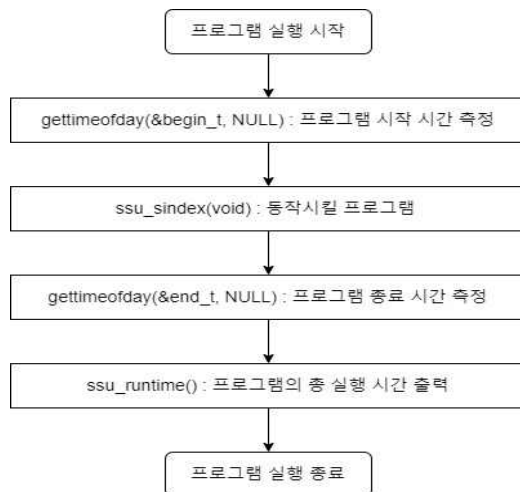


1. 개요

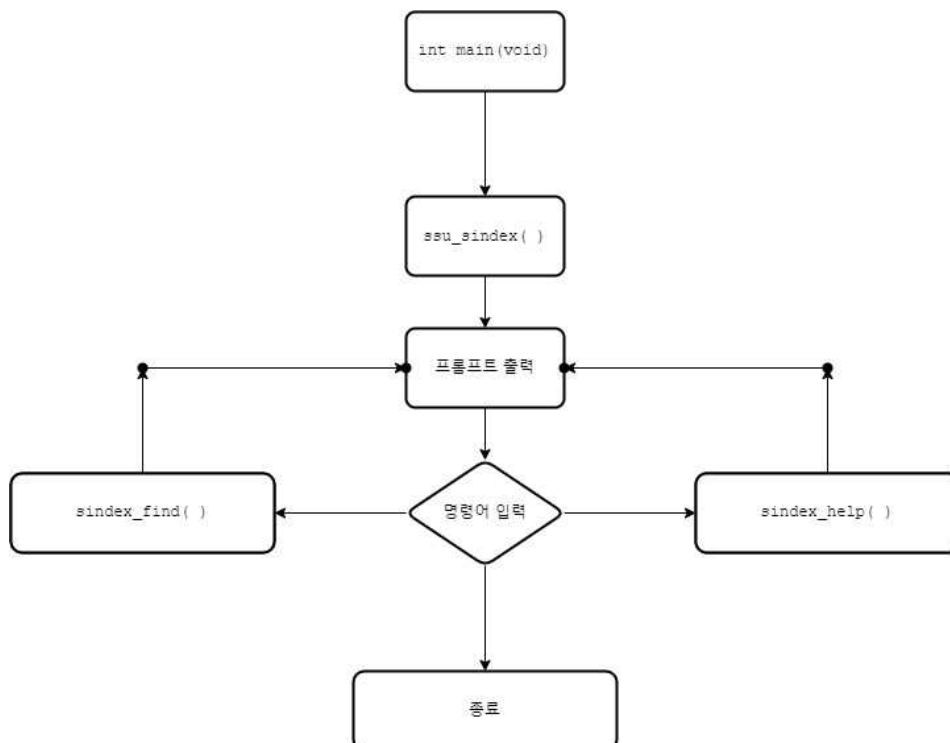
본 프로그램의 ssu_index는 주어진 디렉토리 내에서 지정한 파일 또는 디렉토리 파일과 이름 및 크기가 동일한 파일을 탐색하고 그 중 하나를 골라 그 내용을 비교하는 프로그램이다.

2. 상세 설계

1) main.c : ssu_index 프로그램의 개괄적인 수행



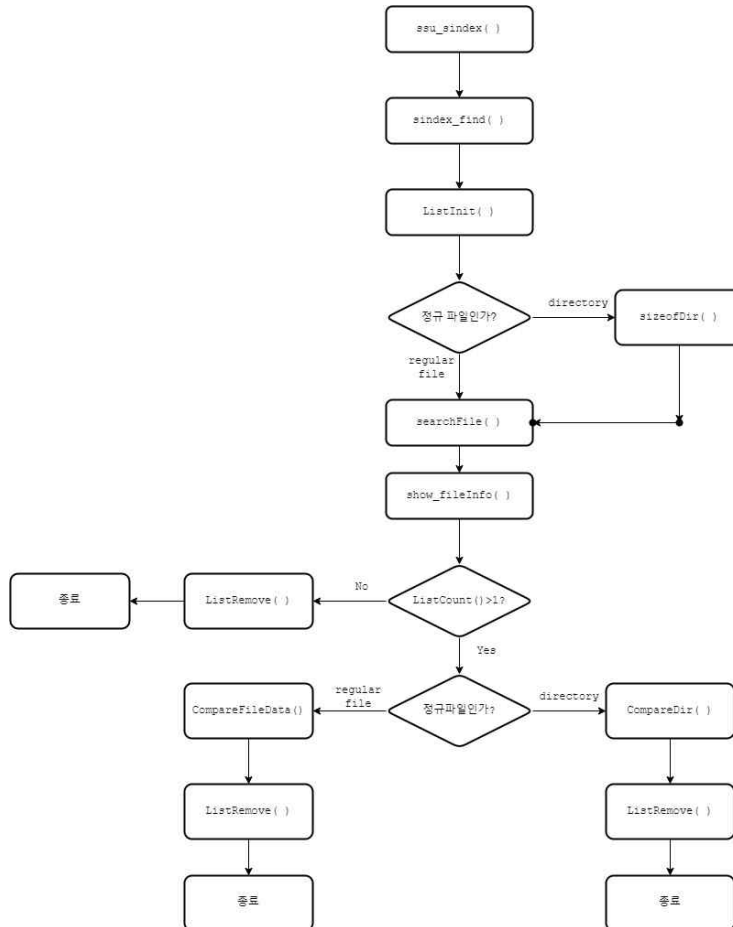
2) ssu_index.c : ssu_index 프로그램의 구체적인 기능 수행



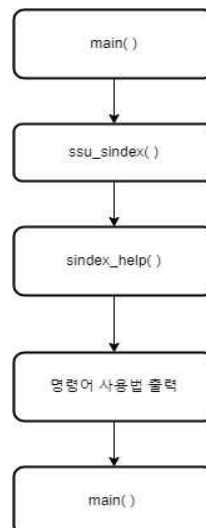
2. 상세 설계

3) 명령어 동작 관련

a) `sindex_find()` : 파일 탐색, 리스트 생성·조희·삭제, 데이터 간 비교 수행

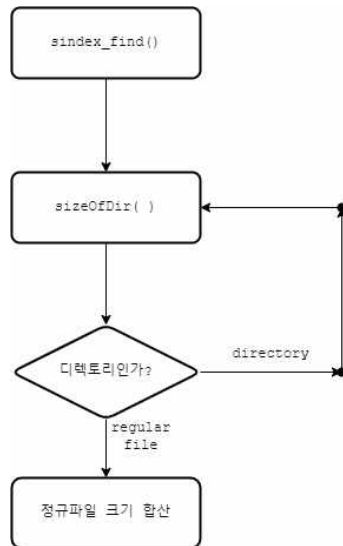


b) `sindex_help()` : 프로그램 사용법 출력

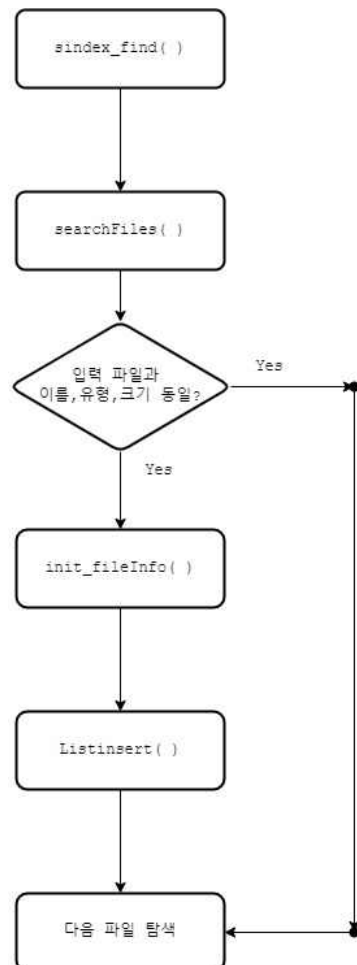


4) 파일 탐색 관련

a) `sizeofDir()` : 디렉토리의 크기 구하는 함수



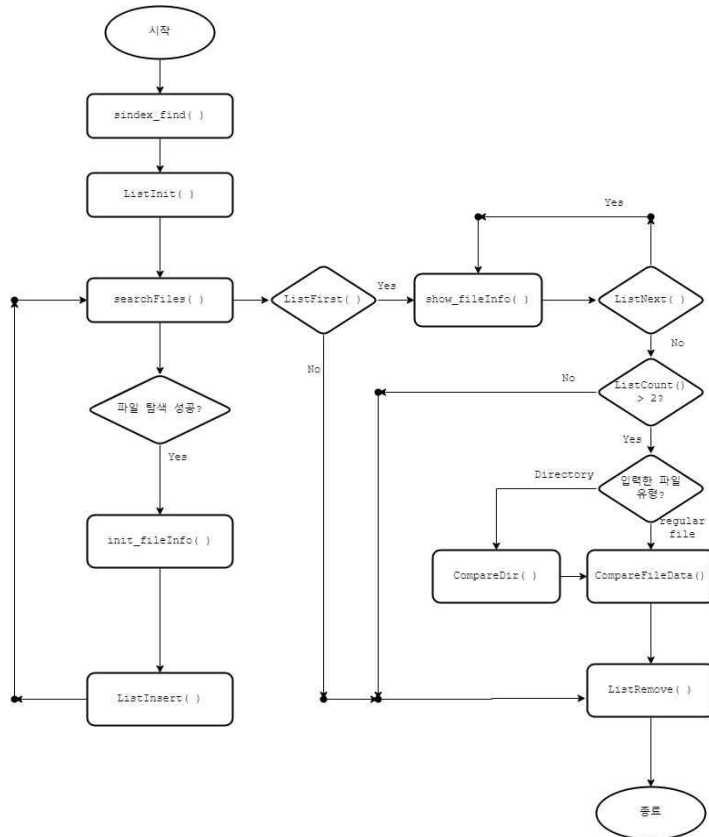
b) `searchFiles()` : `find` 명령어에서의 파일 탐색 수행



