

## 1. 과제 개요

ssu\_sdup은 시스템 내 존재하는 동일(중복)한 파일을 찾고 삭제하는 프로그램이다. 이 프로그램은 내장명령어 fmd5, fsha1를 이용해 지정된 디렉토리와 그 하위 디렉토리의 각 정규 파일의 크기와 md5 혹은 sha1을 이용해 추출한 파일 데이터에 대한 해시값을 바탕으로 해시테이블을 구현한다. 이렇게 생성된 해시테이블은 정규 파일의 크기와 해시값을 기준으로 한 세트를 구성하여 각 세트별로 중복되는 정규파일들의 정보를 리스트 형태로 저장하여 출력한다. 전체 중복 리스트 출력 이후에는 작업을 수행할 세트 index에 대해 사용자가 설정한 옵션에 따라 해당 세트 내 중복 파일을 삭제하는 작업을 수행한다.

## 2. 구현 플랫폼

### 1) Linux 커널 버전(uname -a를 이용하여 확인)

Linux ubuntu 5.13.0-39-generic #44~20.04.1-Ubuntu SMP Thu Mar 24  
16:43:35 UTC 2022 x86\_64 x86\_64 x86\_64 GNU/Linux

### 2) 프로세서

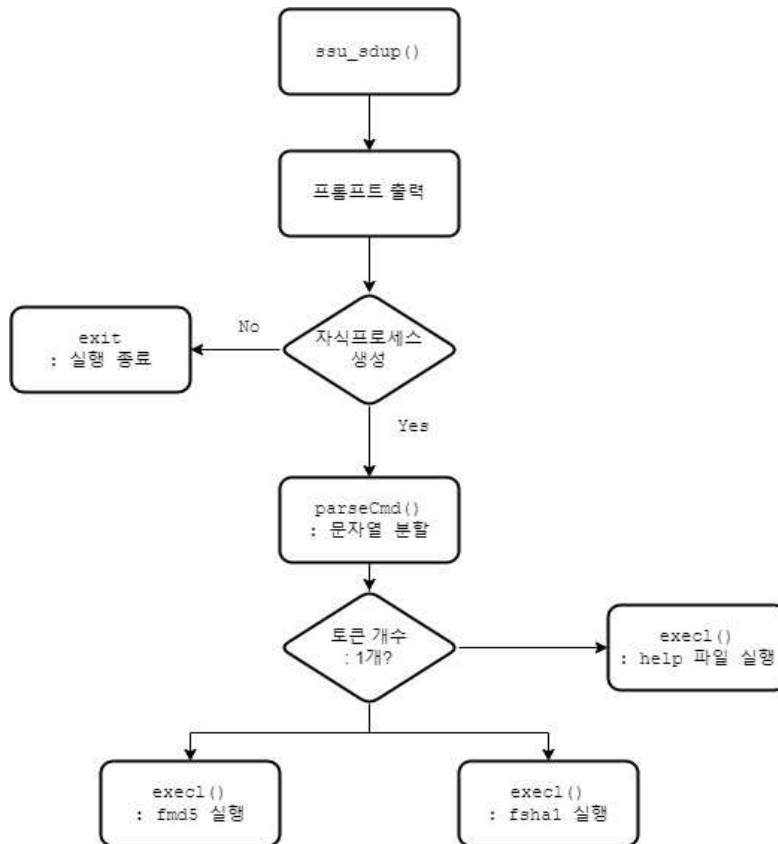
AMD® Ryzen 5 3400g with radeon vega graphics × 2

### 3) 사용 가상머신

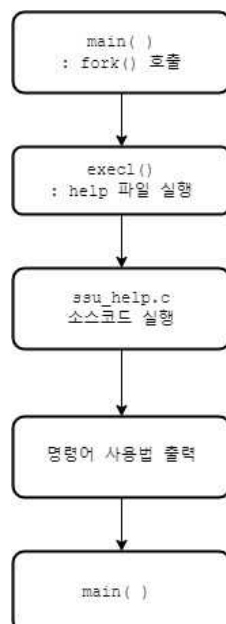
VMware Workstation 16 Player

### 3. 상세 설계

#### 1) ssu\_sdup.c



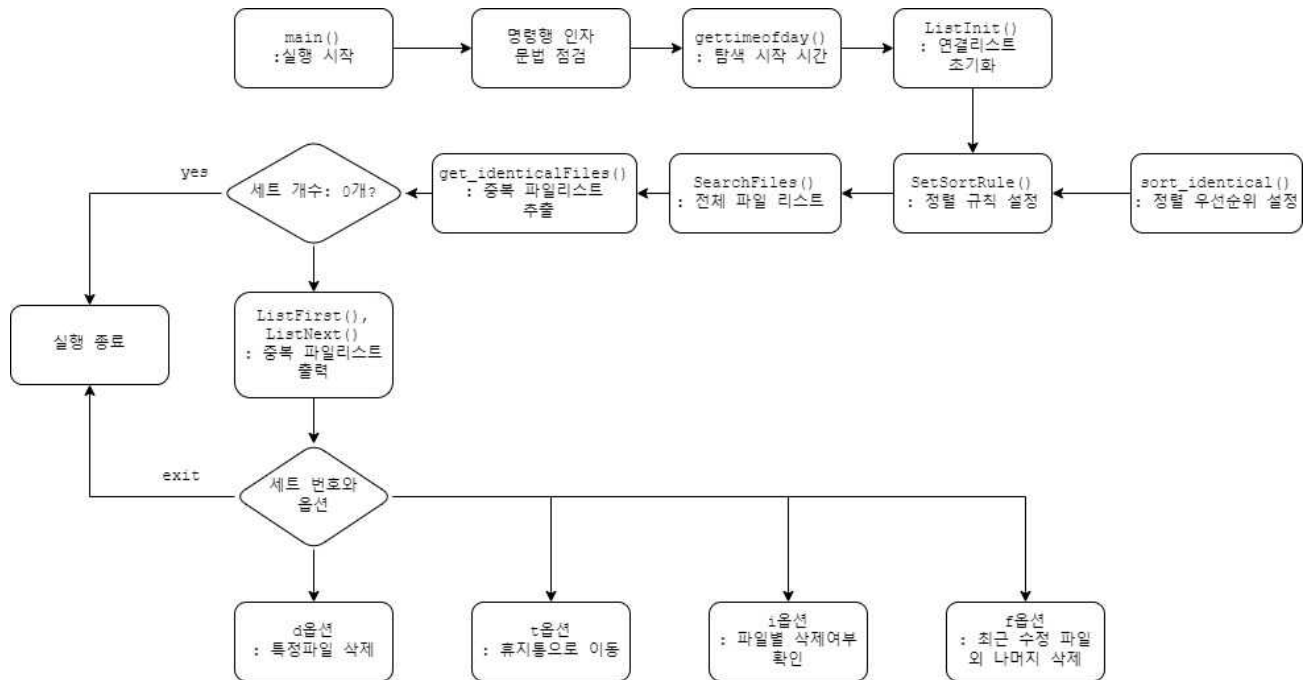
#### 2) ssu\_help.c



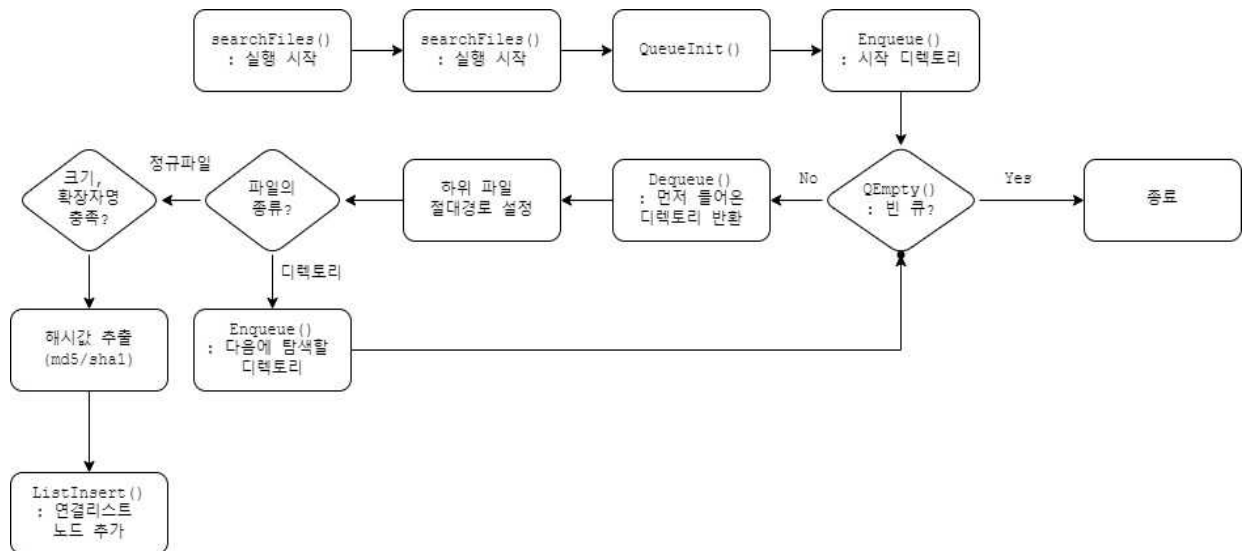
### 3) ssu\_find-md5.c/ssu\_find-sha1.c

#### a) main() 진행 순서도

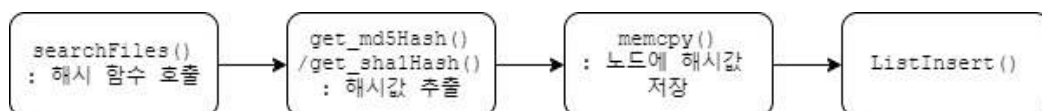
: 사용하는 해시 함수의 차이만 있을 뿐, 각각의 소스코드에서의 전체적인 진행 양상은 동일하다.



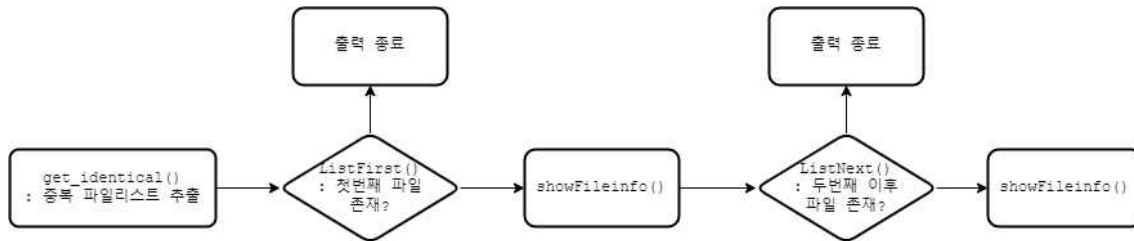
#### b) searchFiles()



#### c) 해시값 관련 함수



#### d) 중복 파일 리스트 출력 관련 함수



## 4. 구현 방법 설명

### 1. ssu\_functions.h/ssu\_functions.c에서 선언 및 정의된 함수

#### 1) 개요

ssu\_sdup.c 및 ssu\_find-md5.c와 ssu\_find-sha1.c에서 공통적으로 활용되는 함수를 ssu\_functions.h 및 ssu\_functions.c에서 선언 및 정의하여 사용한다.

