--r-- ] hello.py

2 directories, 1 files
20211527>
```

5-2-4 '-r', '-s', '-p' 세 가지 옵션은 중복해서 사용 가능함

```
changhyeon@RSP:~$ ./ssu_ext2 ~/test_dir/ext2disk.img
20211527> tree . -s -p
[drwxr-xr-x 4096] .
├── [drwxr-xr-x 4096] A
├── [drwxr-xr-x 4096] B
├── [-rw-r--r-- 0] ssu_open.c
└── [-rw-r--r-- 74] ssu_test.txt

3 directories, 2 files
20211527> tree B -rsp
[drwxr-xr-x 4096] B
├── [drwxr-xr-x 4096] BB
│   └── [-rw-r--r-- 81] bye.cpp
└── [-rw-r--r-- 0] hello.py

2 directories, 2 files
20211527>
```

5-2-5 첫 번째 인자로 올바르지 않은 경로 입력 시 Usage 출력 후 프롬프트 재출력,
첫 번째 인자가 디렉토리가 아닐 경우 에러 메시지 출력 후 프롬프트 재출력
두 번째 인자로 올바르지 않은 옵션이 들어왔을 경우 Usage 출력 후 프롬프트 재출력

```
20211527> tree C
Usage: tree <PATH> [OPTION]...
Display the directory structure if <PATH> is a directory
Options:
  -r : display the directory structure recursively
  -s : display the directory structure including the size of each file
  -p : display the directory structure including the permissions of each directory and file
20211527> tree ssu_test.txt
Error: 'ssu_test.txt' is not directory
20211527> tree . -n
Invalid option -- 'n'
20211527>
```

5-3-0 print 내장명령어 실행. 첫 번째 인자 <PATH>가 파일일 시 해당 경로에 있는 파일의 내용을 출력,
첫 번째 인자가 파일이 아닐 경우 에러 메시지 출력 후 프롬프트 재출력

```
changhyeon@RSP:~$ ./ssu_ext2 /home/changhyeon/test_dir/ext2disk.img
20211527> print ssu_test.txt
Linux System Programming!
Unix System Programming!
Linux Mania
Unix Mania
20211527> print B/BB/bye.cpp
#include <iostream>
using namespace std;

int main() {
        cout << "bye" << endl;
}
20211527> print A
Error: 'A' is not file
20211527> print B
Error: 'B' is not file
20211527>
```

5-3-1 '-n' 옵션 입력 시에는 파일의 내용을 입력받은 라인 수 만큼만 출력

두 번째 인자로 올바르지 않은 옵션이 들어왔을 경우 Usage 출력 후 프롬프트 재출력

첫 번째 인자로 올바르지 않은 경로(존재하지 않는 파일) 입력 시 Usage 출력 후 프롬프트 재출력

```
changhyeon@RSP:~$ ./ssu_ext2 /home/changhyeon/test_dir/ext2disk.img
20211527> print ssu_test.txt -n 2
Linux System Programming!
Unix System Programming!
20211527> print A/hello.c -n
print: option requires an argument -- 'n'
20211527> print B/BB/bye.cpp -a
print: option requires an argument -- 'n'
20211527> print asd
Error: 'asd' is not file
Usage: print <PATH> [-n <lines>]
20211527>
```

5-4-0 help 명령어 실행결과(첫 번째 인자[COMMAND]에 대해 해당 내장명령어에 대한 설명(Usage)를 출력, 첫 번째 인자는 생략 가능하며, 생략 시 모든 내장명령어에 대한 설명(Usage) 출력

```
changhyeon@RSP:~$ ./ssu_ext2 /home/changhyeon/test_dir/ext2disk.img
20211527> help
Usage:
> tree <PATH> [OPTION]... : display the directory structure if <PATH> is a directory
  -r : display the directory structure recursively if <PATH> is a directory
  -s : display the directory structure if <PATH> is a directory, including the size of each file
  -p : display the directory structure if <PATH> is a directory, including the permissions of each directory and file
> print <PATH> [OPTION]... : print the contents on the standard output if <PATH> is file
  -n <line_number> : print only the first <line_number> lines of its contents on the standard output if <PATH> is file
> help [COMMAND] : show commands for program
> exit : exit program
20211527> help tree
Usage: tree <PATH> [OPTION]...
Display the directory structure if <PATH> is a directory
Options:
  -r : display the directory structure recursively
  -s : display the directory structure including the size of each file
  -p : display the directory structure including the permissions of each directory and file
20211527> help asd
invalid command -- 'asd'
Usage:
> tree <PATH> [OPTION]... : display the directory structure if <PATH> is a directory
  -r : display the directory structure recursively if <PATH> is a directory
  -s : display the directory structure if <PATH> is a directory, including the size of each file
  -p : display the directory structure if <PATH> is a directory, including the permissions of each directory and file
> print <PATH> [OPTION]... : print the contents on the standard output if <PATH> is file
  -n <line_number> : print only the first <line_number> lines of its contents on the standard output if <PATH> is file
> help [COMMAND] : show commands for program
> exit : exit program
20211527> help help
Usage: help [COMMAND]
Show commands for program
20211527> help exit
Usage: exit
Exit program
20211527>
```

5-5-0 exit 명령어 실행결과

```
changhyeon@RSP:~$ ./ssu_ext2 /home/changhyeon/test_dir/ext2disk.img
20211527> exit
changhyeon@RSP:~$
```

//

5. 주석달린 소스코드

<Makefile>

```
# Makefile for ssu_ext2
# 컴파일러 설정
CC      = gcc
# 컴파일 옵션 설정
# -std=c11                  : C11 표준 사용
# -D_POSIX_C_SOURCE=200809L: getline 등 POSIX 기능 활성화
# -Wall -Wextra             : 모든 경고 활성화
# -g                        : 디버그 심볼 포함
CFLAGS  = -std=c11 -D_POSIX_C_SOURCE=200809L -Wall -Wextra -g