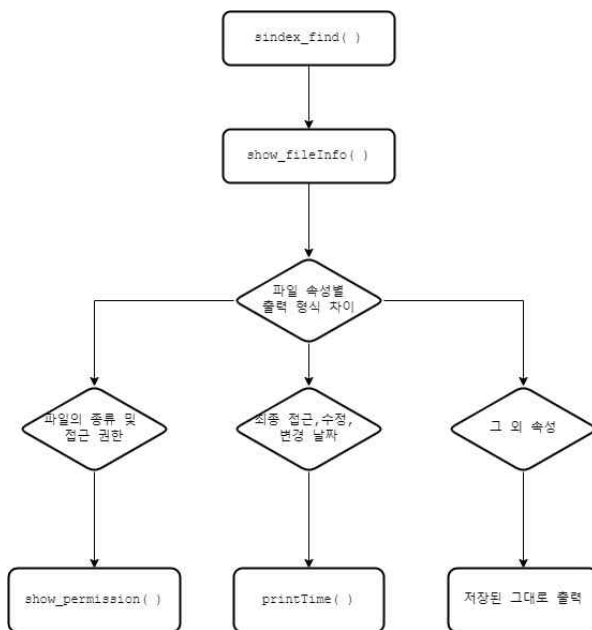
2. 상세 설계

5) 리스트 데이터 삽입, 조회, 삭제

a) 리스트 기능 중심 `sindex_find()`

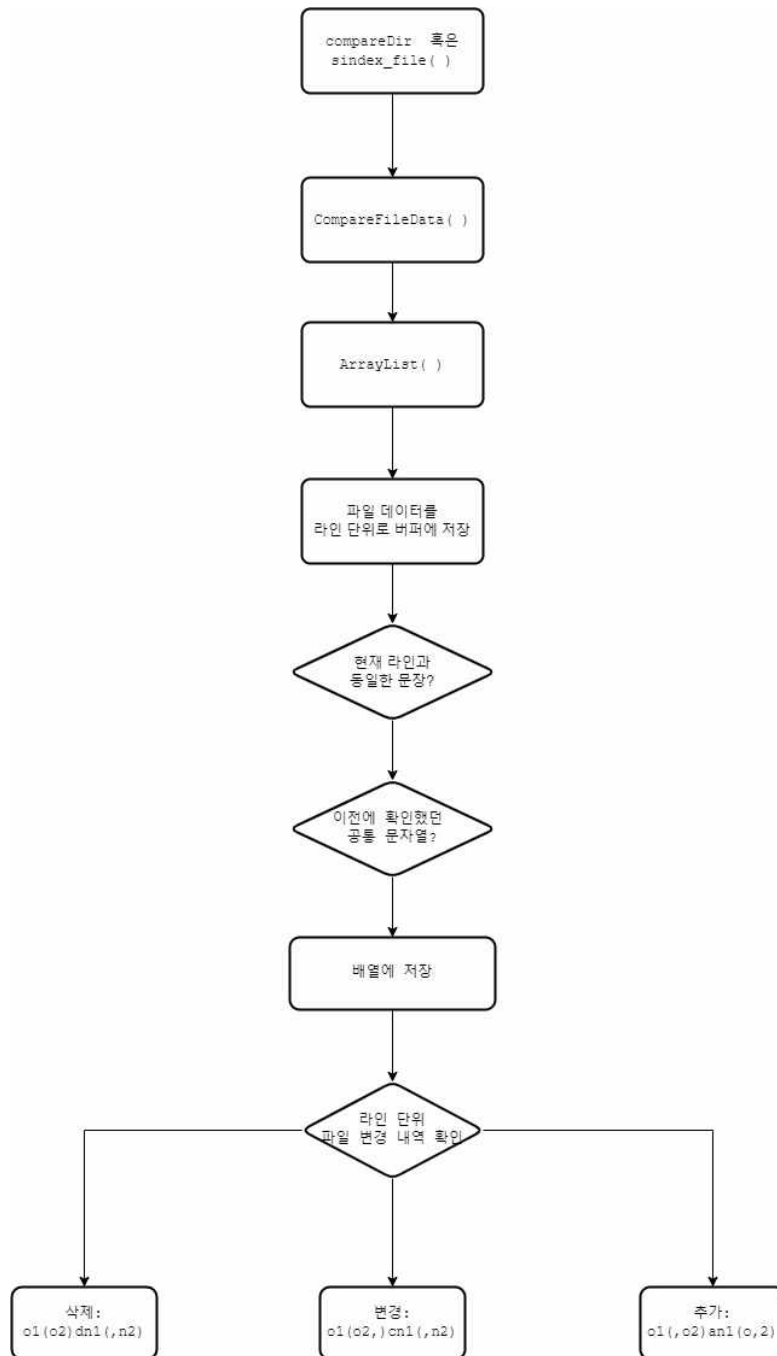


b) `show_fileInfo()` 함수 출력 관련

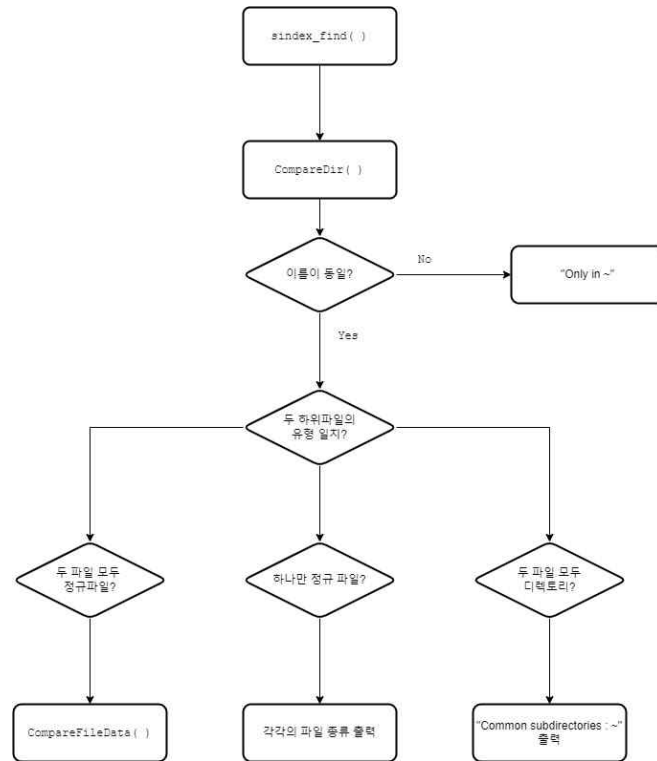


6) 파일/디렉토리 간 비교 관련

a) CompareFileData() : 정규 파일 간 차이 비교 수행(필수기능만 구현)



b) CompareDir() : 디렉토리 간 차이 비교 수행(필수기능만 구현)



3. 구현 방법 설명

1) main.c

a) void ssu_sindex(void);

ssu_sindex 프로그램을 실행하기 위한 모든 기능을 포함하는 함수

b) ssu_runtime(struct timeval *begin_t, struct timeval *end_t);

프로그램 실행 종료 시간에서 실행 시작 시간을 뺀 총 실행 시간을 마이크로초 단
위까지 계산하여 출력하는 함수

2) ssu_sindex.c

a) void ssu_help(void);

help 명령어 입력 시 명령어 사용법을 출력하는 함수

b) void ssu_find(char *filename, char *pathname);

find 명령어 입력 시 두 번째 인자가 되는 경로명을 시작점으로 하여 첫 번째 인자
로 사용된 파일 탐색을 수행하기 위한 함수

3) 파일 탐색 관련

a) long int sizeOfDir(long int *psize, char *filename, int depth);

재귀를 통해 인자로 지정한 디렉토리 파일 및 그 하위 디렉토리 내부의 정규파일들
의 크기의 합을 구한 후, 그 결과값을 디렉토리의 크기로 리턴한다.

b) void searchFiles(List * plist, Info * pinfo, off_t * psize, char *curDir, int depth);

find 명령어의 PATH에 해당하는 디렉토리 경로를 인자로 삼아 그 하위 디렉토리
들의 정규/디렉토리 파일 중 입력한 파일명과 이름 및 크기가 동일한 파일을 재귀적
으로 탐색한다.