#### 2) 디렉토리 탐색 관련

- **int parseCmd(char \*cmd, char \*argv[]);**  
: 공백을 기준으로 하나의 문자열을 분할하여 토큰을 생성하고, 생성된 토큰의 개수를 반환한다.
- **void ssu\_searchTime(struct timeval \*begin\_t, struct timeval \*end\_t);**  
: 탐색 시작 시간과 탐색 종료 시간을 이용해 탐색 시 소요 시간을 마이크로 초 단위까지 계산하여 총 탐색 시간을 출력한다.
- **char \*printTime(time\_t ptime);**  
: 연, 월, 일, 시, 분, 초 순으로 출력될 수 있도록 설정된 형식대로 함수 인자에 해당하는 시간을 문자열로 반환한다.
- **void putCommaToSize(long int size);**  
: 바이트 단위로 나타낸 천 단위마다 ','로 구분하여 출력한다.
- **void searchFiles(char \*dirname, int depth, List \* plist);**  
: 시작 디렉토리에서 하위 파일들에 대한 너비우선탐색을 수행하여 일반 파일이면 해당 파일의 정보를 연결 리스트 형태로 저장한다. 이때 저장되는 파일 정보는 파일의 절대경로와 그 길이, 해시값, 크기, 최종 접근 및 수정 시간이 된다. 실제 구현 시 함수의 정의부는 내장명령어 fmd5, fsha1에서 각각 사용되는 해시 함수의 차이로 인해 ssu\_find-md5.c와 ssu\_find-sha1.c에 따로 구분하여 명기하였다.
- **void get\_identicalFiles(List \*plist, List \*setList);**  
: searchFiles()을 통해 생성된, 전체 파일에 대한 연결리스트에서 해시값이 서로 같은 파일끼리 묶인 중복 파일 리스트를 여러 개 생성한다. 이때, 생성된 중복 파일 리스트는 setList에 저장된다.
- **int sort\_identical(LData d1, LData d2);**  
: searchFiles()를 이용하여 생성되는 리스트를 정렬하는 규칙에 해당하며, 정

렬의 우선 순위는 파일의 크기, 해시값, 파일의 절대경로의 길이 순으로 높다. 또한 반환값이 -1이면 오류차순, 1이면 내림차순 정렬로 간주한다.

- **void showFileList(List \*pset, int setnum);**

: 중복 파일 리스트 세트의 setnum번째 세트의 중복 파일 관련 정보를 출력한다. 출력되는 정보는 파일의 절대경로, 최종 접근 및 수정 시간이다.

### 3) 큐 구현

- **void QueueInit(Queue \*pq);**

: 큐의 head와 rear의 위치를 초기화한다.

- **int QEmpty(Queue \* pq);**

: 인자로 사용되는 큐가 현재 비어있으면 1, 비어있지 않으면 0을 반환한다.

- **void Enqueue(Queue \* pq, LData data);**

: 인자의 데이터를 큐에 삽입한다.

- **LData Dequeue(Queue \* pq);**

: 인자의 데이터를 큐에서 꺼낸다. 이때, 큐 내부에 저장된 데이터 중 가장 먼저 저장된 데이터부터 큐에서 꺼낸다.

### 4) 연결 리스트 구현

- **void ListInit(List \* plist);**

: 인자로 지정된 주소의 리스트를 초기화한다. 이때 리스트의 head에 대한 더미 노드를 생성하고 리스트 내 파일 개수를 0으로 초기화한다. 또한 리스트에서 서로 연결된 노드들을 정렬하는 규칙에 관한 함수 포인터를 초기화한다.

- **void ListInsert(List \* plist, LData data);**

: data에 저장된 파일 정보를 리스트에 저장하고 리스트에 저장된 파일 개수를 1 증가시킨다.

- **int ListFirst(List \* plist, LData \* pdata);**

: 리스트의 첫 번째 데이터가 pdata가 가리키는 메모리 주소에 저장되며, pdata가 리스트의 데이터 참조에 성공할 시 true, 실패 시 false를 반환한다.

- **int ListNext(List \* plist, LData \*pdata);**

: plist가 가리키는 리스트에서 현재 참조 중인 데이터의 다음 데이터를 pdata가 가리키는 메모리에 저장한다. 즉, ListFirst()가 먼저 호출된 이후부터 ListNext()를 통해 리스트 내 두번째 이후의 데이터를 참조하는 작업을 수행한다. 반환값은 참조 성공 시 true, 실패 시 false를 반환한다.

- **LData ListRemove(List \* plist);**

: ListFirst() 또는 ListNext()를 통해 참조된 마지막 데이터를 삭제하고 리스트에 저장된 노드의 개수를 1 감소시키는 함수이다. 반환값은 삭제된 노드의 데이터이다.

- **int ListCount(List \* plist);**

: 현재까지 리스트에 저장된 파일 데이터의 개수를 반환한다.

- **void SetSortRule(List \* plist, int (\*comp)(LData d1, LData d2));**  
: 리스트에 노드를 추가할 시 노드 정렬을 수행하기 위한 규칙에 대한 함수 포인터를 인자로 설정하고, 해당 포인터가 가리키는 함수를 리스트의 정렬 규칙으로 설정한다.

## 2. ssu\_function.c에만 정의된 기타 함수

- **void FreeInsert(List \* plist, LData data);**  
: plist가 가리키는 리스트에서 SetSortRule()을 통해 정렬 규칙을 정의하는 함수가 정의되지 않은 경우 ListInsert()에서 호출되는 함수로, 리스트에 대한 정렬 없이 노드만 추가한다.
- **void SortInsert(List \* plist, LData data);**  
: plist가 가리키는 리스트에서 SetSortRule()을 통해 설정한 정렬 규칙대로 정렬을 수행하면서 리스트에 노드를 추가한다.

## 3. 해시 함수

### 1) ssu\_find-md5.c : md5 해시 함수

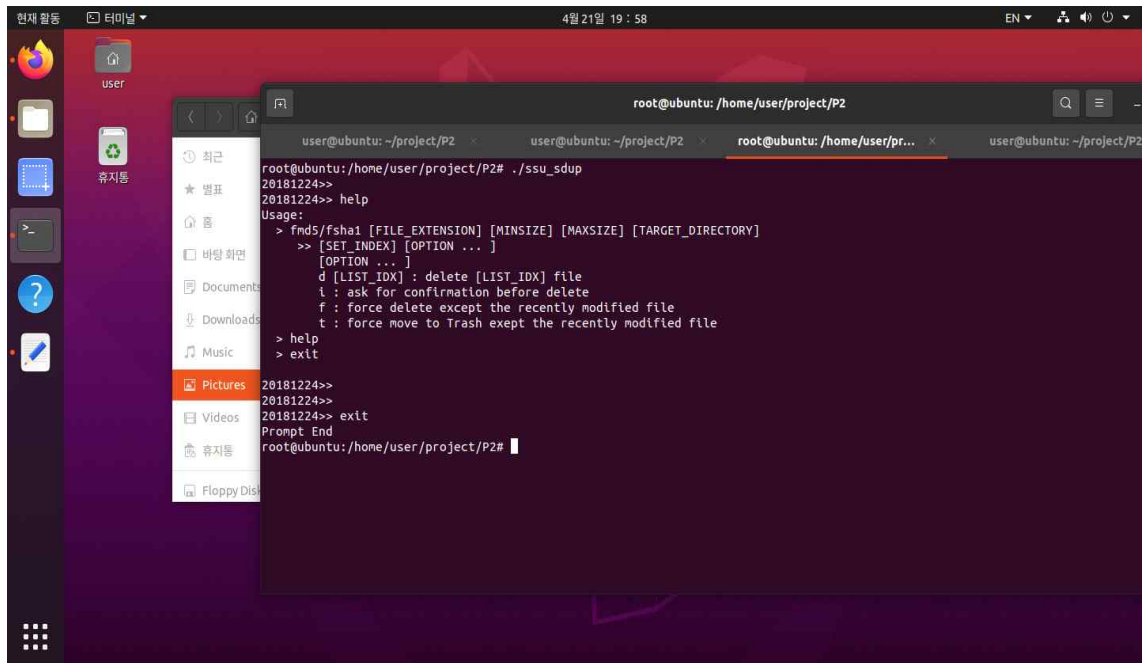
- **void get\_md5Hash(char \*filename, unsigned char \*md5);**  
: 인자로 지정된 파일의 데이터를 읽어와 128비트의 md5 해시값으로 변환하는 작업을 수행한다. 변환된 해시값은 두 번째 인자로 지정된 unsigned char 형 문자열 md5에 저장된다.
- **void put\_md5Hash(unsigned char \*md5);**  
: get\_md5Hash()를 통해 생성한 md5 해시값을 출력한다.

### 2) ssu\_find-sha1.c : sha1 해시 함수

- **void get\_sha1Hash(char \*filename, unsigned char \*sha1);**  
: 인자로 지정된 파일의 데이터를 읽어와 160비트의 sha1 해시값으로 변환하는 작업을 수행한다. 변환된 해시값은 두 번째 인자로 지정된 unsigned char 형 문자열 sha1에 저장된다.
- **void put\_sha1Hash(unsigned char \*sha1);**  
: get\_sha1Hash()를 통해 생성한 sha1 해시값을 출력한다.

## 5. 실행 결과

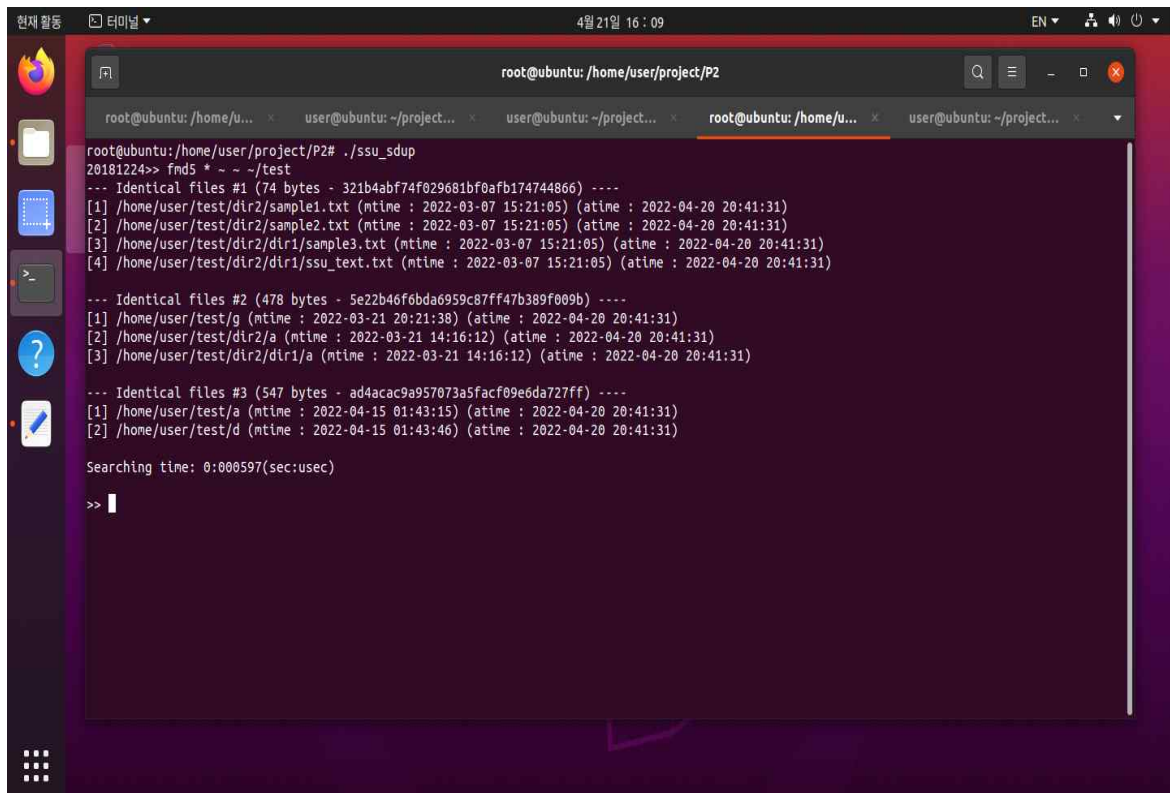
### 1) 프롬프트 출력



```
root@ubuntu: /home/user/project/P2# ./ssu_sdup
20181224>>
20181224>> help
Usage:
> fmd5/fsha1 [FILE_EXTENSION] [MINSIZE] [MAXSIZE] [TARGET_DIRECTORY]
>> [SET_INDEX] [OPTION ... ]
[OPTION ... ]
d [LIST_IDX] : delete [LIST_IDX] file
i : ask for confirmation before delete
f : force delete except the recently modified file
t : force move to Trash except the recently modified file
> help
> exit
20181224>>
20181224>> exit
Prompt End
root@ubuntu: /home/user/project/P2#
```