# 최종 생성할 실행 파일 이름
TARGET  = ssu_ext2

# 소스 파일 목록
SRC     = ssu_ext2.c

# 오브젝트 파일 목록 (.c → .o)
OBJ     = $(SRC:.c=.o)

# 가상(target) 선언: clean, all 은 파일명이 아닌 명령어
.PHONY: all clean

# 기본 타겟: make 또는 make all 시 실행
all: $(TARGET)

# 실행 파일 생성 규칙
# $@ = TARGET, $^ = 의존 대상(OBJ)
$(TARGET): $(OBJ)
        $(CC) $(CFLAGS) -o $@ $^

# .c 파일로부터 .o 파일을 생성하는 패턴 규칙
# $< = 첫 번째 의존 대상(%.c), $@ = 출력 파일(%.o)
%.o: %.c
        $(CC) $(CFLAGS) -c $< -o $@

# clean 타겟: 빌드 결과물 삭제
clean:
        rm -f $(OBJ) $(TARGET)
```

<ssu_ext2.c>

```c
/* ssu_ext2.c */
#include <stdio.h>       // 표준 입출력 함수 사용을 위한 헤더 (printf, perror 등)
#include <stdlib.h>      // 표준 유틸리티 함수 사용을 위한 헤더 (malloc, free, exit 등)
#include <stdint.h>      // 고정 너비 정수형 타입 정의를 위한 헤더 (uint32_t, uint16_t 등)
#include <string.h>      // 문자열 처리 함수 사용을 위한 헤더 (strcmp, strtok, memcpy 등)
#include <unistd.h>      // POSIX API(예: lseek, read) 사용을 위한 헤더
#include <fcntl.h>       // 파일 제어 옵션(O_RDONLY 등) 정의를 위한 헤더
#include <sys/stat.h>    // 파일 상태 정보(struct stat) 사용을 위한 헤더
#include <sys/types.h>   // POSIX 데이터 타입 정의를 위한 헤더
#include <errno.h>       // 오류 번호(errno) 사용을 위한 헤더
#include <getopt.h>      // getopt(), optind, opterr 선언

#define SSU_ID "20211527"          // 과제 제출자 학번 정의
#define EXT2_SUPER_MAGIC 0xEF53 // EXT2 파일 시스템 식별자 상수
#define EXT2_NAME_LEN 255          // EXT2 파일 이름 최대 길이

// ext2 슈퍼블록 구조체 정의
struct ext2_super_block
{
    uint32_t s_inodes_count;         // 전체 아이노드 수
    uint32_t s_blocks_count;         // 전체 블록 수
    uint32_t s_r_blocks_count;       // 예약된 블록 수
    uint32_t s_free_blocks_count; // 사용되지 않은 블록 수
    uint32_t s_free_inodes_count; // 사용되지 않은 아이노드 수
    uint32_t s_first_data_block;   // 첫 번째 데이터 블록 번호
    uint32_t s_log_block_size;      // 블록 크기 지수 (0→1KB, 1→2KB, 2→4KB...)
    uint32_t s_log_frag_size;       // 프래그먼트 크기 지수
    uint32_t s_blocks_per_group;   // 블록 그룹당 블록 수
    uint32_t s_frags_per_group;    // 블록 그룹당 프래그먼트 수
    uint32_t s_inodes_per_group;   // 블록 그룹당 아이노드 수
    uint32_t s_mtime;                  // 마지막 마운트 시각 (초)
    uint32_t s_wtime;                  // 마지막 쓰기 시각 (초)
    uint16_t s_mnt_count;              // 마운트된 횟수
    uint16_t s_max_mnt_count;       // 최대 마운트 허용 횟수
    uint16_t s_magic;                  // 파일 시스템 매직 넘버 (0xEF53)
    uint16_t s_state;                 // 파일 시스템 상태
    uint16_t s_errors;                // 오류 처리 전략
    uint16_t s_minor_rev_level;    // 마이너 리비전 수준
    uint32_t s_lastcheck;             // 마지막 검사 시각
    uint32_t s_checkinterval;       // 검사 간격
```

```
    uint32_t s_creator_os;          // 생성된 운영체제 종류
    uint32_t s_rev_level;           // 리비전 수준
    uint16_t s_def_resuid;          // 기본 예약 UID
    uint16_t s_def_resgid;          // 기본 예약 GID
    uint32_t s_first_ino;           /* rev ≥1일 때 첫 번째 사용 가능한 아이노드 */
    uint16_t s_inode_size;          /* rev ≥1일 때 아이노드 구조체 크기 */
};

// ext2 그룹 디스크립터 구조체 정의
struct ext2_group_desc
{
    uint32_t bg_block_bitmap;       // 블록 비트맵 블록 번호
    uint32_t bg_inode_bitmap;       // 아이노드 비트맵 블록 번호
    uint32_t bg_inode_table;        // 아이노드 테이블 시작 블록 번호
    uint16_t bg_free_blocks_count; // 그룹 내 사용되지 않은 블록 수
    uint16_t bg_free_inodes_count; // 그룹 내 사용되지 않은 아이노드 수
    uint16_t bg_used_dirs_count;    // 그룹 내 디렉터리 개수
    uint16_t bg_pad;                // 패딩
    uint8_t bg_reserved[12];        // 예약된 필드
};