4) 검색된 파일의 속성 정보 관련

a) void init_fileInfo(Info * pinfo, char *filename, long int size);

인자로 지정한 파일의 크기, 종류 및 접근 권한, 할당된 블록 수, 하드링크 수, 사
용자/그룹 ID, 최종 접근, 수정, 변경 시간, 절대 경로를 저장한다.

b) void show_permission(mode_t * pmode);

리스트 형태로 파일의 정보를 출력할 때, 파일의 종류 및 접근 권한 값을 출력하기
위해 파일의 mode 정보를 문자열 형태로 변환한다.

c) void show_fileInfo(List * plist, Info * pinfo);

init_fileInfo()에서 저장된 파일의 속성 정보를 출력하기 위한 함수이다.

d) void printTime(time_t * time);

파일의 최종 접근, 수정, 변경 시간을 나타내기 위한 연, 월, 일, 시, 분 형식을 설
정하여 출력한다.

5) 리스트 생성, 삽입, 조회, 삭제 관련 함수

a) void ListInit(List * plist);

searchFiles() 함수를 호출하여 입력한 파일과 이름 및 크기가 동일한 파일을 찾았
을 때, 이 파일의 정보를 저장하기 위해 리스트의 주소값을 인자로 할당하여 초기화
한다. 이 때 첫 번째 데이터와 그 이후 데이터 간 작업의 일관성을 위해 head 노드

는 더미 노드 형태로 초기화하고 리스트 내 데이터 개수 값을 0으로 초기화한다.

b) void ListInsert(List * plist, LData data);

리스트에 새로운 데이터를 저장하기 위한 노드를 생성하고, 매개변수 data에 해당하는 값을 새로 생성한 노드에 저장한다. 이 때, 새 노드는 항상 연결리스트의 head 노드의 바로 다음 노드부터 추가될 수 있다.

c) int ListFirst(List * plist, LData * pdata);

리스트에 저장된 데이터에 대한 조회를 시작할 지점을 head 노드의 바로 다음 노드로 설정한다. 그리고 리스트 내의 데이터를 활용할 수 있도록 설정된 지점에 해당하는 노드의 데이터를 매개변수 pdata가 가리키는 메모리에 저장하여 사용한다. 그리고 pdata로 읽어들이 노드의 유무에 따라 TRUE/FALSE 값을 리턴한다.

d) int ListNext(List * plist, LData *pdata);

현재 참조한 노드의 바로 다음 노드의 데이터를 매개변수 pdata가 가리키는 메모리에 저장한다. 이 함수는 저장된 노드의 총 개수 혹은 사용자가 입력한 인덱스 번호만큼 반복 호출하는 방식을 통해 리스트에 저장된 파일 탐색 내역에 대한 조회를 수행한다. 또한 노드의 유무에 따라 TRUE/FALSE 중에서 반환값이 설정된다.

e) LData ListRemove(List * plist);

검색 작업의 일관성을 유지하기 위해 함수 sindex_find()에서의 모든 작업을 완료하고 main 함수의 ssu_sindex()에서 새로운 명령어를 입력하기 전, 리스트에 저장된 데이터를 모두 삭제하는 작업을 수행한다. 이때, 반환값은 ListFirst()나 ListNext() 함수를 통해 조회한 가장 최근의 노드에 저장된 데이터이다.

f) int ListCount(List * plist);

리스트에 저장된 노드의 총 개수를 반환한다. 이 노드는 리스트의 0번 인덱스의 파일과 비교할 파일의 인덱스 번호를 입력할 때 입력한 인덱스 번호가 유효한지를 판별하기 위한 용도로 활용된다.

6) 파일 내용 비교작업 수행용 함수

a) void CompareDir(char *origDir, char *compDir);

find 명령어 입력 시 FILENAME 인자에 해당하는 입력된 파일이 디렉토리인 경우, 이 함수의 첫 번째 인자를 비교 기준, 두 번째 인자를 비교할 디렉토리로 설정한다. 이후 두 디렉토리 하위 파일들을 알파벳 순으로 정렬하여 두 디렉토리 간 동일한 이름의 파일이 있는지를 비교하는 작업을 수행한다.

b) void CompareFileData(char *criteria, char *comparefile);

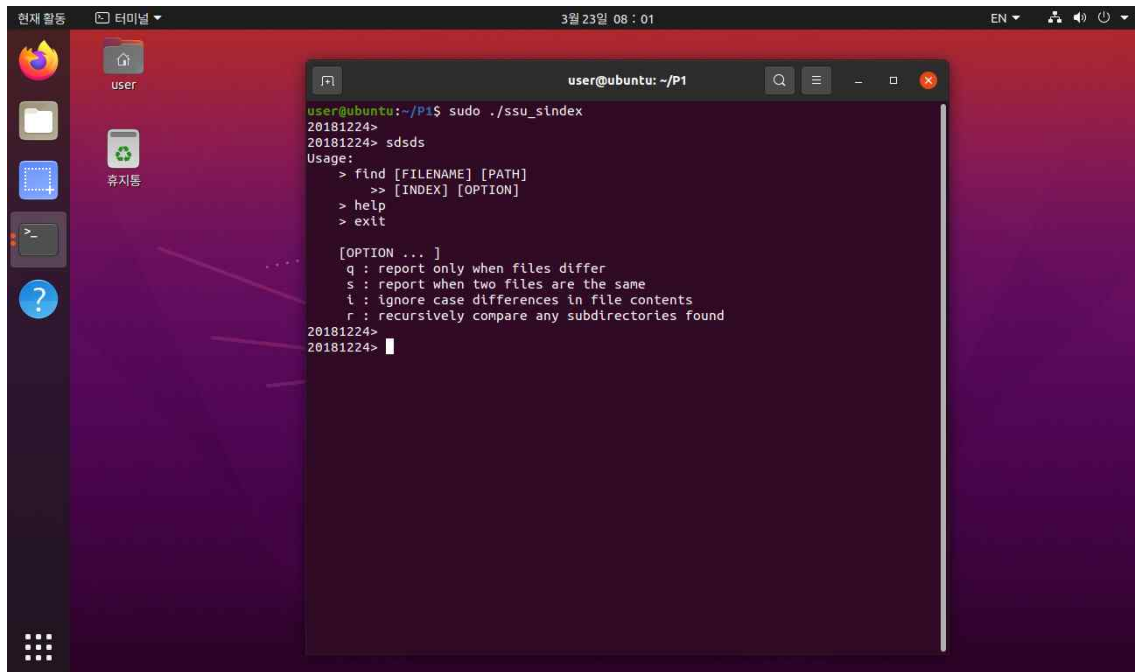
FILENAME 인자로 입력된 파일이 정규 파일인 경우 첫 번째 인자가 되는 원본의 데이터를 두 번째 인자가 되는 인덱스의 파일의 데이터와 비교한다. 이 때 원본과 비교본 간의 데이터 비교는 라인 단위로 수행되며, 데이터 비교 후 원본의 데이터의 추가, 삭제, 변경 내역을 출력한다.

c) void ArrayInit(char (*arr)[BUFFER_SIZE]);

CompareFileData() 함수에서 FILENAME과 INDEX에 해당하는 파일의 데이터를 라인 단위로 저장하기 위한 배열을 초기화한다.

4. 실행 결과

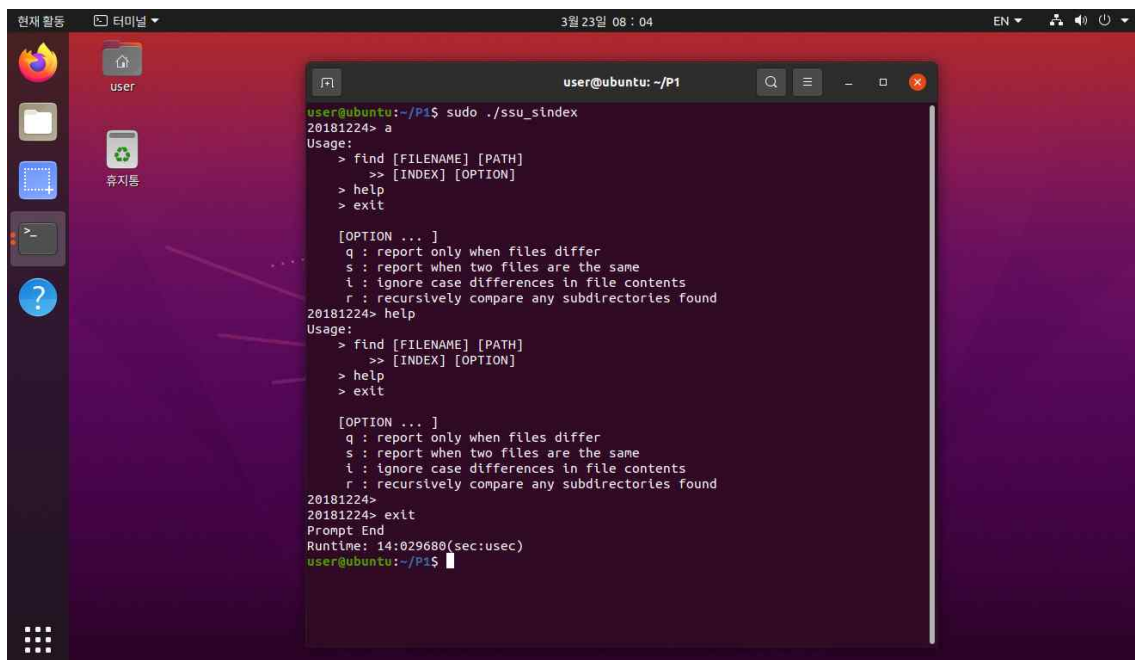
1) 프롬프트 출력



```
user@ubuntu: ~/P1
user@ubuntu:~/P1$ sudo ./ssu_index
20181224>
20181224> sdsds
Usage:
> find [FILENAME] [PATH]
>> [INDEX] [OPTION]
> help
> exit

[OPTION ... ]
q : report only when files differ
s : report when two files are the same
i : ignore case differences in file contents
r : recursively compare any subdirectories found
20181224>
20181224>
```