### 2) 내장명령어 fmd5

#### a) 기본 기능



```
root@ubuntu: /home/user/project/P2# ./ssu_sdup
20181224>> fmd5 * ~ ~/test
--- Identical files #1 (74 bytes - 321b4abf74f029681bf0afb174744866) ----
[1] /home/user/test/dir2/sample1.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-20 20:41:31)
[2] /home/user/test/dir2/sample2.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-20 20:41:31)
[3] /home/user/test/dir2/dir1/sample3.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-20 20:41:31)
[4] /home/user/test/dir2/dir1/ssu_text.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-20 20:41:31)

--- Identical files #2 (478 bytes - 5e22b46f6bda6959c87ff47b389f009b) ----
[1] /home/user/test/g (mtime : 2022-03-21 20:21:38) (atime : 2022-04-20 20:41:31)
[2] /home/user/test/dir2/a (mtime : 2022-03-21 14:16:12) (atime : 2022-04-20 20:41:31)
[3] /home/user/test/dir2/dir1/a (mtime : 2022-03-21 14:16:12) (atime : 2022-04-20 20:41:31)

--- Identical files #3 (547 bytes - ad4acac9a957073a5facf09e6da727ff) ----
[1] /home/user/test/a (mtime : 2022-04-15 01:43:15) (atime : 2022-04-20 20:41:31)
[2] /home/user/test/d (mtime : 2022-04-15 01:43:46) (atime : 2022-04-20 20:41:31)

Searching time: 0:000597(sec:usec)
>>
```

## b) d 옵션 수행

```
root@ubuntu: /home/user/project/P2
user@ubuntu: ~/project/P2
root@ubuntu: /home/user/project/P2
user@ubuntu: ~/project/P2

root@ubuntu: /home/user/project/P2# ./ssu_sdup
20181224>> find * ~ ~/test
--- Identical files #1 (74 bytes - 321b4abf74f029681bf0afb174744866) ----
[1] /home/user/test/dir1/sample3.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 17:07:46)
[2] /home/user/test/dir1/ssu_text.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 17:07:46)
[3] /home/user/test/dir2/sample2.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 17:07:46)
[4] /home/user/test/dir2/dir1/ssu_text.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-20 20:41:31)

--- Identical files #2 (478 bytes - 5e22b46f6bda6959c87ff47b389f009b) ----
[1] /home/user/test/g (mtime : 2022-03-21 20:21:38) (atime : 2022-04-21 17:07:46)
[2] /home/user/test/dir2/a (mtime : 2022-03-21 14:16:12) (atime : 2022-04-21 17:07:46)
[3] /home/user/test/dir2/dir1/a (mtime : 2022-03-21 14:16:12) (atime : 2022-04-21 17:07:46)

--- Identical files #3 (547 bytes - ad4acac9a957073a5facf09e6da727ff) ----
[1] /home/user/test/a (mtime : 2022-04-15 01:43:15) (atime : 2022-04-21 17:07:46)
[2] /home/user/test/d (mtime : 2022-04-15 01:43:46) (atime : 2022-04-21 17:07:46)

Searching time: 0:000998(sec:usec)

>> 1 d 3
"/home/user/test/dir2/sample2.txt" has been deleted in #1

--- Identical files #2 (74 bytes - 321b4abf74f029681bf0afb174744866) ----
[1] /home/user/test/dir1/sample3.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 17:07:46)
[2] /home/user/test/dir1/ssu_text.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 17:07:46)
[3] /home/user/test/dir2/dir1/ssu_text.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-20 20:41:31)

>> █
```

## c) t 옵션 수행

```
root@ubuntu: /home/user/project/P2
user@ubuntu: ~/project/P2
root@ubuntu: /home/user/project/P2
user@ubuntu: ~/project/P2

[1] /home/user/test/dir1/sample3.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 17:07:46)
[2] /home/user/test/dir1/ssu_text.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 17:07:46)
[3] /home/user/test/dir2/dir1/ssu_text.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-20 20:41:31)

--- Identical files #2 (478 bytes - 5e22b46f6bda6959c87ff47b389f009b) ----
[1] /home/user/test/g (mtime : 2022-03-21 20:21:38) (atime : 2022-04-21 17:07:46)
[2] /home/user/test/dir2/a (mtime : 2022-03-21 14:16:12) (atime : 2022-04-21 17:07:46)
[3] /home/user/test/dir2/dir1/a (mtime : 2022-03-21 14:16:12) (atime : 2022-04-21 17:07:46)

--- Identical files #3 (547 bytes - ad4acac9a957073a5facf09e6da727ff) ----
[1] /home/user/test/a (mtime : 2022-04-15 01:43:15) (atime : 2022-04-21 17:07:46)
[2] /home/user/test/d (mtime : 2022-04-15 01:43:46) (atime : 2022-04-21 17:07:46)

Searching time: 0:001039(sec:usec)

>> 1 t
All files in #1 have moved to Trash except "/home/user/test/dir1/sample3.txt" (2022-03-07 15:21:05)

--- Identical files #1 (478 bytes - 5e22b46f6bda6959c87ff47b389f009b) ----
[1] /home/user/test/g (mtime : 2022-03-21 20:21:38) (atime : 2022-04-21 17:07:46)
[2] /home/user/test/dir2/a (mtime : 2022-03-21 14:16:12) (atime : 2022-04-21 17:07:46)
[3] /home/user/test/dir2/dir1/a (mtime : 2022-03-21 14:16:12) (atime : 2022-04-21 17:07:46)

--- Identical files #2 (547 bytes - ad4acac9a957073a5facf09e6da727ff) ----
[1] /home/user/test/a (mtime : 2022-04-15 01:43:15) (atime : 2022-04-21 17:07:46)
[2] /home/user/test/d (mtime : 2022-04-15 01:43:46) (atime : 2022-04-21 17:07:46)

>> █
```



#### d) f 옵션 수행

```
root@ubuntu: /home/user/project/P2
user@ubuntu: ~/project/P2
user@ubuntu: ~/project/P2
root@ubuntu: /home/user/pr...
user@ubuntu: ~/project/P2

20181224>> fmd5 * ~ ~/test
--- Identical files #1 (74 bytes - 321b4abf74f029681bf0afb174744866) ----
[1] /home/user/test/dir1/sample3.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 20:25:17)
[2] /home/user/test/dir1/ssu_text.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 20:25:17)
[3] /home/user/test/dir2/sample1.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 20:25:17)
[4] /home/user/test/dir2/sample2.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 20:25:17)
[5] /home/user/test/dir2/dir1/ssu_text.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-20 20:41:31)

--- Identical files #2 (478 bytes - 5e22b46f6bda6959c87ff47b389f009b) ----
[1] /home/user/test/g (mtime : 2022-03-21 20:21:38) (atime : 2022-04-21 20:25:17)
[2] /home/user/test/dir2/a (mtime : 2022-03-21 14:16:12) (atime : 2022-04-21 20:25:17)
[3] /home/user/test/dir2/dir1/a (mtime : 2022-03-21 14:16:12) (atime : 2022-04-21 20:25:17)

--- Identical files #3 (547 bytes - ad4acac9a957073a5facf09e6da727ff) ----
[1] /home/user/test/a (mtime : 2022-04-15 01:43:15) (atime : 2022-04-21 20:25:17)
[2] /home/user/test/d (mtime : 2022-04-15 01:43:46) (atime : 2022-04-21 20:25:17)

Searching time: 0:000987(sec:usec)

>> 1 f
Left file in #1 : /home/user/test/dir1/sample3.txt (2022-03-07 15:21:05)

--- Identical files #1 (478 bytes - 5e22b46f6bda6959c87ff47b389f009b) ----
[1] /home/user/test/g (mtime : 2022-03-21 20:21:38) (atime : 2022-04-21 20:25:17)
[2] /home/user/test/dir2/a (mtime : 2022-03-21 14:16:12) (atime : 2022-04-21 20:25:17)
[3] /home/user/test/dir2/dir1/a (mtime : 2022-03-21 14:16:12) (atime : 2022-04-21 20:25:17)

--- Identical files #2 (547 bytes - ad4acac9a957073a5facf09e6da727ff) ----
[1] /home/user/test/a (mtime : 2022-04-15 01:43:15) (atime : 2022-04-21 20:25:17)
[2] /home/user/test/d (mtime : 2022-04-15 01:43:46) (atime : 2022-04-21 20:25:17)

>> █
```

#### e) i 옵션 수행(작동 가능하지만 제대로 구현은 안 됨)

```
user@ubuntu: ~/project/P2
user@ubuntu: ~/project/P2$ ./ssu_sdup
20181224>> fmd5 * ~ ~/test
--- Identical files #1 (74 bytes - 321b4abf74f029681bf0afb174744866) ----
[1] /home/user/test/dir1/sample3.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 21:19:11)
[2] /home/user/test/dir1/ssu_text.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 21:19:11)
[3] /home/user/test/dir2/sample1.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 21:19:11)
[4] /home/user/test/dir2/sample2.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 21:19:11)

--- Identical files #2 (547 bytes - ad4acac9a957073a5facf09e6da727ff) ----
[1] /home/user/test/a (mtime : 2022-04-15 01:43:15) (atime : 2022-04-21 21:19:11)
[2] /home/user/test/d (mtime : 2022-04-15 01:43:46) (atime : 2022-04-21 21:19:11)

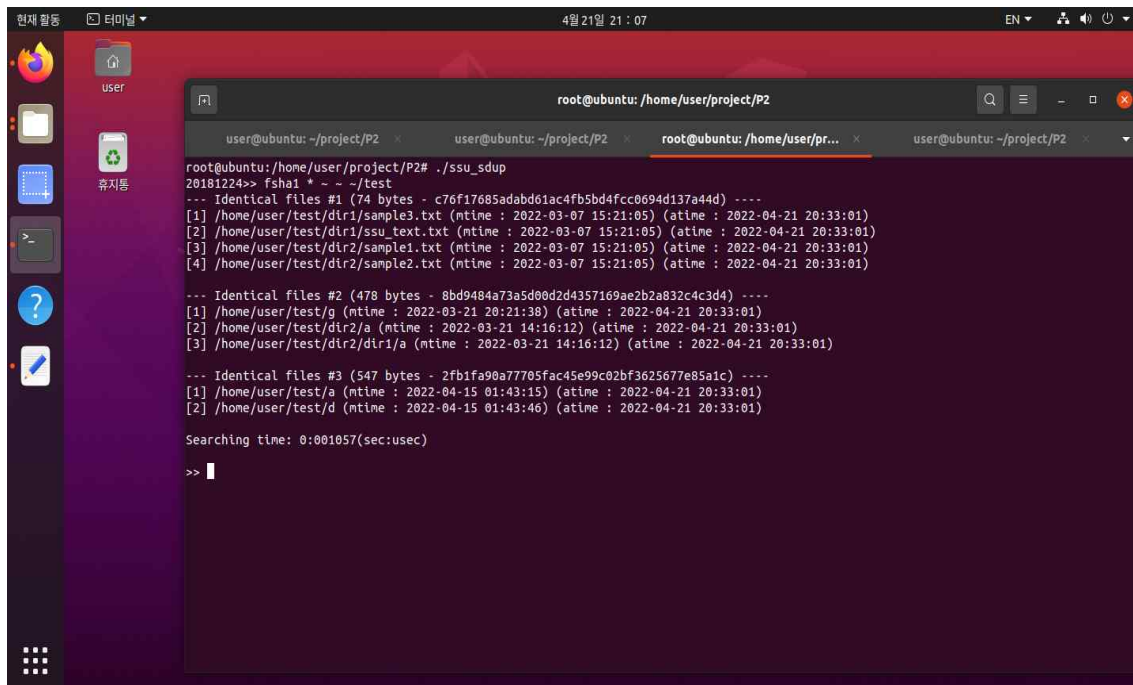
Searching time: 0:001249(sec:usec)

>> 1 i
Delete "/home/user/test/dir1/sample3.txt"? [y/n] y
--- Identical files #1 (74 bytes - 321b4abf74f029681bf0afb174744866) ----
[1] /home/user/test/dir1/ssu_text.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 21:19:11)
[2] /home/user/test/dir2/sample1.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 21:19:11)
[3] /home/user/test/dir2/sample2.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 21:19:11)

usage: [SET_INDEX] [OPTION ... ]
>> >> █
```

### 3) 내장명령어 fsha1

#### a) 기본 수행



A terminal window on an Ubuntu system. The user is root at /home/user/project/P2. They run the command `./ssu_sdup` followed by `fsha1 * ~ ~ -/test`. The output shows three groups of identical files with their SHA1 hashes, sizes, and timestamps. The first group has 4 files (74 bytes), the second has 3 files (478 bytes), and the third has 2 files (547 bytes). The search time is 0:001057(sec:usec).