// ext2 아이노드 구조체 정의
struct ext2_inode
{
    uint16_t i_mode;          // 파일 유형 및 접근 권한
    uint16_t i_uid;           // 소유자 UID
    uint32_t i_size;          // 파일 크기 (바이트)
    uint32_t i_atime;         // 마지막 접근 시간
    uint32_t i_ctime;         // 생성 시간
    uint32_t i_mtime;         // 마지막 수정 시간
    uint32_t i_dtime;         // 삭제 시간
    uint16_t i_gid;           // 그룹 GID
    uint16_t i_links_count; // 하드 링크 수
    uint32_t i_blocks;        // 사용 중인 블록 수
    uint32_t i_flags;         // 파일 플래그
    uint32_t i_osd1;          // OS 의존적 필드 1
    uint32_t i_block[15];    // 데이터 블록 포인터 (직접/간접)
    uint32_t i_generation;   // 파일 버전 (NFS 사용)
    uint32_t i_file_acl;     // ACL 블록
    uint32_t i_dir_acl;      // 디렉터리 ACL
    uint32_t i_faddr;         // 프래그먼트 주소
    uint8_t i_osd2[12];       // OS 의존적 필드 2
};
```

```
// ext2 디렉터리 엔트리 구조체 정의
struct ext2_dir_entry
{
    uint32_t inode;              // 아이노드 번호
    uint16_t rec_len;            // 레코드 길이
    uint8_t name_len;            // 이름 길이
    uint8_t file_type;           // 파일 유형
    char name[EXT2_NAME_LEN]; // 파일 이름
};

// 디렉터리 엔트리를 위한 연결 리스트 구조체
typedef struct DirEntry
{
    char name[EXT2_NAME_LEN + 1]; // 파일/디렉터리 이름 (널 종료)
    uint32_t inode;                  // 아이노드 번호
    mode_t mode;                     // 파일 모드 (권한 및 유형)
    off_t size;                      // 파일 크기
    struct DirEntry *next;           // 다음 엔트리 포인터
} DirEntry;

// 파일 블록 데이터를 저장할 노드
typedef struct BufferNode {
    uint8_t *data;                // 블록 버퍼
    size_t    len;                // 실제 데이터 길이 (마지막 블록은 block_size 미만일 수 있음)
    struct BufferNode *next;    // 다음 노드
} BufferNode;

// 전역 변수 선언
uint16_t inode_size;                 // 아이노드 크기 (기본값 256바이트)
int image_fd;                         // EXT2 이미지 파일 디스크립터
uint32_t block_size;                 // 블록 크기 (바이트)
struct ext2_super_block sb;          // 슈퍼블록 구조체 인스턴스
struct ext2_group_desc *gd_table; // 그룹 디스크립터 구조체 인스턴스
uint32_t inodes_per_group;
uint32_t group_count;

int dir_count = 1;  // 디렉터리 카운터 (루트 포함)
int file_count = 0; // 파일 카운터

// 함수 프로토타입 선언
void read_superblock(void);                                                    // 슈퍼블록 읽기 함수
void read_group_descs(void);                                                   // 그룹 디스크립
```

터 배열 읽기 함수

```
void read_block(uint32_t blk, void *buf);                                                    // 주어진 블록 번호의 전체 블록을 buf에 읽음
void get_inode(uint32_t idx, struct ext2_inode *inode);                                      // 아이노드 정보 읽기
static void parse_dir_block(uint8_t *buf, DirEntry **head, DirEntry **tail);                 // 디렉터리 엔트리 파싱 헬퍼
DirEntry *get_dir_entries(uint32_t inode_no);                                                // 디렉터리 엔트리 목록 가져오기
void free_entries(DirEntry *head);                                                           // 디렉터리 엔트리 메모리 해제
void print_permissions(mode_t mode);                                                         // 권한 정보 출력
void tree_rec(uint32_t inode_no, int depth, int flag_r, int flag_s, int flag_p, int has_next[]); // 재귀적 트리 출력
uint32_t resolve_path(const char *path);                                                     // 경로 해석 함수
void tree_cmd(int argc, char **argv);                                                        // tree 명령 처리 함수
static void append_block(BufferNode **head, BufferNode **tail, uint8_t *buf, size_t len);    // 리스트에 새 노드 추가
static void free_blocks(BufferNode *head);                                                   // 리스트 메모리 해제
void print_cmd(int argc, char **argv);                                                       // print 명령 처리 함수
void help_cmd(int argc, char **argv);                                                        // help 명령 처리 함수


// 슈퍼블록 읽기 함수 구현
void read_superblock()
{
    // 1) 파일 오프셋을 1024바이트로 이동 (슈퍼블록 위치)
    if (lseek(image_fd, 1024, SEEK_SET) < 0)
    {
        perror("lseek superblock"); // 오류 메시지 출력
        exit(EXIT_FAILURE);             // 프로그램 종료
    }
    // 2) 슈퍼블록 구조체 크기만큼 데이터 읽기
    if (read(image_fd, &sb, sizeof(sb)) != sizeof(sb))
    {
        perror("read superblock"); // 오류 메시지 출력
        exit(EXIT_FAILURE);            // 프로그램 종료
    }
    // 매직 넘버 확인 (EXT2 파일 시스템인지 검증)
    if (sb.s_magic != EXT2_SUPER_MAGIC)
    {
        fprintf(stderr, "Not a valid ext2 filesystem₩n"); // 오류 메시지 출력
        exit(EXIT_FAILURE);                                // 프로그램 종료
    }
    // 블록 크기 계산 (1024 << s_log_block_size)
    block_size = 1024 << sb.s_log_block_size;
```

```
    inodes_per_group = sb.s_inodes_per_group;
    // 리비전 레벨에 따라 아이노드 크기 결정
    if (sb.s_rev_level == 0)
        inode_size = 256; // 레거시 아이노드 크기
    else
        inode_size = sb.s_inode_size; // 동적 아이노드 크기
    uint32_t blocks_per_group = sb.s_blocks_per_group;
    // 전체 블록 그룹 수 계산
    group_count = (sb.s_blocks_count + blocks_per_group - 1) / blocks_per_group;
}