2) 내장명령어 exit, help 명령어 수행 결과



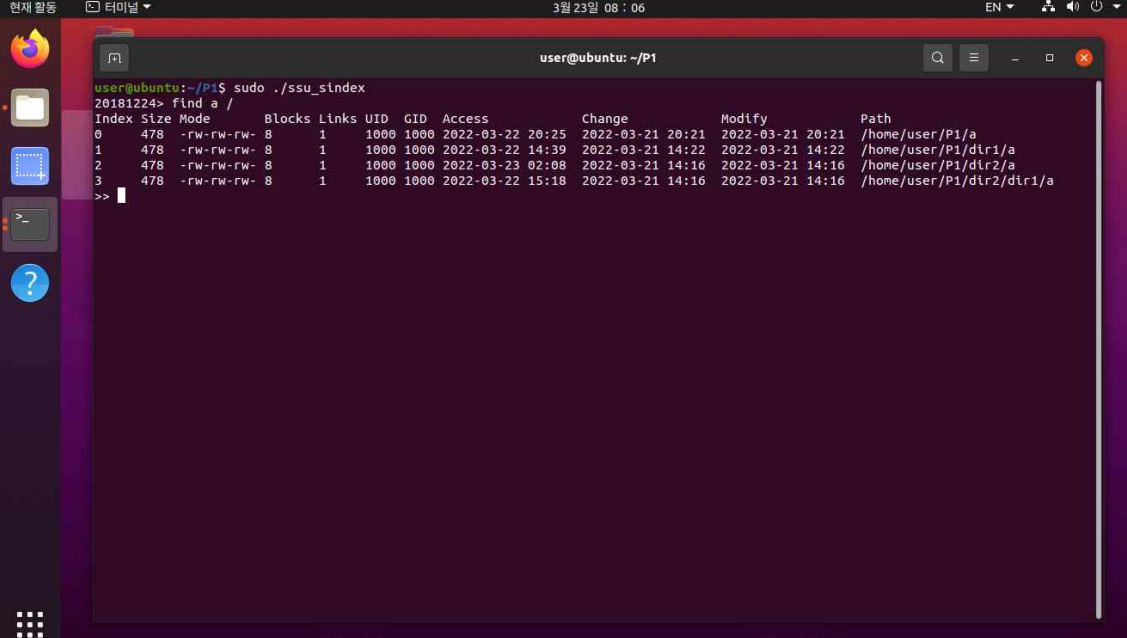
```
user@ubuntu: ~/P1
user@ubuntu:~/P1$ sudo ./ssu_index
20181224> a
Usage:
> find [FILENAME] [PATH]
>> [INDEX] [OPTION]
> help
> exit

[OPTION ... ]
q : report only when files differ
s : report when two files are the same
i : ignore case differences in file contents
r : recursively compare any subdirectories found
20181224> help
Usage:
> find [FILENAME] [PATH]
>> [INDEX] [OPTION]
> help
> exit

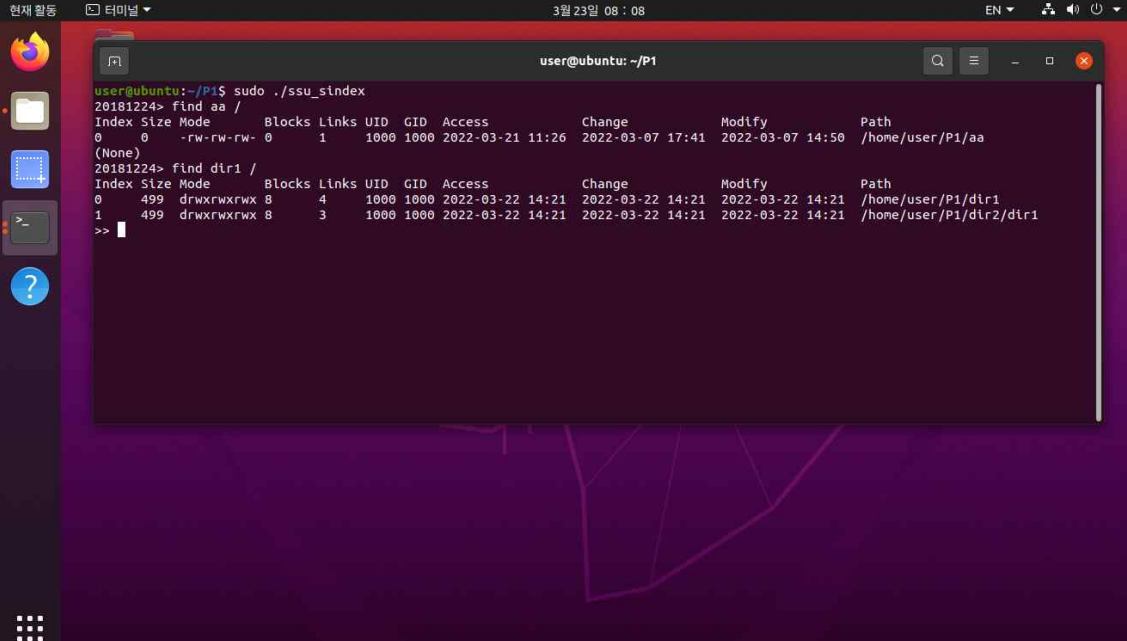
[OPTION ... ]
q : report only when files differ
s : report when two files are the same
i : ignore case differences in file contents
r : recursively compare any subdirectories found
20181224>
20181224> exit
Prompt End
Runtime: 14:029680(sec:usec)
user@ubuntu:~/P1$
```

3) 내장명령어 file 명령어 수행 결과

a) 파일 탐색 결과 출력



```
user@ubuntu:~/P1$ sudo ./ssu_index
20181224> find a /
Index Size Mode      Blocks Links UID  GID Access      Change      Modify      Path
0    478 -rw-rw-rw-    8      1 1000 1000 2022-03-22 20:25 2022-03-21 20:21 2022-03-21 20:21 /home/user/P1/a
1    478 -rw-rw-rw-    8      1 1000 1000 2022-03-22 14:39 2022-03-21 14:22 2022-03-21 14:22 /home/user/P1/dir1/a
2    478 -rw-rw-rw-    8      1 1000 1000 2022-03-23 02:08 2022-03-21 14:16 2022-03-21 14:16 /home/user/P1/dir2/a
3    478 -rw-rw-rw-    8      1 1000 1000 2022-03-22 15:18 2022-03-21 14:16 2022-03-21 14:16 /home/user/P1/dir2/dir1/a
>>
```



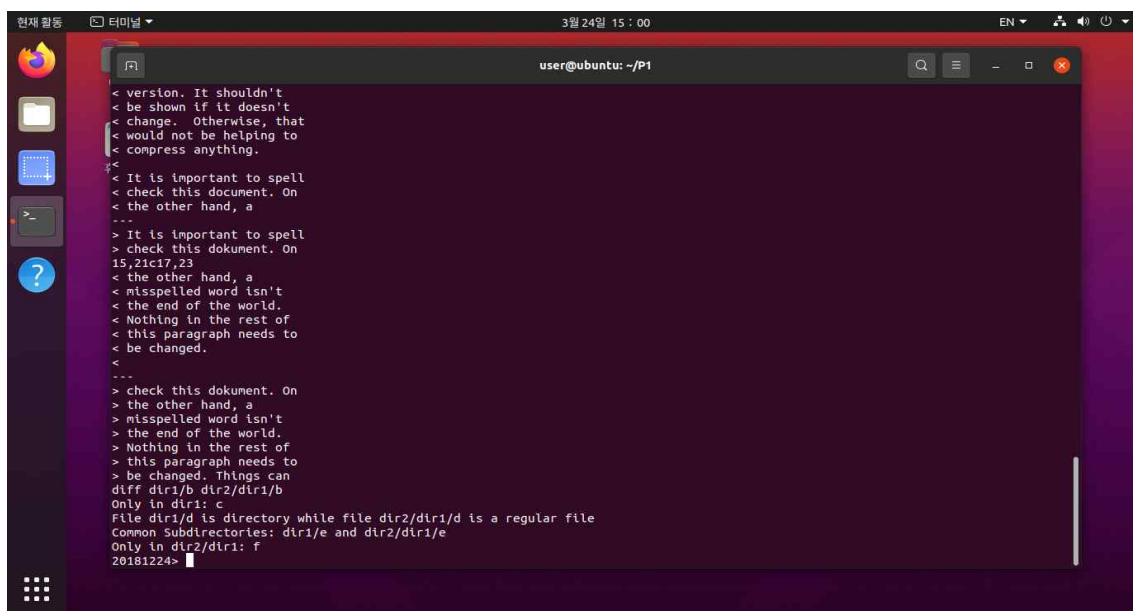
```
user@ubuntu:~/P1$ sudo ./ssu_index
20181224> find aa /
Index Size Mode      Blocks Links UID  GID Access      Change      Modify      Path
0      0 -rw-rw-rw-    0      1 1000 1000 2022-03-21 11:26 2022-03-07 17:41 2022-03-07 14:50 /home/user/P1/aa
(None)
20181224> find dir1 /
Index Size Mode      Blocks Links UID  GID Access      Change      Modify      Path
0    499 drwxrwxrwx    8      4 1000 1000 2022-03-22 14:21 2022-03-22 14:21 2022-03-22 14:21 /home/user/P1/dir1
1    499 drwxrwxrwx    8      3 1000 1000 2022-03-22 14:21 2022-03-22 14:21 2022-03-22 14:21 /home/user/P1/dir2/dir1
>>
```