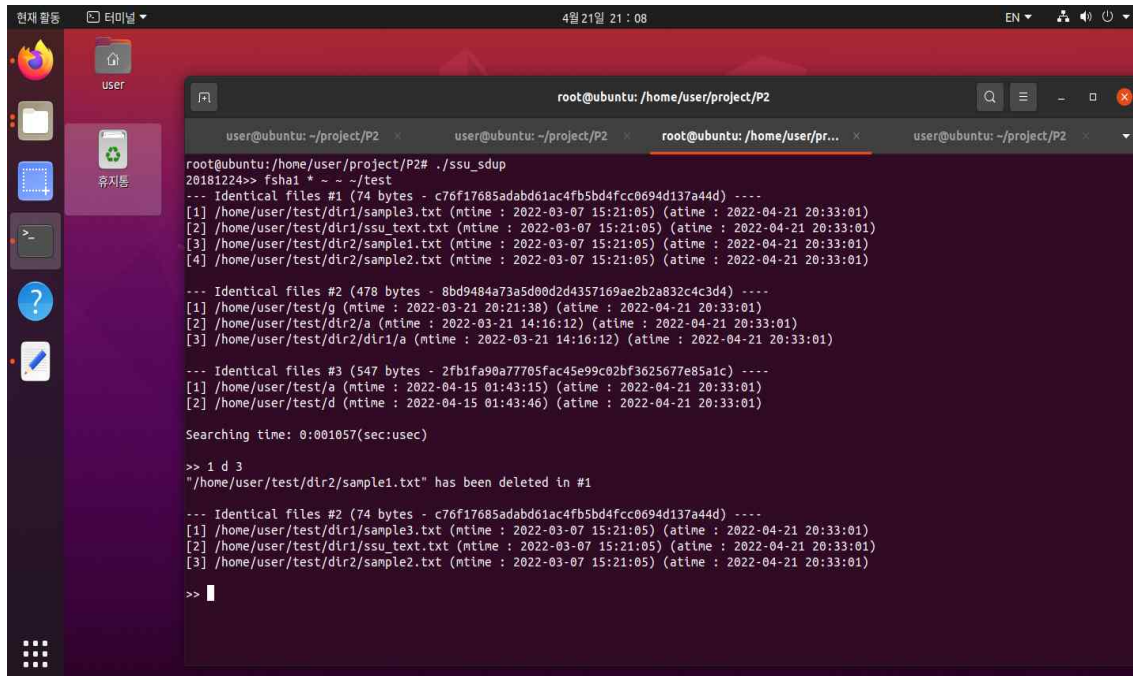
```
root@ubuntu:/home/user/project/P2# ./ssu_sdup
20181224>> fsha1 * ~ ~ -/test
--- Identical files #1 (74 bytes - c76f17685adabd61ac4fb5bd4fcc0694d137a44d) ----
[1] /home/user/test/dir1/sample3.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 20:33:01)
[2] /home/user/test/dir1/ssu_text.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 20:33:01)
[3] /home/user/test/dir2/sample1.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 20:33:01)
[4] /home/user/test/dir2/sample2.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 20:33:01)

--- Identical files #2 (478 bytes - 8bd9484a73a5d0d2d4357169ae2b2a832c4c3d4) ----
[1] /home/user/test/g (mtime : 2022-03-21 20:21:38) (atime : 2022-04-21 20:33:01)
[2] /home/user/test/dir2/a (mtime : 2022-03-21 14:16:12) (atime : 2022-04-21 20:33:01)
[3] /home/user/test/dir2/dir1/a (mtime : 2022-03-21 14:16:12) (atime : 2022-04-21 20:33:01)

--- Identical files #3 (547 bytes - 2fb1fa90a77705fac45e99c02bf3625677e85a1c) ----
[1] /home/user/test/a (mtime : 2022-04-15 01:43:15) (atime : 2022-04-21 20:33:01)
[2] /home/user/test/d (mtime : 2022-04-15 01:43:46) (atime : 2022-04-21 20:33:01)

Searching time: 0:001057(sec:usec)
>> 
```

#### b) d 옵션



A terminal window on an Ubuntu system. The user is root at /home/user/project/P2. They run the command `./ssu_sdup` followed by `fsha1 * ~ ~ -/test`. The output is identical to the previous screenshot. After the search time, they enter `>> 1 d 3`, and a message appears: `"/home/user/test/dir2/sample1.txt" has been deleted in #1`. The output then shows the remaining files.

```
root@ubuntu:/home/user/project/P2# ./ssu_sdup
20181224>> fsha1 * ~ ~ -/test
--- Identical files #1 (74 bytes - c76f17685adabd61ac4fb5bd4fcc0694d137a44d) ----
[1] /home/user/test/dir1/sample3.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 20:33:01)
[2] /home/user/test/dir1/ssu_text.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 20:33:01)
[3] /home/user/test/dir2/sample1.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 20:33:01)
[4] /home/user/test/dir2/sample2.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 20:33:01)

--- Identical files #2 (478 bytes - 8bd9484a73a5d0d2d4357169ae2b2a832c4c3d4) ----
[1] /home/user/test/g (mtime : 2022-03-21 20:21:38) (atime : 2022-04-21 20:33:01)
[2] /home/user/test/dir2/a (mtime : 2022-03-21 14:16:12) (atime : 2022-04-21 20:33:01)
[3] /home/user/test/dir2/dir1/a (mtime : 2022-03-21 14:16:12) (atime : 2022-04-21 20:33:01)

--- Identical files #3 (547 bytes - 2fb1fa90a77705fac45e99c02bf3625677e85a1c) ----
[1] /home/user/test/a (mtime : 2022-04-15 01:43:15) (atime : 2022-04-21 20:33:01)
[2] /home/user/test/d (mtime : 2022-04-15 01:43:46) (atime : 2022-04-21 20:33:01)

Searching time: 0:001057(sec:usec)
>> 1 d 3
"/home/user/test/dir2/sample1.txt" has been deleted in #1
--- Identical files #2 (74 bytes - c76f17685adabd61ac4fb5bd4fcc0694d137a44d) ----
[1] /home/user/test/dir1/sample3.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 20:33:01)
[2] /home/user/test/dir1/ssu_text.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 20:33:01)
[3] /home/user/test/dir2/sample2.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 20:33:01)
>> 
```

### c) t 옵션

```

root@ubuntu: /home/user/project/P2
root@ubuntu: /home/user/project/P2# ./ssu_sdup
20181224>> fmd5 * ~ ~ ~/test
--- Identical files #1 (74 bytes - 321b4abf74f029681bf0afb174744866) ----
[1] /home/user/test/dir1/sample3.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 22:43:41)
[2] /home/user/test/dir1/ssu_text.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 22:43:41)
[3] /home/user/test/dir2/sample1.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 22:43:41)
[4] /home/user/test/dir2/sample2.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 22:43:41)

--- Identical files #2 (478 bytes - 5e22b46f6bda6959c87ff47b389f009b) ----
[1] /home/user/test/g (mtime : 2022-03-21 20:21:38) (atime : 2022-04-21 22:43:41)
[2] /home/user/test/dir2/a (mtime : 2022-03-21 14:16:12) (atime : 2022-04-21 22:43:41)
[3] /home/user/test/dir2/dir1/a (mtime : 2022-03-21 14:16:12) (atime : 2022-04-21 22:43:41)

--- Identical files #3 (547 bytes - ad4acac9a957073a5facf09e6da727ff) ----
[1] /home/user/test/a (mtime : 2022-04-15 01:43:15) (atime : 2022-04-21 22:43:41)
[2] /home/user/test/d (mtime : 2022-04-15 01:43:46) (atime : 2022-04-21 22:43:41)

Searching time: 0:000958(sec:usec)

>> 2 t
All files in #2 have moved to Trash except "/home/user/test/g" (2022-03-21 20:21:38)

--- Identical files #1 (74 bytes - 321b4abf74f029681bf0afb174744866) ----
[1] /home/user/test/dir1/sample3.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 22:43:41)
[2] /home/user/test/dir1/ssu_text.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 22:43:41)
[3] /home/user/test/dir2/sample1.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 22:43:41)
[4] /home/user/test/dir2/sample2.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 22:43:41)

--- Identical files #2 (547 bytes - ad4acac9a957073a5facf09e6da727ff) ----
[1] /home/user/test/a (mtime : 2022-04-15 01:43:15) (atime : 2022-04-21 22:43:41)
[2] /home/user/test/d (mtime : 2022-04-15 01:43:46) (atime : 2022-04-21 22:43:41)

>> █

```

### d) f 옵션

```

root@ubuntu: /home/user/project/P2
root@ubuntu: /home/user/project/P2# ./ssu_sdup
20181224>> fsha1 * ~ ~ ~/test
--- Identical files #1 (74 bytes - c76f17685adabd61ac4fb5bd4fcc0694d137a44d) ----
[1] /home/user/test/dir1/sample3.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 21:18:45)
[2] /home/user/test/dir1/ssu_text.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 21:18:45)
[3] /home/user/test/dir2/sample1.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 21:18:45)
[4] /home/user/test/dir2/sample2.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 21:18:45)

--- Identical files #2 (478 bytes - 8bd9484a73a5d00d2d4357169ae2b2a832c4c3d4) ----
[1] /home/user/test/g (mtime : 2022-03-21 20:21:38) (atime : 2022-04-21 21:18:45)
[2] /home/user/test/dir2/a (mtime : 2022-03-21 14:16:12) (atime : 2022-04-21 21:18:45)
[3] /home/user/test/dir2/dir1/a (mtime : 2022-03-21 14:16:12) (atime : 2022-04-21 21:18:45)

--- Identical files #3 (547 bytes - 2fb1fa90a77705fac45e99c02bf3625677e85a1c) ----
[1] /home/user/test/a (mtime : 2022-04-15 01:43:15) (atime : 2022-04-21 21:18:42)
[2] /home/user/test/d (mtime : 2022-04-15 01:43:46) (atime : 2022-04-21 21:18:45)

Searching time: 0:001013(sec:usec)

>> 2 f
Left file in #2 : /home/user/test/g (2022-03-21 20:21:38)

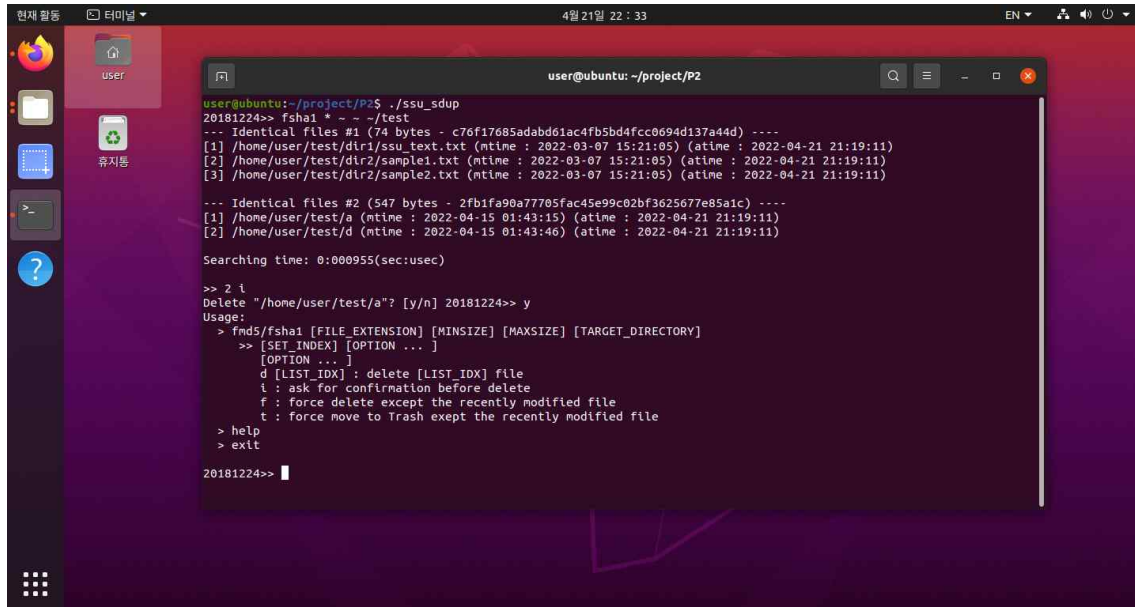
--- Identical files #1 (74 bytes - c76f17685adabd61ac4fb5bd4fcc0694d137a44d) ----
[1] /home/user/test/dir1/sample3.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 21:18:45)
[2] /home/user/test/dir1/ssu_text.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 21:18:45)
[3] /home/user/test/dir2/sample1.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 21:18:45)
[4] /home/user/test/dir2/sample2.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 21:18:45)

--- Identical files #2 (547 bytes - 2fb1fa90a77705fac45e99c02bf3625677e85a1c) ----
[1] /home/user/test/a (mtime : 2022-04-15 01:43:15) (atime : 2022-04-21 21:18:42)
[2] /home/user/test/d (mtime : 2022-04-15 01:43:46) (atime : 2022-04-21 21:18:45)

>> █