// 그룹 디스크립터 배열 읽기
void read_group_descs(void)
{
    // 그룹 디스크립터 테이블의 오프셋 계산 (블록 크기에 따라 다름)
    off_t gd_offset = (block_size == 1024 ? 2 : 1) * block_size;
    if (lseek(image_fd, gd_offset, SEEK_SET) < 0)
    {
        perror("lseek group desc"); // 오류 메시지 출력
        exit(1);                    // 프로그램 종료
    }
    // 그룹 디스크립터 테이블 메모리 할당
    gd_table = malloc(group_count * sizeof(*gd_table));
    if (!gd_table)
    {
        perror("malloc group desc"); // 오류 메시지 출력
        exit(1);                     // 프로그램 종료
    }
    size_t len = group_count * sizeof(*gd_table);
    // 그룹 디스크립터 테이블 데이터 읽기
    if (read(image_fd, gd_table, len) != (ssize_t)len)
    {
        perror("read group descs"); // 오류 메시지 출력
        exit(1);                    // 프로그램 종료
    }
}

// 주어진 블록 번호의 전체 블록을 buf에 읽음
void read_block(uint32_t blk, void *buf)
{
    // 블록 오프셋 계산
    off_t off = (off_t)blk * block_size;
    if (lseek(image_fd, off, SEEK_SET) < 0)
```

```
    {
        perror("lseek block"); // 오류 메시지 출력
        exit(1);                    // 프로그램 종료
    }
    // 블록 데이터 읽기
    if (read(image_fd, buf, block_size) != (ssize_t)block_size)
    {
        perror("read block"); // 오류 메시지 출력
        exit(1);                   // 프로그램 종료
    }
}

// 특정 아이노드 정보 읽기 함수
void get_inode(uint32_t ino, struct ext2_inode *inode_out)
{
    // 아이노드가 속한 그룹 계산
    uint32_t group = (ino - 1) / inodes_per_group;
    // 그룹 내에서의 아이노드 인덱스 계산
    uint32_t index = (ino - 1) % inodes_per_group;
    // 아이노드 테이블의 시작 오프셋 계산
    off_t table = (off_t)gd_table[group].bg_inode_table * block_size;
    // 특정 아이노드의 오프셋 계산
    off_t off = table + (off_t)index * inode_size;
    if (lseek(image_fd, off, SEEK_SET) < 0)
    {
        perror("lseek inode"); // 오류 메시지 출력
        exit(1);                    // 프로그램 종료
    }
    // 아이노드 데이터 읽기
    if (read(image_fd, inode_out, sizeof(*inode_out)) != (ssize_t)sizeof(*inode_out))
    {
        perror("read inode"); // 오류 메시지 출력
        exit(1);                   // 프로그램 종료
    }
}

// 디렉터리 엔트리 파싱 헬퍼
static void parse_dir_block(uint8_t *buf, DirEntry **head, DirEntry **tail)
{
    off_t off = 0;
    // 블록 내의 모든 디렉터리 엔트리 처리
    while (off < block_size)
    {
```

```
        struct ext2_dir_entry *de = (void *)(buf + off);
        if (de->inode) // 유효한 아이노드 번호인 경우
        {
            char name[EXT2_NAME_LEN + 1];
            memcpy(name, de->name, de->name_len); // 이름 복사
            name[de->name_len] = '₩0';                  // 널 종료
            // 특수 디렉터리 (., .., lost+found) 제외
            if (strcmp(name, ".") && strcmp(name, "..") && strcmp(name, "lost+found"))
            {
                // 새로운 디렉터리 엔트리 생성
                DirEntry *e = malloc(sizeof(*e));
                memcpy(e->name, de->name, de->name_len);
                e->name[de->name_len] = '₩0';
                e->inode = de->inode;
                struct ext2_inode tmp;
                get_inode(e->inode, &tmp); // 아이노드 정보 읽기
                unsigned long long tmp_full_size = (unsigned long long)tmp.i_size | ((unsigned long long)tmp.i_dir_acl
<< 32);
                e->size = tmp_full_size;
                e->mode = tmp.i_mode;        // 파일 모드 저장
                e->next = NULL;
                // 연결 리스트에 추가
                if (!*head)
                    *head = *tail = e; // 첫 번째 엔트리인 경우
                else
                {
                    (*tail)->next = e;
                    *tail = e;
                }
            }
        }
        if (de->rec_len < 8) // 레코드 길이가 유효하지 않으면 종료
            break;
        off += de->rec_len; // 다음 엔트리로 이동
    }
}

// 모든 direct/indir blocks 파싱
DirEntry *get_dir_entries(uint32_t ino)
{
    struct ext2_inode node;
    uint8_t *buf = malloc(block_size); // 블록 데이터 버퍼
    DirEntry *head = NULL, *tail = NULL;
    // 아이노드가 속한 그룹 계산
```

```
uint32_t group = (ino - 1) / inodes_per_group;
uint32_t idx = (ino - 1) % inodes_per_group;
// 아이노드 테이블의 시작 오프셋 계산
off_t table = (off_t)gd_table[group].bg_inode_table * block_size;
off_t ino_off = table + idx * inode_size;
if (lseek(image_fd, ino_off, SEEK_SET) < 0)
    exit(1);
if (read(image_fd, &node, sizeof(node)) != sizeof(node))
    exit(1);

uint32_t ptrs = block_size / sizeof(uint32_t);
uint32_t *ind = malloc(block_size); // 간접 블록 버퍼