b) 정규 파일 비교 결과 출력

```
user@ubuntu:~/P1$ sudo ./ssu_sindex
20181224> find a /
Index Size Mode      Blocks Links UID  GID Access      Change      Modify      Path
0      478 -rw-rw-rw- 8        1 1000 1000 2022-03-23 20:27 2022-03-21 20:21 2022-03-21 20:21 /home/user/P1/a
1      478 -rw-rw-rw- 8        1 1000 1000 2022-03-23 20:27 2022-03-21 14:22 2022-03-21 14:22 /home/user/P1/dir1/a
2      478 -rw-rw-rw- 8        1 1000 1000 2022-03-24 12:47 2022-03-21 14:16 2022-03-21 14:16 /home/user/P1/dir2/a
3      478 -rw-rw-rw- 8        1 1000 1000 2022-03-23 20:27 2022-03-21 14:16 2022-03-21 14:16 /home/user/P1/dir2/dir1/a
>> 1
1,4d0
> This part of the
> document has stayed the
> same from version to
> version. It shouldn't
< changes.
<
< This paragraph contains
< text that is outdated.
...
> compress anything.
>
> It is important to spell
11,17c14,15
< text that is outdated.
< It will be deleted in the
< near future.
<
< It is important to spell
< check this dokument. On
< the other hand, a
...
> It is important to spell
> check this document. On
17,23c15,21
< the other hand, a
```

c) 디렉토리 비교 결과 출력

```
20181224> find dir1 /
Index Size Mode      Blocks Links UID  GID Access      Change      Modify      Path
0      499 drwxrwxrwx 8        4 1000 1000 2022-03-23 20:27 2022-03-22 14:21 2022-03-22 14:21 /home/user/P1/dir1
1      499 drwxrwxrwx 8        3 1000 1000 2022-03-23 20:27 2022-03-22 14:21 2022-03-22 14:21 /home/user/P1/dir2/dir1
>> 1
diff dir1/a dir2/dir1/a
1d0
> This is an important
1,5c8,11
< notice! It should be
< located at this document!
<
< This part of the
< document has stayed the
...
> compress the size of the
> changes.
>
> This paragraph contains
5,24c11,12
< document has stayed the
< same from version to
< version. It shouldn't
< be shown if it doesn't
< change. Otherwise, that
< would not be helping to
< compress anything.
<
< It is important to spell
< check this document. On
< the other hand, a
< misspelled word isn't
< the end of the world.
< Nothing in the rest of
```



5. 소스코드

1) main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/time.h>
#include "ssu_sindex.h"

#define SEC_TO_MICRO 1000000

void ssu_runtime(struct timeval *begin_t, struct timeval *end_t);

int main(void)
{
    struct timeval begin_t, end_t;

    gettimeofday(&begin_t, NULL); // 프로그램 시작 시간

    ssu_sindex();                  // ssu_sindex 프로그램 실행

    gettimeofday(&end_t, NULL);    // 프로그램 종료시간
    ssu_runtime(&begin_t, &end_t); // 프로그램 실행 시간 출력 후 종료
    exit(0);
}

void ssu_runtime(struct timeval *begin_t, struct timeval *end_t)
{
    // 초단위 계산
    end_t->tv_sec -= begin_t->tv_sec;

    // 마이크로초단위 자리내림
    if (end_t->tv_usec < begin_t->tv_usec) {
        end_t->tv_sec--;
        end_t->tv_usec += SEC_TO_MICRO;
    }

    // 마이크로초단위 계산
    end_t->tv_usec -= begin_t->tv_usec;
    printf("Runtime: %ld:%06ld(sec:usec)\n", end_t->tv_sec, end_t->tv_usec);
}
```

2) ssu_sindex.h

```
#ifndef MAIN_H_
#define MAIN_H_

#include <sys/stat.h>
#include <sys/types.h>
#include <dirent.h>
#include <time.h>

#ifndef TRUE
#define TRUE 1
#endif

#ifndef FALSE
#define FALSE 0
#endif

#ifndef CMD_LEN
#define CMD_LEN 64 // 내장명령문의 최대 길이
#endif

#ifndef TOKEN_CNT
#define TOKEN_CNT 3// 문자열 파싱 시 생성되는 토큰 개수
#endif

#ifndef FILE_MAX
#define FILE_MAX 256 // 파일명 최대 길이
#endif

#ifndef PATH_MAX
#define PATH_LEN 4096 // 경로 최대 길이
#endif

#ifndef PER_LEN
#define PER_LEN 11 // 파일 접근권한 출력용 문자열 길이(show_permission 함수)
#endif

#ifndef BUFFER_SIZE
#define BUFFER_SIZE 1024
#endif

typedef struct _record
{
    int index;
    off_t size;
    mode_t modes;
    blkcnt_t blockCnt;
}
```

```

nlink_t linkCnt;
uid_t uid;
gid_t gid;
time_t access;
time_t change;
time_t modify;
char path[PATH_MAX];
} Info;

typedef Info * LData; // 리스트에 저장할 데이터 타입

typedef struct _node // 연결리스트용 노드 구조체
{
    LData data;
    struct _node * next;
} Node;

typedef struct _linkedlist
{
    Node * head;
    Node * tail;
    Node * cur;
    Node * before;
    int numOfData;
    int (*comp)(char *path1, char *path2);
} LinkedList;

typedef LinkedList List;

// ssu_index 기본 기능
void ssu_index(void); // ssu_index 프로그램 실행 함수
void index_find(char *filename, char *pathname); // find 명령어 수행
void index_help(void); // help 명령어 수행

// 지정한 파일 탐색 관련 함수
long int sizeofDir(long int *psize, char *filename, int depth);
void searchFiles(List * plist, Info * pinfo, off_t * psize, char *curDir, int depth);

// 검색된 파일 정보 데이터 변경, 출력 관련
int init_fileInfo(Info * pinfo, char *filename, long int size); // 파일 속성 초기화/저장
void show_permission(mode_t * pmode); // 파일 종류 및 접근 권한
정보 출력
void show_fileInfo(List * plist, Info * pinfo); // 파일 속성 출력
void printTime(time_t * time); // 연, 월, 일, 시, 분 출력

// find 명령어의 탐색결과 출력용 리스트 관련 함수
void ListInit(List * plist); // 리스트 초기화
void ListInsert(List * plist, LData data); // 리스트 행 추가

```

```

int ListFirst(List * plist, LData * pdata); // 리스트 탐색 시작지점 데이터 불러오기
int ListNext(List * plist, LData *pdata); // 다음 리스트 이동 및 데이터 불러오기
LData ListRemove(List * plist);           // 리스트 행 삭제
int ListCount(List * plist);               // 리스트 전체 행 개수

// 파일 내용 비교작업 수행용 함수
void CompareDir(char *origDir, char *compDir);
void CompareFileData(char *criteria, char *comparefile);
void ShowDiff(int (*common)[BUFFER_SIZE], char (*orig)[BUFFER_SIZE], char
(*comp)[BUFFER_SIZE], int len1, int len2);
void ArrayInit(char (*arr)[BUFFER_SIZE]);
#endif

```

3) ssu_index.c

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>
#include <ctype.h>
#include "ssu_index.h"

char fileName[FILE_MAX] = {0,}; // FILENAME 인자로 입력한 파일명
char filePath[PATH_MAX] = {0,}; // FILENAME 인자에 해당하는 파일의 절대경로

char cwd[PATH_MAX] = {0, }; // 현재 작업 중인 디렉토리 경로
char startPath[PATH_MAX] = {0, }; // 입력한 탐색 시작할 경로

char compareFile[PATH_MAX] = {0, }; // FILENAME과 비교할 파일
의 절대경로
char origBuf[BUFFER_SIZE][BUFFER_SIZE] = {0, }; // 원본 파일의 각 라
인별 데이터 저장
char compBuf[BUFFER_SIZE][BUFFER_SIZE] = {0, }; // 비교본 파일의 각
라인별 데이터 저장
int LCS[BUFFER_SIZE][BUFFER_SIZE] = {0, }; // 원본과 비교본의 현재 라인
기준 일치하는 라인의 최대 개수

// 프롬프트 출력 및 명령어 find, exit, help 실행하는 함수
void ssu_index(void)
{
    struct stat statbuf;
    char command[CMD_LEN] = {0, };
    char *tokens[TOKEN_CNT] = {NULL, };
    int token_cnt;
    char *ptr;
    FILE *fp;