```

### e) i 옵션(구현 미함)



```
user@ubuntu: ~/project/P2
user@ubuntu:~/project/P2$ ./ssu_sdup
20181224>> fsha1 * ~ - /test
--- Identical files #1 (74 bytes - c76f17685adabdd61ac4fb5bd4fcc0694d137a44d) ----
[1] /home/user/test/dir1/ssu_text.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 21:19:11)
[2] /home/user/test/dir2/sample1.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 21:19:11)
[3] /home/user/test/dir2/sample2.txt (mtime : 2022-03-07 15:21:05) (atime : 2022-04-21 21:19:11)

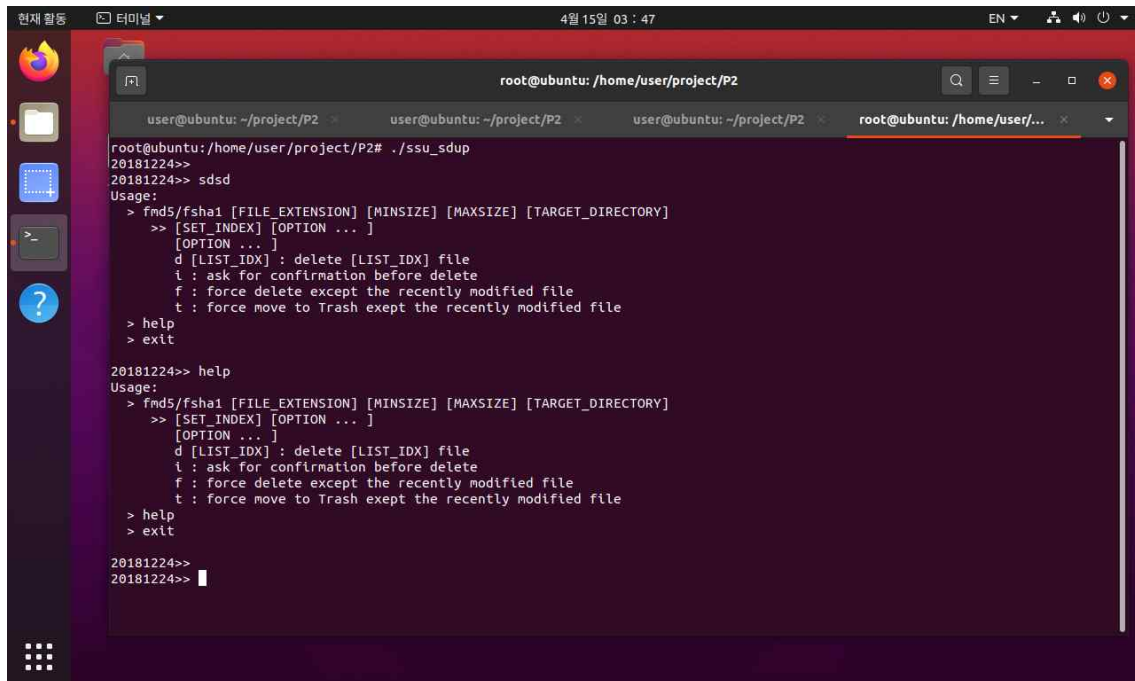
--- Identical files #2 (547 bytes - 2fb1fa90a77705fac45e99c02bf3625677e85a1c) ----
[1] /home/user/test/a (mtime : 2022-04-15 01:43:15) (atime : 2022-04-21 21:19:11)
[2] /home/user/test/d (mtime : 2022-04-15 01:43:46) (atime : 2022-04-21 21:19:11)

Searching time: 0:000955(sec:usec)

>> 2 i
Delete "/home/user/test/a"? [y/n] 20181224>> y
Usage:
> fmd5/fsha1 [FILE_EXTENSION] [MINSIZE] [MAXSIZE] [TARGET_DIRECTORY]
>> [SET_INDEX] [OPTION ... ]
[OPTION ... ]
d [LIST_IDX] : delete [LIST_IDX] file
i : ask for confirmation before delete
f : force delete except the recently modified file
t : force move to Trash except the recently modified file
> help
> exit
20181224>>
```



#### 4) 내장명령어 help



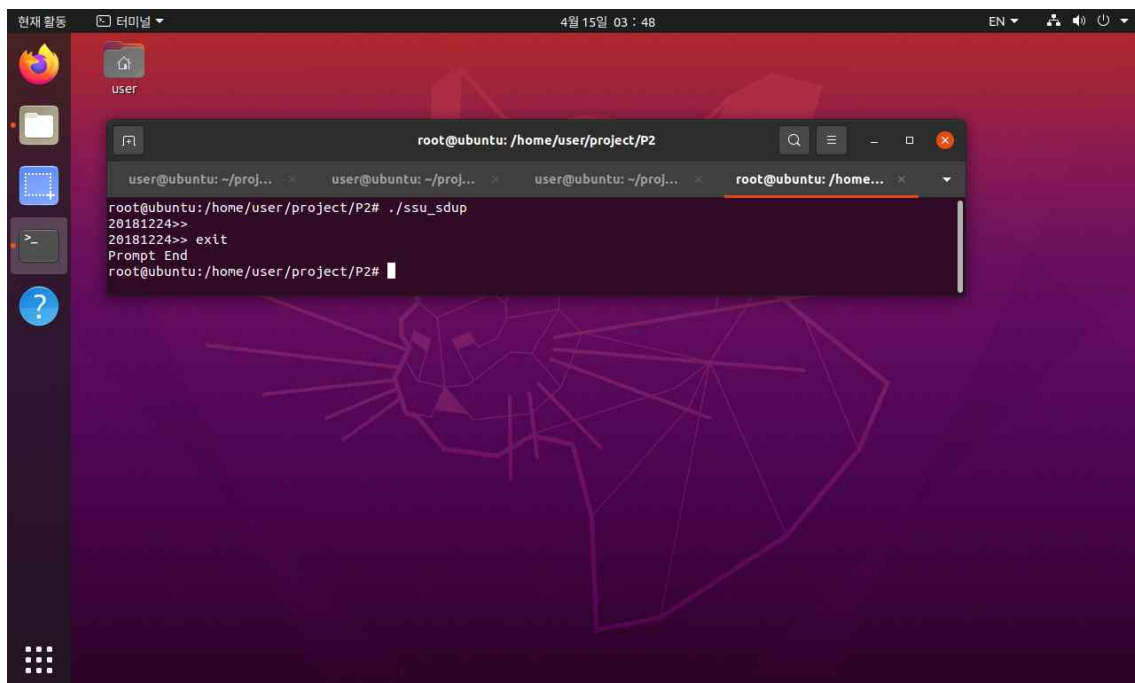
A terminal window on an Ubuntu desktop. The terminal shows the execution of the `ssu_sdup` program. The user enters `sdsc`, which displays a usage message. Then, the user enters `help`, which displays the same usage message. The terminal text is as follows:

```
root@ubuntu: /home/user/project/P2# ./ssu_sdup
20181224>>
20181224>> sdsc
Usage:
> fmd5/fsha1 [FILE_EXTENSION] [MINSIZE] [MAXSIZE] [TARGET_DIRECTORY]
>> [SET_INDEX] [OPTION ... ]
[OPTION ... ]
d [LIST_IDX] : delete [LIST_IDX] file
i : ask for confirmation before delete
f : force delete except the recently modified file
t : force move to Trash except the recently modified file
> help
> exit

20181224>> help
Usage:
> fmd5/fsha1 [FILE_EXTENSION] [MINSIZE] [MAXSIZE] [TARGET_DIRECTORY]
>> [SET_INDEX] [OPTION ... ]
[OPTION ... ]
d [LIST_IDX] : delete [LIST_IDX] file
i : ask for confirmation before delete
f : force delete except the recently modified file
t : force move to Trash except the recently modified file
> help
> exit

20181224>>
20181224>>
```

#### 5) 내장명령어 exit



A terminal window on an Ubuntu desktop. The terminal shows the execution of the `ssu_sdup` program. The user enters `exit`, which displays the prompt `Prompt End`. The terminal text is as follows:

```
root@ubuntu: /home/user/project/P2# ./ssu_sdup
20181224>>
20181224>> exit
Prompt End
root@ubuntu: /home/user/project/P2#
```

## 6. 소스코드

### 1) ssu\_sdup.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
#include <sys/types.h>
#include "ssu_functions.h"

#define PROMPT "20181224>> "

int main(void)
{
    char command[BUFMAX];
    pid_t pid;
    int status;

    while (1) {
        // 명령어 입력
        printf(PROMPT);
        memset(command, 0, BUFMAX);
        fgets(command, BUFMAX, stdin);

        // Enter 키 혹은 exit 입력
        if (strcmp(command, "\n") == 0)
            continue;
        else if (strcmp(command, "exit\n") == 0) {
            printf("Prompt End\n");
            break;
        }
        else // 기타 다른 명령어 입력
            command[strlen(command)-1] = '\0';

        // 내장명령어 fmd5, fsha1, help를 실행하기 위한 자식 프로세스 생성
        pid = fork();
        if (pid == 0) { // 자식 프로세스
            char *argv[ARGMAX];
            int argc;

            argc = parseCmd(command, argv);

            // fmd5/fsha1 명령어 실행
            if (strcmp(argv[0], "fmd5") == 0 || strcmp(argv[0], "fsha1") ==
0) {
                // 주어진 인자 중 하나라도 입력이 없으면 에러 처리
                if (argc != ARGMAX) {
                    fprintf(stderr, "usage: %s [FILE_EXTENSION]
```

```

[MINSIZE] [MAXSIZE] [TARGET_DIRECTORY]Wn", argv[0]);
        exit(1);
    }

    // fmd5/fsha1 명령어 실행
    if (execl(argv[0], argv[0], argv[1], argv[2], argv[3],
argv[4], NULL) == -1)
    {
        fprintf(stderr, "%s: execl errorWn", argv[0]);
        exit(1);
    }
}
else { // help 명령어 실행
    if (execl("help", argv[0], NULL) == -1) {
        fprintf(stderr, "%s: execl errorWn", argv[0]);
        exit(1);
    }
}
}