// 직접 블록 처리 (0-11)
for (int i = 0; i < 12; i++)
{
    if (node.i_block[i]) // 블록이 할당된 경우
    {
        read_block(node.i_block[i], buf);    // 블록 데이터 읽기
        parse_dir_block(buf, &head, &tail); // 디렉터리 엔트리 파싱
    }
}
// 단일 간접 블록 처리 (12)
if (node.i_block[12])
{
    read_block(node.i_block[12], ind); // 간접 블록 읽기
    for (uint32_t i = 0; i < ptrs && ind[i]; i++)
    {
        read_block(ind[i], buf);                // 실제 데이터 블록 읽기
        parse_dir_block(buf, &head, &tail); // 디렉터리 엔트리 파싱
    }
}
// 이중 간접 블록 처리 (13)
if (node.i_block[13])
{
    read_block(node.i_block[13], ind); // 이중 간접 블록 읽기
    for (uint32_t i = 0; i < ptrs && ind[i]; i++)
    {
        uint32_t *ind2 = malloc(block_size);
        read_block(ind[i], ind2); // 단일 간접 블록 읽기
        for (uint32_t j = 0; j < ptrs && ind2[j]; j++)
        {
            read_block(ind2[j], buf);              // 실제 데이터 블록 읽기
```

```
                parse_dir_block(buf, &head, &tail); // 디렉터리 엔트리 파싱
            }
            free(ind2);
        }
    }
    // 삼중 간접 블록 처리 (14)
    if (node.i_block[14])
    {
        read_block(node.i_block[14], ind); // 삼중 간접 블록 읽기
        for (uint32_t i = 0; i < ptrs && ind[i]; i++)
        {
            uint32_t *ind2 = malloc(block_size);
            read_block(ind[i], ind2); // 이중 간접 블록 읽기
            for (uint32_t j = 0; j < ptrs && ind2[j]; j++)
            {
                uint32_t *ind3 = malloc(block_size);
                read_block(ind2[j], ind3); // 단일 간접 블록 읽기
                for (uint32_t k = 0; k < ptrs && ind3[k]; k++)
                {
                    read_block(ind3[k], buf);              // 실제 데이터 블록 읽기
                    parse_dir_block(buf, &head, &tail); // 디렉터리 엔트리 파싱
                }
                free(ind3);
            }
            free(ind2);
        }
    }
    free(ind);
    free(buf);
    return head; // 파싱된 디렉터리 엔트리 목록 반환
}

// 디렉터리 엔트리 메모리 해제 함수
void free_entries(DirEntry *head)
{
    while (head)
    {
        DirEntry *tmp = head; // 현재 노드 임시 저장
        head = head->next;     // 다음 노드로 이동
        free(tmp);                // 현재 노드 메모리 해제
    }
}
```

```
// 권한 정보 출력 함수
void print_permissions(mode_t mode)
{
    char perms[11]; // 권한 문자열 버퍼 (10자리 + 널 종료)

    // 파일 유형 설정
    perms[0] = (S_ISDIR(mode) ? 'd' : '-'); // 디렉터리 여부

    // 사용자 권한 설정
    perms[1] = (mode & S_IRUSR) ? 'r' : '-'; // 읽기 권한
    perms[2] = (mode & S_IWUSR) ? 'w' : '-'; // 쓰기 권한
    perms[3] = (mode & S_IXUSR) ? 'x' : '-'; // 실행 권한

    // 그룹 권한 설정
    perms[4] = (mode & S_IRGRP) ? 'r' : '-';
    perms[5] = (mode & S_IWGRP) ? 'w' : '-';
    perms[6] = (mode & S_IXGRP) ? 'x' : '-';

    // 기타 사용자 권한 설정
    perms[7] = (mode & S_IROTH) ? 'r' : '-';
    perms[8] = (mode & S_IWOTH) ? 'w' : '-';
    perms[9] = (mode & S_IXOTH) ? 'x' : '-';

    perms[10] = '₩0';       // 문자열 종료
    printf("%s ", perms); // 권한 문자열 출력
}

// 재귀적 트리 출력 함수
void tree_rec(uint32_t inode_no, int depth, int flag_r, int flag_s, int flag_p, int has_next[])
{
    DirEntry *list = get_dir_entries(inode_no); // 디렉터리 엔트리 목록 가져오기
    int total = 0;

    // 엔트리 총 개수 계산
    for (DirEntry *t = list; t; t = t->next)
        total++;

    int idx = 0; // 현재 인덱스 초기화
    for (DirEntry *p = list; p; p = p->next)
    {
        idx++;

        // 트리 구조를 위한 접두사 출력
```

```
for (int d = 0; d < depth; d++)
{
    if (has_next[d])
        printf("│   "); // 수직선 출력
    else
        printf("    "); // 공백 출력
}

// 현재 항목의 분기 표시
if (idx < total)
{
    printf("├── "); // 중간 항목
    has_next[depth] = 1;
}
else
{
    printf("└── "); // 마지막 항목
    has_next[depth] = 0;
}

// 옵션에 따른 추가 정보 출력
if (flag_p || flag_s)
{
    printf("[");
    if (flag_p)
        print_permissions(p->mode); // 권한 정보 출력
    if (flag_s)
        printf("%lld", (long long)p->size); // 파일 크기 출력
    printf("] ");
}

printf("%s₩n", p->name); // 파일/디렉터리 이름 출력

// 카운트 및 재귀 처리
if (S_ISDIR(p->mode))
{                       // 디렉터리인 경우
    dir_count++; // 디렉터리 카운트 증가
    if (flag_r)
    {                                                                   // 재귀 옵션이 활성화된 경우
        tree_rec(p->inode, depth + 1, flag_r, flag_s, flag_p, has_next); // 재귀 호출
    }
}
else
```