```



```

DIR *dirp;

while (1) {
    fileName[0] = 0;
    filePath[0] = 0;
    cwd[0] = 0;
    startPath[0] = 0;

    // 프롬프트 출력 및 명령어 입력
    printf("20181224> ");
    fgets(command, sizeof(command), stdin);
    // 엔터키만 입력한 경우: 프롬프트 재출력
    if (strcmp(command, "\n") == 0) {
        command[strlen(command)-1] = '\0';
        continue;
    }
    command[strlen(command)-1] = '\0';

    // 입력한 명령어에서 문자열 파싱
    ptr = strtok(command, " \n");
    token_cnt = 0;
    while (ptr != NULL) {
        tokens[token_cnt] = ptr;
        token_cnt++;
        ptr = strtok(NULL, " \n");
    }

    if (strcmp(tokens[0], "find") == 0) { // find 명령어 실행
        // FILENAME, PATH 입력 여부 점검
        if (token_cnt < TOKEN_CNT) {
            fprintf(stderr, "Usage: find [FILENAME] [PATH]\n");
            continue;
        }
        // FILENAME의 파일 존재 여부 확인
        realpath(tokens[1], filePath); // 입력된 경로/파일명을 절대
경로로 변환

        if (lstat(filePath, &statbuf) < 0) {
            fprintf(stderr, "%s is not exist\n", filePath);
            continue;
        }
        if (S_ISREG(statbuf.st_mode) || S_ISDIR(statbuf.st_mode)) {
            // 파일명 추출
            // FILENAME의 경로 표시 방식에 관계없이 동일한 결과
            내기 위함

            getcwd(cwd, PATH_MAX);
            ptr = filePath;
            ptr += strlen(cwd)+1; // 절대경로에서 파일명만 추출
            strcpy(fileName, ptr);

```

```

    }
    // PATH 인자의 경로에 해당하는 디렉토리 존재 여부 확인
    realpath(tokens[2], startPath);
    if (lstat(startPath, &statbuf) < 0) {
        fprintf(stderr, "%s is not exist\n", startPath);
        continue;
    }
    if (!S_ISDIR(statbuf.st_mode))
        continue;

    sindex_find(fileName, startPath);
    chdir(cwd);
}
else if (strcmp(tokens[0], "exit") == 0) {    // exit 명령어 실행
    puts("Prompt End");
    return;
}
else {    // help 및 기타 명령어 실행
    sindex_help();
}

}

}
// help 명령어 수행 함수
void sindex_help(void)
{
    printf("Usage:\n");
    printf("    > find [FILENAME] [PATH]\n");
    printf("Wt>> [INDEX] [OPTION]\n");
    printf("    > help\n");
    printf("    > exit\n");
    printf("    [OPTION ... ]\n");
    printf("    q : report only when files differ\n");
    printf("    s : report when two files are the same\n");
    printf("    i : ignore case differences in file contents\n");
    printf("    r : recursively compare any subdirectories found\n");
    return;
}

// find 명령어 수행 함수
void sindex_find(char *filename, char *pathname)
{
    struct stat statbuf;
    off_t size = 0;
    char curDir[PATH_MAX]; // 현재 디렉토리 경로
    int depth = 0;
    int index = 0;
    char subCmd[CMD_LEN];
    List list;

```

```

Info * pinfo;

lstat(filename, &statbuf);
// 리스트 초기화
ListInit(&list);

// [FILENAME]의 파일을 [PATH]를 시작디렉토리로 하여 탐색
if (S_ISREG(statbuf.st_mode)) {
    size = statbuf.st_size;
    searchFiles(&list, pinfo, &size, pathname, depth);
}
else if (S_ISDIR(statbuf.st_mode)) {
    sizeOfDir(&size, filePath, depth);
    searchFiles(&list, pinfo, &size, pathname, depth);
}

// 리스트 출력
printf("%5s %4s %-10s %-6s %-5s %-4s %-4s %-16s %-16s %-16s %-30s",
        "Index", "Size", "Mode", "Blocks", "Links", "UID", "GID",
        "Access", "Change", "Modify", "Path");

if (ListFirst(&list, &pinfo)) {
    printf("%-5d ", index++);
    show_fileInfo(&list, pinfo);

    while (ListNext(&list, &pinfo)) {
        printf("%-5d ", index++);
        show_fileInfo(&list, pinfo);
    }
}

// 출력된 리스트 중 하나와 비교
if (ListCount(&list) < 2) { // 0번 인덱스 외 다른 파일 없음
    printf("(None)\n");
}
else {
    // 0번 인덱스 외 다른 파일 존재
    int listNum;

    while (1) {
        int i = 0;
        listNum = 0;

        printf(">> ");
        fgets(subCmd, sizeof(subCmd), stdin);
        // 엔터키만 입력한 경우: 프롬프트 재출력
        if (strcmp(subCmd, "\n") == 0) {
            subCmd[strlen(subCmd)-1] = '\0';
            fprintf(stderr, "index input is null\n");
        }
    }
}

```

```

        continue;
    }
    // 숫자 입력인 경우
    while (isdigit(subCmd[i])) {
        listNum = 10 * listNum + subCmd[i++] - 48;
        // 검색된 인덱스 범위 초과 여부 확인
        if (listNum >= ListCount(&list)) {
            listNum = -1 ;
            break;
        }
    }

    subCmd[strlen(subCmd)-1] = 'W0';
    if (listNum == -1) { // 인덱스 범위 초과
        fprintf(stderr, "index number is not existWn");
        continue;
    }
    else if (listNum == 0) { // 숫자 아닌 문자 입력
        fprintf(stderr, "index is not numberWn");
        continue;
    }
    else // 범위 내 인덱스 번호 입력
        break;
}
// 입력한 인덱스 번호에 해당하는 파일 정보 불러오기
if (ListFirst(&list, &pinfo)) {
    for (int i = 1; i <= listNum; i++)
        ListNext(&list, &pinfo);
    strcpy(compareFile, pinfo->path);
}

// 원본과 비교본 간 내용 비교
if (S_ISREG(statbuf.st_mode)) {
    if (statbuf.st_size >= BUFFER_SIZE) // 파일 크기가 1024
        바이트 이상이면 비교 없음
        printf("%s is too big!Wn", fileName);
    else
        CompareFileData(filePath, pinfo->path);
}
else if (S_ISDIR(statbuf.st_mode)) {
    CompareDir(filePath, pinfo->path);
}
}

// 전체 작업 종료 후 저장된 리스트 모두 삭제
if (ListFirst(&list, &pinfo)) {
    pinfo = ListRemove(&list);
    free(pinfo);
}

```

```

        while (ListNext(&list, &pinfo)) {
            pinfo = ListRemove(&list);
            free(pinfo);
        }
    }

    return;
}

// 지정한 디렉토리 파일의 크기 구하는 함수
long int sizeOfDir(long int *psize, char *curDir, int depth)
{
    struct dirent *entry;
    struct stat statbuf;
    DIR *dir;
    int fd;
    long int size = 0;

    if ((dir = opendir(curDir)) == NULL, chdir(curDir) < 0) {
        fprintf(stderr, "opendiri, chdir error for %sWn", curDir);
        exit(1);
    }

    while ((entry = readdir(dir)) != NULL) {
        char newDir[PATH_MAX];
        int len;

        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..") ==
0)
            continue;

        lstat(entry->d_name, &statbuf);
        if (entry->d_type == DT_DIR && S_ISDIR(statbuf.st_mode)) {
            len = snprintf(newDir, sizeof(newDir)-1, "%s/%s", curDir,
entry->d_name);
            newDir[len] = 0;
            size += sizeOfDir(psize, newDir, depth+1);
        }
        else if (entry->d_type == DT_REG) {
            len = snprintf(newDir, sizeof(newDir)-1, "%s/%s", curDir,
entry->d_name);
            newDir[len] = 0;
            if ((fd = open(entry->d_name, O_RDONLY)) < 0) {
                fprintf(stderr, "file open error for %sWn",
entry->d_name);
                continue;
            }

```

```

        size += lseek(fd, 0, SEEK_END);
        close(fd);
    }
}

*psize = size;
closedir(dir);
chdir("..");
return *psize;
}

// find 명령어 세부작업 1: 입력한 파일명에 해당하는 파일 검색
void searchFiles(List * plist, Info * pinfo, off_t *psize, char *path, int depth)
{
    DIR *dir;
    struct dirent **nameList;
    struct stat statbuf[2];    // 0번 인덱스는 FILENAME, 1번 인덱스는 현재 탐색중인
파일
    char curDir[PATH_MAX];
    int nameCount;
    int idx = 0;              // nameList의 인덱스 번호
    long int dirSize;
    int dirDepth;

    realpath(path, curDir);
    // 파일 오픈
    if ((dir = opendir(curDir)) == NULL || chdir(curDir) < 0) {
        fprintf(stderr, "opendir error for %sWn", curDir);
        return;
    }
    // 현재 디렉토리의 전체 파일 목록 불러오기(알파벳순 정렬)
    if ((nameCount = scandir(curDir, &nameList, NULL, alphasort) < 0))
    {
        fprintf(stderr, "%s: readdir error and scandir errorWn", curDir);
        return;
    }

    // 파일 탐색
    while (nameList[idx] = readdir(dir))
    {
        char newPath[PATH_MAX];
        int len;
        // 절대 경로로 변환
        len = snprintf(newPath, sizeof(newPath)-1, "%s/%s", curDir,
nameList[idx]->d_name);
        newPath[len] = 0;
        // 정규 파일인 경우
        lstat(newPath, &statbuf[0]);
        lstat(nameList[idx]->d_name, &statbuf[1]);
    }
}