// 부모 프로세스는 자식 프로세스 종료까지 대기
while (wait(&status) != pid);
}

exit(0);
}

```

## 2) ssu\_help.c

```

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

int main(void)
{
    printf("Usage:Wn");
    printf(" > fmd5/fsha1 [FILE_EXTENSION] [MINSIZE] [MAXSIZE]
[TARGET_DIRECTORY]Wn");
    printf("    >> [SET_INDEX] [OPTION ... ]Wn");
    printf(" [OPTION ... ]Wn");
    printf(" d [LIST_IDX] : delete [LIST_IDX] fileWn");
    printf(" i : ask for confirmation before deleteWn");
    printf(" f : force delete except the recently modified fileWn");
    printf(" t : force move to Trash exepct the recently modified fileWn");
    printf(" > helpWn");
    printf(" > exitWnWn");

    exit(0);
}

```

### 3) ssu\_find-md5.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <dirent.h>
#include <errno.h>
#include <limits.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <openssl/md5.h>
#include "ssu_functions.h"

char *ext;           // 추출한 파일 확장자명
long int minsize;
long int maxsize;
char dirPath[BUFMAX]; // [TARGET_DIRECTORY]의 절대경로
void get_md5Hash(char *filename, unsigned char *md); // 파일로부터 md5 해시값
추출
void put_md5Hash(unsigned char *md); // 해시값 출력
List setList[LISTMAX];
int setIdx = 0;

int main(int argc, char *argv[])
{
    char *ptr;
    double min, max;
    char dirName[BUFMAX];
    struct stat statbuf;
    struct timeval begin_t, end_t;
    long int size;
    unsigned char *hash;
    List list;
    Info * pinfo;

    // [FILE_EXTENSION] 인자 점검
    if (argv[1][0] == '*') {
        if (strlen(argv[1]) == 1) { // "*"만 입력
            ext = argv[1] + 1;
        }
        else if (argv[1][1] == '.' && strlen(argv[1]) > 2) { // ".*(확장자명)"
            ext = argv[1] + 1;
        }
        else { // 입력된 것이 "*.나 "*asdsds" 같은 경우
            fprintf(stderr, "file extension error\n");
            exit(1);
        }
    }
}
```



```

else { // "*"나 "*(확장자명)" 이외의 입력
    fprintf(stderr, "file extension error!Wn");
    exit(1);
}
// [MINSIZE] 인자 점검
// 실수로 변환할 숫자가 없는 경우
if ((min = strtod(argv[2], &ptr)) == 0) {
    if (strcmp(argv[2], "~") == 0) { // "~" 입력
        minsize = 1;
    }
    else { // "~" 외 다른 문자열 입력
        fprintf(stderr, "minsize error!Wn");
        exit(1);
    }
}
else if (strcmp(ptr, "W0") != 0) { // 실수 변환 후 문자열 존재
    if (strcmp(ptr, "KB") == 0 || strcmp(ptr, "kb") == 0) // KB 단위 byte
        변환
        {
            min *= 1024;
        }
    else if (strcmp(ptr, "MB") == 0 || strcmp(ptr, "mb") == 0) // MB단위
        byte 변환
        {
            min *= 1024*1024;
        }
    else if (strcmp(ptr, "GB") == 0 || strcmp(ptr, "gb") == 0) // GB단위
        byte 변환
        {
            min *= 1024*1024*1024;
        }
    else {
        fprintf(stderr, "minsize error!Wn");
        exit(1);
    };
    minsize = (long)min;
}
else
    minsize = (long)min;
// [MAXSIZE] 인자 점검
// 숫자 없이 "~"만 입력된 경우
if ((max = strtod(argv[3], &ptr)) == 0) {
    if (strcmp(argv[3], "~") == 0) { // "~" 입력
        maxsize = LONG_MAX; // long형의 최대치로 설정
    }
    else { // "~" 외 다른 문자열 입력
        fprintf(stderr, "maxsize error!Wn");
        exit(1);
    }
}

```

```

    }
}
else if (strcmp(ptr, "W0") != 0) { // 실수 변환 후 문자열 존재
    if (strcmp(ptr, "KB") == 0 || strcmp(ptr, "kb") == 0) // KB 단위 byte
        변환
        {
            max *= 1024;
        }
    else if (strcmp(ptr, "MB") == 0 || strcmp(ptr, "mb") == 0) // MB단위
        byte 변환
        {
            max *= 1024*1024;
        }
    else if (strcmp(ptr, "GB") == 0 || strcmp(ptr, "gb") == 0) // GB단위
        byte 변환
        {
            max *= 1024*1024*1024;
        }
        else {
            fprintf(stderr, "maxsize errorWn");
            exit(1);
        }
        maxsize = (long)max;
    }
    else {
        maxsize = (long)max;
    }
}
// minsize가 maxsize보다 큰 경우 에러 처리
if (minsize > maxsize) {
    fprintf(stderr, "minsize must not be bigger than maxsizeWn");
    exit(1);
}

// [TARGET_DIRECTORY] 인자 점검
if ((ptr = strstr(argv[4], "~")) != NULL) { // "~(홈 디렉토리)"를 포함한 경로인
    경우
        sprintf(dirName, "%s/%s%s", "/home", getenv("USER"), ptr+1);
    }
    else {
        ptr = argv[4];
        strcpy(dirName, ptr);
    }
}
// 입력한 경로를 절대경로로 변환, 실제로 유효한 경로가 아니면 에러 처리
if (realpath(dirName, dirPath) == NULL) {
    fprintf(stderr, "%s is not existWn", dirPath);
    exit(1);
}
// 입력한 경로의 파일이 디렉토리인지 확인

```

```

if (lstat(dirPath, &statbuf) < 0) {
    fprintf(stderr, "lstat error for %s\n", dirPath);
    exit(1);
}
if (!S_ISDIR(statbuf.st_mode)) {
    fprintf(stderr, "Path is not directory\n");
    exit(1);
}

// 검색 시간 측정
gettimeofday(&begin_t, NULL);

// 지정된 디렉토리의 모든 하위 파일들에 대한 파일 리스트 초기화
// 중복 파일 리스트 초기화 및 정렬 규칙 설정
ListInit(&list);
SetSortRule(&list, sort_identical);
for (int i = 0; i < LISTMAX; i++) {
    ListInit(&setList[i]);
    SetSortRule(&setList[i], sort_identical);
}
// 명령행 인자의 조건을 충족하는 전체 파일 리스트 생성
searchFiles(dirPath, 0, &list);
// 전체 파일에서 중복 리스트 추출: 현재 노드 기준 직후 노드의 해시값이 다르면 재귀
setIdx = get_identicalFiles(&list, setIdx);
// 중복 파일 리스트가 존재하지 않는 경우
if (setIdx == 0) {
    printf("No duplicates in %s\n", dirPath);
    gettimeofday(&end_t, NULL);
    ssu_searchTime(&begin_t, &end_t); // 전체 검색 시간 출력
}
else {
    for (int i = 0; i < setIdx; i++) {
        showFileList(&setList[i], i);
    }
    gettimeofday(&end_t, NULL);
    ssu_searchTime(&begin_t, &end_t); // 전체 검색 시간 출력

// >> [SET_INDEX] [OPTION ... ] 작업 수행
while (1) {
    char cmd[BUFMAX]; // 명령문
    int set_num; // 세트 번호
    int list_num; // 한 세트 내 파일 번호
    int cnt; // 명령문 파싱용
    char *token[TOKMAX];
    Info *pinfo;

    // 삭제를 수행할 세트 번호 및 옵션 입력
    memset(cmd, 0, BUFMAX);

```

```

printf(">> ");
fgets(cmd, BUFMAX, stdin);

// 입력이 없으면 에러 처리
if (strcmp(cmd, "Wn") == 0) {
    fprintf(stderr, "usage: [SET_INDEX] [OPTION ...
]Wn");

    continue;
}
else if (strcmp(cmd, "exitWn") == 0) {
    printf(">> Back to PromptWn");
    break;
}
else
    cmd[strlen(cmd)-1] = 'W0';
// 인자의 개수는 최소 2개 이상
if ((cnt = parseCmd(cmd, token)) < 2) {
    fprintf(stderr, "usage: [SET_INDEX] [OPTION ...
]Wn");

    continue;
}
// 첫번째 인자가 인덱스 범위 내 숫자인지 확인
if ((set_num = atoi(token[0])) <= 0 || set_num > setIdx) {
    fprintf(stderr, "set index errorWn");
    continue;
}
if (strcmp(token[1], "d") == 0) { // d옵션 설정
    int index = 1; // 인덱스 번호
    Info * rminfo; // 삭제된 데이터

    // [LIST_IDX] 에 해당하는 파일 삭제
    if (cnt > 2 && (list_num = atoi(token[2])) > 0 &&
        list_num <
ListCount(&setList[set_num-1])) {
        // 지정한 파일 제거
        if (ListFirst(&setList[set_num-1], &pinfo)) {
            for (int i = 1; i <=
ListCount(&setList[set_num-1]); i++) {
                // 현재 인덱스 번호가
                [LIST_IDX]와 동일
                if (i == list_num) {
                    // 중복 파일
                    리스트에서 삭제
                    rminfo =
ListRemove(&setList[set_num-1]);
                    // 실제 파일 삭제
                    unlink(rminfo->path);

```

```

printf("W"%sW" has
been deleted in #%%dWnWn", rminfo->path, set_num);

break;

}
else

ListNext(&setList[set_num-1], &pinfo);

}
// 삭제 후 중복 리스트 내 파일이 1개만

// 중복파일리스트에서 제거
if (ListCount(&setList[set_num-1]) <

2) {

for (int i = set_num; i <=

setIdx; i++)

setList[i-1] =

setList[i];

}

}
// 삭제 후 중복 리스트 출력
showFileList(&setList[set_num-1], set_num);
}
else { // 인자의 개수가 맞지 않거나 범위를 벗어난 경우
fprintf(stderr, "argument or index range
errorWn");

}

}

else if (strcmp(token[1], "t") == 0) { // t옵션 설정
char trash[PATHMAX];
time_t max = 0;
Info * temp; // 최근 수정 파일 정보 임시

// 휴지통 절대경로 생성
sprintf(trash, "%s/%s/%s", "/home", getenv("USER"),
".local/share/Trash/files");

// 최근 수정된 파일 외 나머지 파일 휴지통 이동
// 최종 수정시간 동일할 시 이전의 값 유지
if (ListFirst(&setList[set_num-1], &pinfo)) {
while (ListCount(&setList[set_num-1]) > 1)
{
char *sub;
char newpath[PATHMAX];
// 파일명 추출
sub = strrchr(pinfo->path, '/');
// 휴지통으로 연결된 경로 생성
strcpy(newpath, trash);
strcat(newpath, sub);

```

```

// 현재 파일이 최근 수정된 파일인 경우
if (max < pinfo->mtime) {
    max = pinfo->mtime;
    temp = pinfo;
}
else { // 수정 시간이 오래된 경우
        // 중복 파일 리스트에서 수정

시간이 오래된 파일 제거

ListRemove(&setList[set_num-1]);

// 현재 파일의 위치를

휴지통으로 변경, 기존 위치의 파일 삭제

// 동명의 파일이 이미 존재할

시 계속 진행(해시값, 이름 모두 동일)

if (link(pinfo->path,
newpath) < 0) {

        continue;
    }
    unlink(pinfo->path);
}
ListNext(&setList[set_num-1],
&pinfo);
}
}
// 삭제 결과 출력
printf("All files in #%d have moved to Trash except
W"%sW" (%s)WnWn", set_num, temp->path, printTime(temp->mtime));
// 삭제 후 중복 리스트 내 파일이 1개만 존재
// 중복파일리스트에서 제거
if (ListCount(&setList[set_num-1]) < 2) {

    for (int i = set_num; i < setIdx; i++) {
        setList[i-1] = setList[i];
    }
    setIdx--;
}
// 휴지통으로 파일 이동 후 출력
for (int i = 0; i < setIdx; i++) {
    showFileList(&setList[i], i);
}
}
else if (strcmp(token[1], "i") == 0) // i 옵션 설정
{
    Info *rminfo; // 삭제된 파일 정보
    int except = FALSE; // y/n 이외의 입력이 들어온

경우

if (ListFirst(&setList[set_num-1], &pinfo)) {
    for (int i = 0; i < set_num; i++) {