```
        {
            file_count++; // 파일 카운트 증가
        }
    }
    free_entries(list); // 엔트리 목록 메모리 해제
}

// 경로 해석 함수
uint32_t resolve_path(const char *path)
{
    if (!path || !*path)
        return 0; // 유효하지 않은 경로
    if (strcmp(path, ".") == 0)
        return 2; // 현재 디렉터리 (루트 아이노드 2)

    char tmp[1024];
    strncpy(tmp, path, sizeof(tmp)); // 경로 복사
    tmp[sizeof(tmp) - 1] = '₩0';       // 널 종료 보장

    // "./" 접두사 제거
    while (!strncmp(tmp, "./", 2))
        memmove(tmp, tmp + 2, strlen(tmp + 2) + 1);

    char *token = strtok(tmp, "/"); // 첫 번째 토큰 분리
    uint32_t cur = 2;                    // 루트 아이노드로 시작

    while (token)
    { // 모든 경로 구성 요소 처리
        struct ext2_inode di;
        get_inode(cur, &di); // 현재 아이노드 정보 가져오기

        if (!S_ISDIR(di.i_mode))
            return 0; // 디렉터리가 아니면 실패

        DirEntry *list = get_dir_entries(cur); // 현재 디렉터리 엔트리 가져오기
        DirEntry *p = list;
        uint32_t next = 0;

        // 일치하는 이름 찾기
        while (p)
        {
            if (!strcmp(p->name, token))
            {
```

```
                next = p->inode;
                break;
            }
            p = p->next;
        }

        free_entries(list); // 엔트리 목록 메모리 해제

        if (!next)
            return 0; // 일치하는 항목 없음

        cur = next;                    // 다음 아이노드로 이동
        token = strtok(NULL, "/"); // 다음 토큰
    }
    return cur; // 최종 아이노드 반환
}

// tree 명령 처리 함수
void tree_cmd(int argc, char **argv)
{
    if (argc < 2)
    {
        printf("Usage: tree <PATH> [OPTION]...₩n");
        return;
    }

    // 옵션 플래그 초기화
    int flag_r = 0, flag_s = 0, flag_p = 0;

    // 명령행 인수 처리
    for (int i = 2; i < argc; i++)
    {
        if (argv[i][0] == '-')
        {
            for (int j = 1; argv[i][j]; j++)
            {
                if (argv[i][j] == 'r')
                    flag_r = 1; // 재귀 옵션
                else if (argv[i][j] == 's')
                    flag_s = 1; // 크기 표시 옵션
                else if (argv[i][j] == 'p')
                    flag_p = 1; // 권한 표시 옵션
                else
```

```
            {
                char *help_argv[] = {"help", "tree", NULL};
                help_cmd(2, help_argv);
                return;
            }
        }
    }
}

uint32_t ino = resolve_path(argv[1]); // 경로 해석
if (!ino)
{
    // 존재하지 않는 경로일 때 help_cmd 호출
    char *help_argv[] = {"help", "tree", NULL};
    help_cmd(2, help_argv);
    return;
}

// 아이노드 읽어서 디렉터리인지 확인
struct ext2_inode di;
get_inode(ino, &di);
// large_file(rev1) 모드: 상위 32비트는 i_dir_acl 에 저장됨
unsigned long long full_size = (unsigned long long)di.i_size | ((unsigned long long)di.i_dir_acl << 32);

if (!S_ISDIR(di.i_mode))
{
    // 전체 경로가 아닌 파일명만 출력
    const char *path = argv[1];
    const char *name = strrchr(path, '/');
    if (name)
        name++; // '/' 다음 문자부터
    else
        name = path;
    printf("Error: '%s' is not directory₩n", name);
    return;
}

// 카운터 및 상태 배열 초기화
dir_count = 1; // 루트 디렉터리 포함
file_count = 0;
int has_next[256] = {0}; // 트리 구조 상태 추적 배열

// 옵션에 따른 추가 정보 출력
```

```
    if (flag_p || flag_s)
    {
        printf("[");
        if (flag_p)
            print_permissions(di.i_mode); // 권한 정보 출력
        if (flag_s)
            printf("%llu", full_size); // 파일 크기 출력
        printf("] ");
    }
    printf("%s₩n", argv[1]);                                    // 루트 경로 출력
    tree_rec(ino, 0, flag_r, flag_s, flag_p, has_next); // 재귀적 트리 출력

    // 요약 정보 출력
    printf("₩n%d directories, %d files₩n", dir_count, file_count);
}

// 리스트에 새 노드 추가
static void append_block(BufferNode **head, BufferNode **tail, uint8_t *buf, size_t len) {
    BufferNode *node = malloc(sizeof(BufferNode));
    node->data = buf;  // 이미 malloc된 buf 포인터 그대로 사용
    node->len  = len;
    node->next = NULL;
    if (!*head) {
        *head = *tail = node;
    } else {
        (*tail)->next = node;
        *tail = node;
    }
}

// 리스트 메모리 해제
static void free_blocks(BufferNode *head) {
    while (head) {
        BufferNode *tmp = head;
        head = head->next;
        free(tmp->data);
        free(tmp);
    }
}

// print 명령 처리 함수
void print_cmd(int argc, char **argv)
{
```

```
int flag_n = 0, n = 0;          // -n 옵션 사용 여부 저장 플래그(flag_n)와 출력할 줄 수(n) 초기화

// 3) -n 옵션 파싱 (argc == 4일 때만)
if (argc == 4) {
    // argv[2]는 반드시 "-n"
    if (strcmp(argv[2], "-n") != 0) {
        printf("print: option requires an argument -- 'n'₩n");  // 오류 메시지 출력
        return;
    }
    flag_n = 1;                     // -n 사용 표시
    n = atoi(argv[3]);              // 출력할 줄 수 변환
    if (n <= 0) {                   // 유효한 숫자 검사
        printf("print: invalid number of lines: %s₩n", argv[3]);
        return;
    }
}