```

```

        if (nameList[idx]->d_type == DT_REG) {
            // 상대경로/절대경로가 동일한 파일인 경우
            len = snprintf(newPath, sizeof(newPath)-1, "%s/%s", curDir,
nameList[idx]->d_name);
            newPath[len] = 0;
            if ((strcmp(fileName, nameList[idx]->d_name) == 0 ||
strcmp(filePath, newPath) == 0)) {
                // 리스트에 추가
                if (*psize == statbuf[1].st_size) {
                    pinfo = (Info *)malloc(sizeof(Info));
                    init_fileInfo(pinfo, newPath, *psize);
                    ListInsert(plist, pinfo);
                    break;
                }
            }
        }
        else if (nameList[idx]->d_type == DT_DIR) { // 디렉토리 파일인
경우
            // 현재 디렉토리를 경로에 추가
            if (strcmp(nameList[idx]->d_name, ".") == 0 ||
strcmp(nameList[idx]->d_name, "..") == 0)
                continue;
            if (strcmp(nameList[idx]->d_name, "run") == 0 ||
strcmp(nameList[idx]->d_name, "proc") == 0)
                continue;
            if (strcmp(fileName, nameList[idx]->d_name) == 0 ||
strcmp(filePath, newPath) == 0) {
                len = snprintf(newPath, sizeof(newPath)-1, "%s/%s",
curDir, nameList[idx]->d_name);
                newPath[len] = 0;
                dirSize = 0;
                dirDepth = 0;
                if (S_ISDIR(statbuf[0].st_mode) &&
S_ISDIR(statbuf[1].st_mode)) {
                    dirSize = sizeOfDir(&dirSize, newPath,
dirDepth);

                    if (dirSize == *psize) {
                        pinfo = (Info *)malloc(sizeof(Info));
                        init_fileInfo(pinfo, newPath, dirSize);
                        ListInsert(plist, pinfo);
                    }
                }
            }
            else
                searchFiles(plist, pinfo, psize, newPath, depth+1); //
재귀를 통한 깊이우선탐색
        }
        idx++;

```

```

    }
    closedir(dir);    // 파일 탐색 종료 후 디렉토리 닫음
    chdir("..");
}

// find 명령어 세부작업 2: 검색한 파일 속성 정보 출력
// 리스트에 저장할 파일 속성 정보
int init_fileInfo(Info * pinfo, char *file, long int size)
{
    struct stat buf;
    int depth = 0;

    // 현재 파일 정보 불러오기
    if (lstat(file, &buf) < 0) {
        fprintf(stderr, "%s's fileinfo errorWn", file);
        return FALSE;
    }

    // 파일 크기 정보 불러오기: 디렉토리면 하위 정규 파일들의 합 저장
    if (S_ISDIR(buf.st_mode))
        pinfo->size = size;
    else
        pinfo->size = buf.st_size;

    pinfo->modes = buf.st_mode;           // 파일 종류 및 접근 권한
    pinfo->blockCnt = buf.st_blocks;     // 할당된 블록 수
    pinfo->linkCnt = buf.st_nlink;       // 하드링크 개수
    pinfo->uid = buf.st_uid;             // 사용자 ID
    pinfo->gid = buf.st_gid;             // 그룹 ID
    pinfo->access = buf.st_atime;        // 최종 접근 시간
    pinfo->change = buf.st_ctime;        // 최종 상태 변경 시간
    pinfo->modify = buf.st_mtime;        // 최종 수정 시간
    realpath(file, pinfo->path);        // 파일의 절대 경로
    return TRUE;
}

// 파일 속성 중 파일 접근권한 정보 출력
void show_permission(mode_t * pmode)
{
    char permission[11];

    // 파일 유형 정보: 디렉토리, 정규 파일, 그 외
    if (S_ISDIR(*pmode))
        permission[0] = 'd';
    else if (S_ISREG(*pmode))
        permission[0] = '-';
    else

```



```

        permission[0] = '?';

// 사용자 읽기, 쓰기, 실행 권한
if (*pmode & S_IRUSR)
    permission[1] = 'r';
else
    permission[1] = '-';
if (*pmode & S_IWUSR)
    permission[2] = 'w';
else
    permission[2] = '-';
if (*pmode & S_IXUSR)
    permission[3] = 'x';
else
    permission[3] = '-';

// 그룹 읽기, 쓰기, 실행 권한
if (*pmode & S_IRUSR)
    permission[4] = 'r';
else
    permission[4] = '-';
if (*pmode & S_IWUSR)
    permission[5] = 'w';
else
    permission[5] = '-';
if (*pmode & S_IXUSR)
    permission[6] = 'x';
else
    permission[6] = '-';

// 기타 읽기, 쓰기, 실행 권한
if (*pmode & S_IRUSR)
    permission[7] = 'r';
else
    permission[7] = '-';
if (*pmode & S_IWUSR)
    permission[8] = 'w';
else
    permission[8] = '-';
if (*pmode & S_IXUSR)
    permission[9] = 'x';
else
    permission[9] = '-';

permission[10] = 'W0';

// 파일 접근 권한 출력
printf("%s ", permission);

```

```

}
// 리스트에 저장할 파일 접근 권한 출력
void show_fileInfo(List * plist, Info * pinfo)
{
    printf("%-4ld ", pinfo->size);
    show_permission(&pinfo->modes);
    printf("%-6ld %-5ld %-4d %-4d ", pinfo->blockCnt, pinfo->linkCnt, pinfo->uid,
pinfo->gid);
    printTime(&pinfo->access);
    printTime(&pinfo->change);
    printTime(&pinfo->modify);
    printf("%-sWn", pinfo->path);
}
// 파일 최종 접근 시간, 변경 시간, 수정 시간 정보 출력용
void printTime(time_t * ptime)
{
    struct tm * t;
    t = localtime(ptime);
    printf("%04d-%02d-%02d %02d:%02d ",
            (t->tm_year)+ 1900,      (t->tm_mon)+ 1,      t->tm_mday,      t->tm_hour,
t->tm_min);
}

// 리스트 초기화
void ListInit(List * plist)
{
    plist->head = (Node *)malloc(sizeof(Node));
    plist->head->next = NULL;
    plist->tail = plist->head;
    plist->numOfData = 0;
}
// 리스트 노드 추가
void ListInsert(List * plist, LData data)
{
    Node * newNode = (Node *)malloc(sizeof(Node));
    newNode->data = data;

    newNode->next = plist->head->next;
    plist->head->next = newNode;

    (plist->numOfData)++ ;
}
// 탐색 시작할 리스트 노드의 데이터 불러오기
int ListFirst(List * plist, LData * pdata)
{
    if (plist->head->next == NULL)
        return FALSE;

```

```

    plist->before = plist->head;
    plist->cur = plist->head->next;

    *pdata = plist->cur->data;
    return TRUE;
}
// 다음에 탐색할 노드의 데이터 불러오기
int ListNext(List * plist, LData * pdata)
{
    if (plist->cur->next == NULL)
        return FALSE;

    plist->before = plist->cur;
    plist->cur = plist->cur->next;

    *pdata = plist->cur->data;
    return TRUE;
}
// 리스트 노드 삭제
LData ListRemove(List * plist)
{
    Node * rmPos = plist->cur;
    LData rmData = rmPos->data;

    plist->before->next = plist->cur->next;
    plist->cur = plist->before;

    free(rmPos);
    (plist->numOfData)--;
    return rmData;
}
// 리스트 노드 개수 리턴
int ListCount(List * plist)
{
    return plist->numOfData;
}

void CompareDir(char *origDir, char *compDir)
{
    struct dirent **origEnt; // 기준 디렉토리의 파일 목록
    struct dirent **compEnt; // 비교할 디렉토리의 파일 목록
    DIR *dirp1, *dirp2;
    FILE *fp1, *fp2;
    int cnt1, cnt2; // 디렉토리 내 파일 개수
    int i, j;
    char *path1 = origDir + strlen(cwd) + 1; // 기준 디렉토리명 추출
    char *path2 = compDir + strlen(cwd) + 1; // 비교할 디렉토리명 추출
    char orig[BUFFER_SIZE]; // 기준 디렉토리 내 현재 파일의 절대경