```

여부 입력

pinfo->path);

// 파일 삭제

ListRemove(&setList[set\_num-1]);

ListNext(&setList[set\_num-1], &pinfo);

파일 유지

ListNext(&setList[set\_num-1], &pinfo);

이상

set\_num-1);

char ans;

// 삭제

// 현재 파일 삭제 여부 결정

memset(&ans, 0, 1);

printf("Delete W"%sW"? [y/n] ",

scanf("%c", &ans);

if (ans == 'y' || ans == 'Y') {

rminfo =

unlink(rminfo->path);

}

else if (ans == 'n' || ans == 'N') { //

}

else { // 이외의 입력이 들어온 경우

except = TRUE;

break;

}

}

}

// y/n 이외의 입력이 들어온 경우

if (except == TRUE)

break;

else {

// 삭제 후 중복 리스트 내 파일이 1개만 존재

// 중복파일리스트에서 제거

if (ListCount(&setList[set\_num-1]) < 2) {

for (int i = set\_num; i < setIdx; i++)

{

setList[i-1] = setList[i];

}

setIdx--;

}

else { // 현재 중복리스트에 남은 파일이 2개

showFileList(&setList[set\_num-1],

}

}

}

else if (strcmp(token[1], "f") == 0) {

// f옵션 설정

저장

```
time_t max = 0;
Info * temp; // 최근 수정 파일 정보 임시

// 최종 수정시간 동일할 시 이전의 값 유지
if (ListFirst(&setList[set_num-1], &pinfo)) {
    while (ListCount(&setList[set_num-1]) > 1)
    {
        char *sub;
        char newpath[PATHMAX];
        // 현재 파일이 최근 수정된 파일인 경우
        if (max < pinfo->mtime) {
            max = pinfo->mtime;
            temp = pinfo;
        }
        else { // 수정 시간이 오래된 경우
            // 중복 파일 리스트에서 수정

시간이 오래된 파일 제거

ListRemove(&setList[set_num-1]);

                                unlink(pinfo->path);
                                }
                                ListNext(&setList[set_num-1],
&pinfo);
                                }
                                }
// 삭제 결과 출력
printf("Left file in #%d : %s (%s)\n\n", set_num,
temp->path, printTime(temp->mtime));
// 삭제 후 중복 리스트 내 파일이 1개만 존재
// 중복파일리스트에서 제거
if (ListCount(&setList[set_num-1]) < 2) {
    for (int i = set_num; i < setIdx; i++) {
        setList[i-1] = setList[i];
    }
    setIdx--;
}
// 나머지 중복 파일 리스트 출력
for (int i = 0; i < setIdx; i++) {
    showFileList(&setList[i], i);
}
}
else {
    fprintf(stderr, "option error\n");
}
}
// 파일 리스트에 할당된 메모리 해제
if (ListFirst(&list, &pinfo)) {
    ListRemove(&list);
```



```

        while (ListNext(&list, &pinfo))
            ListRemove(&list);
    }
    // 중복 파일 리스트의 각 세트별로 할당된 메모리 해제
    for (int i = 0; i < setIdx; i++) {
        if (ListFirst(&setList[i], &pinfo)) {
            pinfo = ListRemove(&setList[i]);
            free(pinfo);
            while (ListNext(&setList[i], &pinfo)) {
                pinfo = ListRemove(&setList[i]);
                free(pinfo);
            }
        }
    }
}
exit(0);
}

// 파일 1개의 md5해시값을 구하는 함수
void get_md5Hash(char *filename, unsigned char *md)
{
    MD5_CTX c;
    int i, err=0;
    int fd; // 읽어들 파일에 대한 파일 디스크립터
    static unsigned char buf[BUFSIZE]; // 읽어들 파일의 데이터
    FILE *IN;

    // 인자로 지정된 파일 오픈
    IN = fopen(filename, "r");
    if (IN == NULL)
    {
        fprintf(stderr, "fopen error for %s\n", filename);
        return;
    }
    fd = fileno(IN); // 파일 디스크립터 생성
    MD5_Init(&c); // md5_ctx 구조체 초기화
    for (;;)
    {
        i = read(fd, buf, BUFSIZE); // BUFSIZE만큼 파일 데이터 패딩
        if (i <= 0)
            break;
        MD5_Update(&c, buf, (unsigned long)i); // 패딩된 데이터에 대한
        해시값 부여
    }
    MD5_Final(&(md[0]), &c); // 해시 추출 결과를 md에 저장
    fclose(IN);
}

// 현재 파일의 해시값 출력
void put_md5Hash(unsigned char *md)
{
    for (int i = 0; i < MD5_DIGEST_LENGTH; i++) {

```

```

        printf("%02x",md[i]);
    }
}
// 중복 파일 리스트 탐색: 해시값이 바뀌는 지점에서 구분
int get_identicalFiles(List *plist, int setIdx)
{
    Info * pinfo;
    if (ListFirst(plist, &pinfo)) {
        // 현재 파일 = 첫번째 파일, 다음 파일의 해시값도 동일
        if (memcmp(pinfo->hash, plist->cur->next->data->hash, HASHMAX) ==
0) {
            ListInsert(&setList[setIdx], pinfo);
        }
        while (ListNext(plist, &pinfo) && plist->cur->next != NULL) {
            // 직전 파일 해시값 != 현재 파일 해시값 && 현재 해시값 == 다음
해시값(중복 리스트의 시작)
            if (memcmp(pinfo->hash, plist->before->data->hash,
HASHMAX) != 0
&& memcmp(pinfo->hash,
plist->cur->next->data->hash, HASHMAX) == 0) {
                ListInsert(&setList[setIdx], pinfo);
            }
            // 직전 해시값 == 현재 해시값 == 다음 해시값
            else if (memcmp(pinfo->hash, plist->before->data->hash,
HASHMAX) == 0
&& memcmp(pinfo->hash,
plist->cur->next->data->hash, HASHMAX) == 0) {
                ListInsert(&setList[setIdx], pinfo);
            }
            // 직전 해시값 == 현재 해시값 && 현재 해시값 != 다음
해시값(중복 리스트의 끝)
            else if (memcmp(pinfo->hash, plist->before->data->hash,
HASHMAX) == 0
&& memcmp(pinfo->hash,
plist->cur->next->data->hash, HASHMAX) != 0) {
                ListInsert(&setList[setIdx], pinfo);
                setIdx++; // 중복 파일 리스트의 총 개수 증가
            }
            else;
        }
    }
    return setIdx;
}
}
}
// 시작 디렉토리의 모든 하위 파일을 조회하여 조건에 맞는 파일을 리스트에 추가
void searchFiles(char *dirpath, int depth, List *plist)
{
    int count, idx = 0;
    char *ptr;
    char curpath[PATHMAX]; // 현재 경로 저장
    unsigned char md5[HASHMAX]; // md5 해시값 저장용
    Queue listq; // 너비우선탐색용 큐

```

```

Info *pinfo;                                // 리스트에 저장할 파일 정보
struct stat statbuf;
struct dirent **namelist;  // 현재 디렉토리 내 하위 파일 리스트
int start = 0;

QueueInit(&listq);
// 시작 디렉토리의 값을 큐에 저장
pinfo = (Info *)malloc(sizeof(Info));
strcpy(pinfo->path, dirpath);
pinfo->depth = depth;
Enqueue(&listq, pinfo);

// 현재 큐가 비어 있는지 확인
while (!QEmpty(&listq)) {
    // 큐에 저장된 가장 오래된 데이터 가져오기
    Info *curInfo = Dequeue(&listq);
    if (access(curInfo->path, F_OK) != 0) {
        // for Linux Permission denied
        if (errno == 13)
            return;
        fprintf(stderr, "access error for %s\n", dirpath);
        exit(1);
    }
    // 디렉토리 내 하위 파일 이름을 사전순 정렬
    if ((count = scandir(curInfo->path, &namelist, NULL, alphasort)) < 0) {
        fprintf(stderr, "scandir error for %s\n", curInfo->path);
        exit(1);
    }
    else if (count == 0)
        return;
    // 파일 개수만큼 하위 파일 탐색
    for (int i = start; i < count; i++) {
        if (!strcmp(namelist[i]->d_name, ".") ||
            !strcmp(namelist[i]->d_name, ".."))
            continue;
        // 시작 디렉토리가 루트 디렉토리인지 여부에 따른 하위 경로 설정
        방식
        if (strcmp(curInfo->path, "/") == 0) {                // 시작
            디렉토리 = 루트 디렉토리
            if (strcmp(namelist[i]->d_name, "proc") == 0 ||
                strcmp(namelist[i]->d_name, "run")
                == 0 ||
                strcmp(namelist[i]->d_name, "sys")
                == 0) {
                continue;
            }
            sprintf(curpath, "%s%s", curInfo->path,
namelist[i]->d_name);

```

```

    }
    else { // 시작 디렉토리 != 루트 이외의 디렉토리
        sprintf(curpath, "%s/%s", curInfo->path,
namelist[i]->d_name);
    }
    // 현재 파일의 정보 가져오기
    if ((lstat(curpath, &statbuf) < 0) && (!access(curpath, F_OK))) {
        fprintf(stderr, "lstat error for %s\n", curpath);
        exit(1);
    }
    // 정규파일인 경우 파일의 크기 및 해시값을 리스트에 저장
    if (S_ISREG(statbuf.st_mode)) {
        // 파일의 크기가 지정된 범위에 해당
        if (statbuf.st_size >= minsize && statbuf.st_size <=
maxsize) {

            // 확장자명을 따로 지정한 경우
            if (strcmp(ext, "") != 0) {
                // 파일의 확장자명이 일치하는 경우
                if ((ptr = strrchr(curpath, '.')) !=
NULL && strcmp(ext, ptr) == 0) {

                    pinfo = (Info
*)malloc(sizeof(Info));

                    // 파일 주소
                    strcpy(pinfo->path, curpath);
                    // 해시값
                    get_md5Hash(curpath, md5);
                    memcpy(pinfo->hash, md5,
sizeof(unsigned char) * HASHMAX);

                    // 파일 크기
                    pinfo->size = statbuf.st_size;
                    // 최종 수정 및 접근 시간
                    pinfo->mtime =
statbuf.st_mtime;
                    pinfo->atime =
statbuf.st_atime;

                    // 파일 절대경로의 길이
                    pinfo->depth =
curInfo->depth + 1;

                    // 연결리스트에 저장
                    ListInsert(plist, pinfo);
                }
            }
        }
    }
    else { // 확장자명을 따로 지정하지 않은 경우
        get_md5Hash(curpath, md5);
        pinfo = (Info *)malloc(sizeof(Info));
        strcpy(pinfo->path, curpath);
        memcpy(pinfo->hash, md5,
sizeof(unsigned char) * HASHMAX);
    }
}