// 옵션 파싱 후 남은 인자가 없으면 에러 처리
if (3 == argc)
{
    printf("print: option requires an argument -- 'n'₩n"); // 파일 경로 인자가 없음을 알리는 메시지 출력
    return;                         // 함수 종료
}

const char *path = argv[1];  // 남은 인자에서 파일 경로(path) 지정
uint32_t ino = resolve_path(path); // resolve_path를 호출해 경로에 대한 아이노드 번호(ino) 계산
if (!ino)                           // 아이노드 번호가 0이면 경로가 유효하지 않음
{
    // 존재하지 않는 파일 입력시 Usage 출력력
    char *help_argv[] = {"help", "print", NULL};
    help_cmd(2, help_argv);
    return;                         // 함수 종료
}

// 아이노드 구조체 읽기 준비
struct ext2_inode node;                                         // ext2_inode 구조체 변수 node 선언
uint32_t group = (ino - 1) / inodes_per_group;                  // 아이노드가 속한 그룹 번호 계산
uint32_t idx = (ino - 1) % inodes_per_group;                    // 그룹 내에서의 아이노드 인덱스 계산
off_t table = (off_t)gd_table[group].bg_inode_table * block_size;// 그룹 디스크립터에서 아이노드 테이블 시작 블록
계산
off_t off = table + idx * inode_size;                           // 아이노드 위치 오프셋 계산
if (lseek(image_fd, off, SEEK_SET) < 0)                         // 파일 디스크립터를 아이노드 위치로 이동
{
    perror("lseek inode"); return;                              // 오류 시 perror로 메시지 출력 후 함수 종료
```

```
}
if (read(image_fd, &node, sizeof(node)) != sizeof(node))          // 아이노드 정보를 node에 읽기
{
    perror("read inode"); return;                                  // 오류 시 perror로 메시지 출력 후 함수 종료
}

// 디렉터리인지 확인: 디렉터리면 파일이 아니므로 에러 처리
if (S_ISDIR(node.i_mode)) {                                        // inode 모드가 디렉터리인지 검사
    printf("Error: '%s' is not file₩n", path);                  // 디렉터리임을 알리는 메시지 출력
    return;                                                        // 함수 종료
}

// 읽은 블록을 담을 연결 리스트 초기화
BufferNode *head = NULL, *tail = NULL; // 연결 리스트의 헤드(head)와 꼬리(tail) 포인터 초기화
uint8_t *buf;                                  // 블록 데이터를 읽어올 임시 버퍼 포인터 buf 선언
uint32_t ptrs = block_size / sizeof(uint32_t); // 블록 당 포인터 수 계산 (간접 포인터 처리용)
uint32_t *ind = NULL;                          // 간접 포인터 블록을 읽어들일 포인터 ind 선언
int lines_printed = 0, done = 0;          // 출력된 줄 수(lines_printed)와 완료 여부(done) 초기화

// Direct blocks 처리: i_block[0..11]
for (int i = 0; i < 12 && !done; i++)
{
    if (!node.i_block[i]) continue;      // 블록 번호가 0이면 건너뛰기
    buf = malloc(block_size);              // block_size 크기로 메모리 할당
    read_block(node.i_block[i], buf);   // 해당 블록 번호의 데이터를 buf로 읽기
    size_t len = block_size;               // 읽은 데이터 길이는 block_size
    append_block(&head, &tail, buf, len);// 연결 리스트에 새 노드로 추가
}
// Single indirect 블록 처리 (i_block[12])
if (!done && node.i_block[12])
{
    ind = malloc(block_size);              // 간접 포인터 블록용 메모리 할당
    read_block(node.i_block[12], ind); // 포인터 블록 읽기
    for (uint32_t i = 0; i < ptrs && ind[i]; i++)
    {
        buf = malloc(block_size);         // 데이터 블록용 메모리 할당
        read_block(ind[i], buf);          // 포인터가 가리키는 데이터 블록 읽기
        append_block(&head, &tail, buf, block_size); // 리스트에 추가
    }
    free(ind);                                 // 간접 포인터 블록 메모리 해제
}
// Double indirect 블록 처리 (i_block[13])
if (!done && node.i_block[13])
```

```
{
    ind = malloc(block_size);              // 1차 간접 포인터 블록 메모리 할당
    read_block(node.i_block[13], ind); // 1차 포인터 블록 읽기
    for (uint32_t i = 0; i < ptrs && ind[i]; i++)
    {
        uint32_t *ind2 = malloc(block_size);// 2차 포인터 블록 메모리 할당
        read_block(ind[i], ind2);             // 2차 포인터 블록 읽기
        for (uint32_t j = 0; j < ptrs && ind2[j]; j++)
        {
            buf = malloc(block_size);       // 데이터 블록용 메모리 할당
            read_block(ind2[j], buf);       // 2차 포인터가 가리키는 데이터 블록 읽기
            append_block(&head, &tail, buf, block_size); // 리스트에 추가
        }
        free(ind2);                           // 2차 포인터 블록 해제
    }
    free(ind);                                // 1차 포인터 블록 해제
}
// Triple indirect 블록 처리 (i_block[14])
if (!done && node.i_block[14])
{
    ind = malloc(block_size);              // 1차 포인터 블록 메모리 할당
    read_block(node.i_block[14], ind); // 1차 포인터 블록 읽기
    for (uint32_t i = 0; i < ptrs && ind[i]; i++)
    {
        uint32_t *ind2 = malloc(block_size);// 2차 포인터 블록 메모리 할당
        read_block(ind[i], ind2);             // 2차 포인터 블록 읽기
        for (uint32_t j = 0; j < ptrs && ind2[j]; j++)
        {
            uint32_t *ind3 = malloc(block_size);// 3차 포인터 블록 메모리 할당
            read_block(ind2[j], ind3);      // 3차 포인터 블록 읽기
            for (uint32_t k = 0; k < ptrs && ind3[k]; k++)
            {
                buf = malloc(block_size);   // 데이터 블록용 메모리 할당
                read_block(ind3[k], buf);   // 3차 포인터가 가리키는 데이터 블록 읽기
                append_block(&head, &tail, buf, block_size); // 리스트에 추가
            }
            free(ind3);                       // 3차 포인터 블록 해제
        }
        free(ind2);                           // 2차 포인터 블록 해제
    }
    free(ind);                                // 1차 포인터 블록 해제
}
```