```

로

```
char comp[BUFFER_SIZE]; // 비교할 디렉토리 내 현재 파일의 절대경로

// 기준 디렉토리와 비교할 디렉토리 오픈
dirp1 = opendir(origDir);
dirp2 = opendir(compDir);
if (dirp1 == NULL || dirp2 == NULL) {
    fprintf(stderr, "opendir errorWn");
    return;
}

// 두 디렉토리의 하위 파일목록을 알파벳순으로 정렬
cnt1 = scandir(origDir, &origEnt, NULL, alphasort);
cnt2 = scandir(compDir, &compEnt, NULL, alphasort);
if (cnt1 < 0 || cnt2 < 0) {
    fprintf(stderr, "scandir errorWn");
    return;
}

// 두 디렉토리 간 파일명 비교 수행
for (i = 0; i < cnt1; i++) {
    if (strcmp(origEnt[i]->d_name, ".") == 0 || strcmp(origEnt[i]->d_name, "..") == 0)
        continue;
    for (j = 0; j < cnt2; j++) {
        if (strcmp(compEnt[j]->d_name, ".") == 0 || strcmp(compEnt[j]->d_name, "..") == 0)
            continue;

        // 찾고자 하는 파일 이름이 동일
        if (strcmp(origEnt[i]->d_name, compEnt[j]->d_name) == 0) {
            if (origEnt[i]->d_type == DT_REG && compEnt[j]->d_type == DT_REG) // 둘 다 정규 파일
            {
                int same = TRUE;

                // 파일 데이터 불러오기
                snprintf(orig, sizeof(orig)-1, "%s/%s", origDir, origEnt[i]->d_name);
                snprintf(comp, sizeof(comp)-1, "%s/%s", compDir, compEnt[j]->d_name);

                fp1 = fopen(orig, "r");
                fp2 = fopen(comp, "r");
                if (fp1 == NULL || fp2 == NULL) {
                    fprintf(stderr, "subdirectory fopen errorWn");
                }
            }
        }
    }
}
```

```

        break;
    }

    // 파일의 일치 여부 확인
    while (!feof(fp1)) {
        char c1 = fgetc(fp1);
        while (!feof(fp2)) {
            int c2 = fgetc(fp2);
            if (c1 != c2) {
                same = FALSE;
            }
        }
    }
    fclose(fp1);
    fclose(fp2);

    // 두 파일이 다를 경우 차이 출력
    if (same == FALSE) {
        printf("diff %s/%s %s/%sWn", path1,
origEnt[i]->d_name, path2, compEnt[j]->d_name);
        CompareFileData(orig, comp);
    }
}
else if (origEnt[i]->d_type == DT_DIR &&
compEnt[j]->d_type == DT_DIR) // 둘 다 디렉토리 파일
{
    printf("Common Subdirectories: %s/%s and
%s/%sWn", path1, origEnt[i]->d_name, path2, compEnt[j]->d_name);
}
else { // 두 파일의 종류가 서로 다른 경우
    printf("File %s/%s is ", path1,
origEnt[i]->d_name);

    if (origEnt[i]->d_type == DT_REG) {
        printf("regular file ");
    }
    else if (origEnt[i]->d_type == DT_DIR)
    {
        printf("directory ");
    }
    else;
    printf("while file %s/%s is a ", path2,
compEnt[j]->d_name);

    if (compEnt[j]->d_type == DT_REG) {
        printf("regular fileWn");
    }
    else if (compEnt[j]->d_type == DT_DIR) {
        printf("directoryWn");
    }
}

```

```

        }
        break;
    }
    else if (j == cnt2-1) {    // 기준 디렉토리의 한 파일이 비교 디렉
토리에 없는 파일
        printf("Only in %s: %sWn", path1, origEnt[i]->d_name);
    }
}
if (i == cnt1-1) { // 기준 디렉토리에만 있고 비교 디렉토리에 없는 파일
    for (j = i; j < cnt2; j++)
        printf("Only in %s: %sWn", path2,
compEnt[j]->d_name);
    }
}

closedir(dirp1);
closedir(dirp2);
}

// 파일 내용 비교 작업 수행용 함수
void CompareFileData(char *original, char *comparision)
{
    FILE *fp1, *fp2;
    int len1 = 0, len2 = 0, min;
    int i, j;
    char same[BUFFER_SIZE][BUFFER_SIZE];
    int orig[BUFFER_SIZE];
    int comp[BUFFER_SIZE];
    int common[2][BUFFER_SIZE] = {0,};    // 0번: 원본, 1번: 인덱스
    int idx, cnt;
    int o1, n1;
    int o2, n2;

    fp1 = fopen(original, "r");
    fp2 = fopen(comparision, "r");

    if (fp1 == NULL || fp2 == NULL) {
        fprintf(stderr, "file not foundWn");
        return;
    }

    ArrayInit(origBuf);
    ArrayInit(compBuf);

    // 라인 단위로 비교할 데이터 읽기 수행, 라인 길이 구하기
    while (fgets(origBuf[len1], BUFFER_SIZE, fp1) != NULL) len1++;
    while (fgets(compBuf[len2], BUFFER_SIZE, fp2) != NULL) len2++;

```

```

// 비교본의 라인 중 원본과 같은 것을 찾는 작업 수행
idx = 0;
cnt = 0;
for (i = 0; i < len2; i++)
{
    for (j = 0; j < len1; j++)
    {
        // 라인의 데이터가 동일
        if (strcmp(origBuf[j], compBuf[i]) == 0)
        {
            // 한 라인에 대해 같은 문장이 여러 개
            for (int k = 0; k < BUFFER_SIZE; k++)
            {
                // 이미 같은 라인이 존재하면 라인 번호 저장 안
                if (strcmp(same[k], compBuf[i]) == 0) {
                    break;
                }
                else if (k == BUFFER_SIZE-1) // 이전에 저장
                {
                    if (i > 0 && i < len2)
                        strcpy(same[idx],
compBuf[i]);
                    else
                        strcpy(same[idx],
compBuf[i-1]);

                    comp[idx] = i;

                    if (orig[idx-1] > orig[idx])
                        orig[idx] = j;
                    else
                        orig[idx] = orig[idx-1];
                    idx++;
                }
            }
            break;
        }
        else {
            if (strcmp(origBuf[j-1], compBuf[i-1]) == 0) { //
직전의 라인이 서로 같은 경우
                for (int k = 0; k < BUFFER_SIZE; k++)
                {
                    // 추출한 공통의 문자열 중 현재 라인에
                    if (strcmp(same[k], compBuf[i]) ==
0) {

```

```

break;
}
else if (k == BUFFER_SIZE-1)    //
{
    strcpy(same[idx],
compBuf[i]);

    comp[idx] = i-1;
    if (orig[idx-1] > orig[idx]
&& idx > 0)
        orig[idx-1] =
orig[idx];
    else
        orig[idx] = j;

    common[0][cnt] =
    common[1][cnt] =
    cnt++;
}
else;
}
}
else if (strcmp(origBuf[j+1], compBuf[i+1]) == 0) {
    if (i == 0 && j > 0)
        printf("%d %dWn", j, i);
}
}
}

i = 0;
for (i = 0; i < cnt-1; i++) {
    o1 = common[0][i];
    n1 = common[1][i];

    // 원본에 라인이 새로 추가된 경우
    if (o1 == 0 && n1 > 0)
    {
        n2 = n1;
        while (strcmp(same[i], compBuf[n2]) != 0) n2++;
        printf("%d%cWn", o1-1, 'a');
        if (o1 == n2)
            printf("%dWn", n2);
        else

```



```

        printf("%d,%d\\n", o1, n2);
    for (int k = o1-1; k < n2; k++)
    {
        printf("< %s", compBuf[k]);
        if (k == len2-1) {
            printf("WW No newline at end of file\\n");
            break;
        }
    }
}
else if (o1 > 0 && n1 == 0)        // 원본에서 라인이 삭제된 경우
{
    o2 = o1;
    while (strcmp(same[i], compBuf[o2]) != 0) o2++;

    if (o2 <= o1)
        printf("%d", o1);
    else
        printf("%d,%d", o1, o2);
    printf("%c", 'd');
    printf("%d\\n", n1);

    for (int k = o1-1; k < o2; k++)
    {
        printf("> %s", origBuf[k]);
        if (k == len1-1) {
            printf("WW No newline at end of file\\n");
            break;
        }
    }
}
else {        // 일정 라인 범위에서 원본의 데이터가 변경된 경우
    o2 = common[0][i+1];
    n2 = common[1][i+1];

    if (o1 < o2 && (o2 > n1+1 || o2 < n1-1)) {
        printf("%d,%d%c%d,%d\\n", o1, o2, 'c', n1+1, n2+1);
    }

    if (o1 == len1-1)
        o2 = len1;
    for (int k = o1; k <= o2; k++)
    {
        printf("< %s", origBuf[k]);
        if (k == len1-1) {
            printf("WW No newline at end of file\\n");
            break;
        }
    }
}

```

```

        }
    }
    printf("---Wn");
    for (int k = n1; k <= n2; k++)
    {
        printf("> %s", compBuf[k]);
        if (k == len2-1) {
            printf("WW No newline at end of fileWn");
            break;
        }
    }
}

fclose(fp1);
fclose(fp2);
}

// 배열 초기화 함수
void ArrayInit(char (*arr)[BUFFER_SIZE])
{
    for (int i = 0; i < BUFFER_SIZE; i++) {
        for (int j = 0; j < BUFFER_SIZE; j++)
            arr[i][j] = 0;
    }
}

```