```
    // 연결 리스트 순회하며 실제 출력 수행
    for (BufferNode *p = head; p; p = p->next)
    {
        for (size_t i = 0; i < p->len; i++)
        {
            putchar(p->data[i]);              // 버퍼의 데이터를 한 바이트씩 출력
            if (flag_n && p->data[i] == '₩n' && ++lines_printed >= n) // -n 옵션이 켜져 있고, 출력 줄 수가 n에 도
달하면
            {
                done = 1;                     // 출력 완료 플래그 설정
                break;                        // 내부 루프 탈출
            }
        }
        if (done) break;                      // 완료 플래그 확인 후 외부 루프 탈출
    }

    free_blocks(head);                        // 연결 리스트 및 모든 버퍼 메모리 해제
}

// help 명령 처리 함수
void help_cmd(int argc, char **argv)
{
    // 사용법 문자열
    const char *usage =
        "Usage:₩n"
        "> tree <PATH> [OPTION]... : display the directory structure if <PATH> is a directory₩n"
        "   -r : display the directory structure recursively if <PATH> is a directory₩n"
        "   -s : display the directory structure if <PATH> is a directory, including the size of each file₩n"
        "   -p : display the directory structure if <PATH> is a directory, including the permissions of each directory
and file₩n"
        "> print <PATH> [OPTION]... : print the contents on the standard output if <PATH> is file₩n"
        "   -n <line_number> : print only the first <line_number> lines of its contents on the standard output if
<PATH> is file₩n"
        "> help [COMMAND] : show commands for program₩n"
        "> exit : exit program₩n";

    if (argc == 1)
    {
        printf("%s", usage); // 기본 도움말 출력
        return;
    }

    // 특정 명령에 대한 도움말
    if (!strcmp(argv[1], "tree"))
    {
```

```
        printf("Usage: tree <PATH> [OPTION]...₩n");
        printf("Display the directory structure if <PATH> is a directory₩n");
        printf("Options:₩n");
        printf("   -r : display the directory structure recursively₩n");
        printf("   -s : display the directory structure including the size of each file₩n");
        printf("   -p : display the directory structure including the permissions of each directory and file₩n");
    }
    else if (!strcmp(argv[1], "print"))
    {
        printf("Usage: print <PATH> [OPTION]...₩n");
        printf("Print the contents on the standard output if <PATH> is file₩n");
        printf("Options:₩n");
        printf("   -n <line_number> : print only the first <line_number> lines of its contents₩n");
    }
    else if (!strcmp(argv[1], "help"))
    {
        printf("Usage: help [COMMAND]₩n");
        printf("Show commands for program₩n");
    }
    else if (!strcmp(argv[1], "exit"))
    {
        printf("Usage: exit₩n");
        printf("Exit program₩n");
    }
    else
    {
        printf("invalid command -- '%s'₩n", argv[1]);
        printf("%s", usage); // 잘못된 명령 시 기본 도움말 출력
    }
}

// 메인 함수
int main(int argc, char *argv[])
{
    if (argc != 2)
    {
        fprintf(stderr, "Usage Error : %s <EXT2_IMAGE>₩n", argv[0]);
        exit(EXIT_FAILURE);
    }

    // EXT2 이미지 파일 열기
    image_fd = open(argv[1], O_RDONLY);
    if (image_fd < 0)
```

```
{
    perror("open image");
    exit(EXIT_FAILURE);
}

read_superblock(); // 슈퍼블록 읽기
read_group_descs();

// 대화형 셸 구현
char *line = NULL;
size_t len = 0;
while (1)
{
    printf("%s> ", SSU_ID); // 프롬프트 출력

    // 사용자 입력 읽기
    if (getline(&line, &len, stdin) <= 0)
        break;
    if (line[0] == '₩n')
        continue; // 빈 줄 무시

    line[strcspn(line, "₩n")] = '₩0'; // 개행 문자 제거

    // 명령어 분할
    char *argv2[16];
    int argc2 = 0;
    char *tok = strtok(line, " ");
    while (tok && argc2 < 16)
    {
        argv2[argc2++] = tok;
        tok = strtok(NULL, " ");
    }

    if (argc2 == 0)
        continue; // 빈 명령 무시

    // 명령어 분기 처리
    if (!strcmp(argv2[0], "exit"))
        break; // 종료
    else if (!strcmp(argv2[0], "help"))
        help_cmd(argc2, argv2);
    else if (!strcmp(argv2[0], "tree"))
        tree_cmd(argc2, argv2);
```

```
        else if (!strcmp(argv2[0], "print"))
            print_cmd(argc2, argv2);
        else
            help_cmd(1, argv2); // 알 수 없는 명령어 도움말 출력
    }

    close(image_fd); // 파일 디스크립터 닫기
    return 0;
}
```