```

```

        pinfo->size = statbuf.st_size;
        pinfo->mtime = statbuf.st_mtime;
        pinfo->atime = statbuf.st_atime;
        pinfo->depth = curInfo->depth + 1;
        ListInsert(plist, pinfo);
    }
}
}
else if (S_ISDIR(statbuf.st_mode)) { // 디렉토리인 경우
    큐에 추가
        pinfo = (Info *)malloc(sizeof(Info));
        strcpy(pinfo->path, curpath); // 하위
        경로 생성용 절대경로
        pinfo->depth = curInfo->depth + 1; // 하위
        파일 절대경로 길이 계산용
        Enqueue(&listq, pinfo);
    }
    else
        ;
    // 동일 파일 내 하위 디렉토리가 다른 하위디렉토리의 절대경로에
    연결되는 것 방지
    strcpy(curpath, curInfo->path);
}
}
}
// 중복 파일리스트 출력
void showFileList(List *pset, int setnum)
{
    struct stat statbuf;
    int index = 1;
    Info * pinfo;
    if (ListFirst(pset, &pinfo)) {
        // 각 중복 파일 리스트의 세트 번호, 파일 크기, 해시값 출력
        printf("--- Identical files #%d (", setnum+1);
        putCommaToSize(pinfo->size);
        printf(" bytes - ");
        put_md5Hash(pinfo->hash); // 사용되는 해시함수 차이로 인해 따로
        정의
        printf(") ----Wn");
        // 중복 파일 리스트에 저장된 파일 정보 출력
        printf("[%d] %s (mtime : %s) (atime : %s)Wn", index++, pinfo->path,
            printTime(pinfo->mtime), printTime(pinfo->atime));
        while (ListNext(pset, &pinfo)) {
            printf("[%d] %s (mtime : %s) (atime : %s)Wn", index++,
                pinfo->path,
                printTime(pinfo->mtime),
                printTime(pinfo->atime));
        }
        printf("Wn");
    }
}

```

```

    }
}
// 정렬 기준 설정: 최종 수정 시간은 내림차순, 나머지는 오름차순
int sort_identical(LData a, LData b)
{
    // 정렬 우선순위: 파일 크기>해시값>절대경로 길이>파일명>최종 수정시간>접근시간 순
    // 크기 정렬
    if (a->size > b->size)
        return 1;           // 내림차순
    else if (a->size < b->size)
        return -1;          // 오름차순
    else {
        // 해시값 순서 기준 정렬
        if (memcmp(a->hash, b->hash, HASHMAX) > 0)
            return 1;
        else if (memcmp(a->hash, b->hash, HASHMAX) < 0)
            return -1;
        else {
            // 절대경로 길이 기준 정렬
            if (a->depth > b->depth)
                return 1;
            else if (a->depth < b->depth)
                return -1;
            else {
                // 파일명 기준 정렬
                if (strcmp(a->path, b->path) > 0)
                    return 1;
                else if (strcmp(a->path, b->path) < 0)
                    return -1;
            }
        }
    }
}
return 0;
}

```

#### 4) ssu\_find-sha1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <dirent.h>
#include <errno.h>
#include <limits.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <openssl/sha.h>
#include "ssu_functions.h"

char *ext;           // 추출한 파일 확장자명
long int minsize;
long int maxsize;
char dirPath[BUFMAX]; // [TARGET_DIRECTORY]의 절대경로

void get_md5Hash(char *filename, unsigned char *md); // 파일로부터 md5 해시값
추출
void put_md5Hash(unsigned char *md);                // 해시값 출력
List setList[LISTMAX];
int setIdx = 0;

int main(int argc, char *argv[])
{
    char *ptr;
    double min, max;
    char dirName[BUFMAX];
    struct stat statbuf;
    struct timeval begin_t, end_t;
    long int size;
    unsigned char *hash;
    List list;
    Info * pinfo;

    // [FILE_EXTENSION] 인자 점검
    if (argv[1][0] == '*') {
        if (strlen(argv[1]) == 1) { // "*"만 입력
            ext = argv[1] + 1;
        }
        else if (argv[1][1] == '.' && strlen(argv[1]) > 2) { // ".*(확장자명)"
            ext = argv[1] + 1;
        }
        else { // 입력된 것이 "*.나 "asdsds" 같은 경우
            fprintf(stderr, "file extension error\n");
            exit(1);
        }
    }
```

```

    }
    else { // "*"나 "*(확장자명)" 이외의 입력
        fprintf(stderr, "file extension error!Wn");
        exit(1);
    }

    // [MINSIZE] 인자 점검
    // 실수로 변환할 숫자가 없는 경우
    if ((min = strtod(argv[2], &ptr)) == 0) {
        if (strcmp(argv[2], "~") == 0) { // "~" 입력
            minsize = 1;
        }
        else { // "~" 외 다른 문자열 입력
            fprintf(stderr, "minsize error!Wn");
            exit(1);
        }
    }
    else if (strcmp(ptr, "W0") != 0) { // 실수 변환 후 문자열 존재
        if (strcmp(ptr, "KB") == 0 || strcmp(ptr, "kb") == 0) // KB 단위 byte
            변환
            {
                min *= 1024;
            }
        else if (strcmp(ptr, "MB") == 0 || strcmp(ptr, "mb") == 0) // MB단위
            byte 변환
            {
                min *= 1024*1024;
            }
        else if (strcmp(ptr, "GB") == 0 || strcmp(ptr, "gb") == 0) // GB단위
            byte 변환
            {
                min *= 1024*1024*1024;
            }
        else {
            fprintf(stderr, "minsize error!Wn");
            exit(1);
        };
        minsize = (long)min;
    }
    else
        minsize = (long)min;

    // [MAXSIZE] 인자 점검
    // 숫자 없이 "~"만 입력된 경우
    if ((max = strtod(argv[3], &ptr)) == 0) {
        if (strcmp(argv[3], "~") == 0) { // "~" 입력
            maxsize = LONG_MAX; // long형의 최대치로 설정
        }
    }

```



```

        else { // "~" 외 다른 문자열 입력
            fprintf(stderr, "maxsize error!Wn");
            exit(1);
        }
    }
    else if (strcmp(ptr, "W0") != 0) { // 실수 변환 후 문자열 존재
        if (strcmp(ptr, "KB") == 0 || strcmp(ptr, "kb") == 0) // KB 단위 byte
            변환
            {
                max *= 1024;
            }
        else if (strcmp(ptr, "MB") == 0 || strcmp(ptr, "mb") == 0) // MB단위
            byte 변환
            {
                max *= 1024*1024;
            }
        else if (strcmp(ptr, "GB") == 0 || strcmp(ptr, "gb") == 0) // GB단위
            byte 변환
            {
                max *= 1024*1024*1024;
            }
        else {
            fprintf(stderr, "maxsize errorWn");
            exit(1);
        }
        maxsize = (long)max;
    }
    else {
        maxsize = (long)max;
    }
    // minsize가 maxsize보다 큰 경우 에러 처리
    if (minsize > maxsize) {
        fprintf(stderr, "minsize must not be bigger than maxsizeWn");
        exit(1);
    }

    // [TARGET_DIRECTORY] 인자 점검
    if ((ptr = strstr(argv[4], "~")) != NULL) { // "~(홈 디렉토리)"를 포함한 경로인
        경우
        sprintf(dirName, "%s/%s%s", "/home", getenv("USER"), ptr+1);
    }
    else {
        ptr = argv[4];
        strcpy(dirName, ptr);
    }
    // 입력한 경로를 절대경로로 변환, 실제로 유효한 경로가 아니면 에러 처리
    if (realpath(dirName, dirPath) == NULL) {
        fprintf(stderr, "%s is not existWn", dirPath);
    }

```

```

        exit(1);
    }
    // 입력한 경로의 파일이 디렉토리인지 확인
    if (lstat(dirPath, &statbuf) < 0) {
        fprintf(stderr, "lstat error for %sWn", dirPath);
        exit(1);
    }
    if (!S_ISDIR(statbuf.st_mode)) {
        fprintf(stderr, "Path is not directoryWn");
        exit(1);
    }

    // 검색 시간 측정
    gettimeofday(&begin_t, NULL);

    // 지정된 디렉토리의 모든 하위 파일들에 대한 파일 리스트 초기화
    // 중복 파일 리스트 초기화 및 정렬 규칙 설정
    ListInit(&list);
    SetSortRule(&list, sort_identical);
    for (int i = 0; i < LISTMAX; i++) {
        ListInit(&setList[i]);
        SetSortRule(&setList[i], sort_identical);
    }

    // 명령행 인자의 조건을 충족하는 전체 파일 리스트 생성
    searchFiles(dirPath, 0, &list);
    // 전체 파일에서 중복 리스트 추출: 현재 노드 기준 직후 노드의 해시값이 다르면 재귀
    setIdx = get_identicalFiles(&list, setIdx);

    // 중복 파일 리스트가 존재하지 않는 경우
    if (setIdx == 0) {
        printf("No duplicates in %sWn", dirPath);
        gettimeofday(&end_t, NULL);
        ssu_searchTime(&begin_t, &end_t); // 전체 검색 시간 출력
    }
    else {
        for (int i = 0; i < setIdx; i++) {
            showFileList(&setList[i], i);
        }
        gettimeofday(&end_t, NULL);
        ssu_searchTime(&begin_t, &end_t); // 전체 검색 시간 출력

        // >> [SET_INDEX] [OPTION ... ] 작업 수행
        while (1) {
            char cmd[BUFMAX]; // 명령문
            int set_num; // 세트 번호
            int list_num; // 한 세트 내 파일 번호
            int cnt; // 명령문 파싱용

```

```

char *token[TOKMAX];
Info *pinfo;
// 삭제를 수행할 세트 번호 및 옵션 입력
memset(cmd, 0, BUFMAX);
printf(">> ");
fgets(cmd, BUFMAX, stdin);
// 입력이 없으면 에러 처리
if (strcmp(cmd, "\n") == 0) {
    fprintf(stderr, "usage: [SET_INDEX] [OPTION ...
]Wn");

    continue;
}
else if (strcmp(cmd, "exitWn") == 0) {
    printf(">> Back to PromptWn");
    break;
}
else
    cmd[strlen(cmd)-1] = '\0';
// 인자의 개수는 최소 2개 이상
if ((cnt = parseCmd(cmd, token)) < 2) {
    fprintf(stderr, "usage: [SET_INDEX] [OPTION ...
]Wn");

    continue;
}
// 첫번째 인자가 인덱스 범위 내 숫자인지 확인
if ((set_num = atoi(token[0])) <= 0 || set_num > setIdx) {
    fprintf(stderr, "set index errorWn");
    continue;
}
if (strcmp(token[1], "d") == 0) { // d옵션 설정
    int index = 1; // 인덱스 번호
    Info * rminfo; // 삭제된 데이터
    // [LIST_IDX] 에 해당하는 파일 삭제
    if (cnt > 2 && (list_num = atoi(token[2])) > 0 &&
        list_num <
ListCount(&setList[set_num-1])) {
        // 지정한 파일 제거
        if (ListFirst(&setList[set_num-1], &pinfo)) {
            for (int i = 1; i <=
ListCount(&setList[set_num-1]); i++) {
                // 현재 인덱스 번호가
                [LIST_IDX]와 동일
                if (i == list_num) {
                    // 중복 파일
                    리스트에서 삭제
                    rminfo =
ListRemove(&setList[set_num-1]);
                }
            }
        }
    }
    // 실제 파일 삭제

```

```

unlink(rminfo->path);

printf("W"%sW" has
been deleted in #%%dWnWn", rminfo->path, set_num);

break;
}
else

ListNext(&setList[set_num-1], &pinfo);
}
// 삭제 후 중복 리스트 내 파일이 1개만
존재
// 중복파일리스트에서 제거
if (ListCount(&setList[set_num-1]) <
2) {
if
(ListCount(&setList[set_num-1]) == 1)
ListRemove(&setList[set_num-1]);
for (int i = set_num; i <=
setIdx; i++)
setList[i-1] =
setList[i];
}
}
// 삭제 후 중복 리스트 출력
showFileList(&setList[set_num-1], set_num);
}
else { // 인자의 개수가 맞지 않거나 범위를 벗어난 경우
fprintf(stderr, "argument or index range
errorWn");
}
}
else if (strcmp(token[1], "t") == 0) { // t옵션 설정
char trash[PATHMAX];
time_t max = 0;
Info * temp; // 최근 수정 파일 정보 임시
저장
// 휴지통 절대경로 생성
sprintf(trash, "%s/%s/%s", "/home", getenv("USER"),
".local/share/Trash/files");
// 최근 수정된 파일 외 나머지 파일 휴지통 이동
// 최종 수정시간 동일할 시 이전의 값 유지
if (ListFirst(&setList[set_num-1], &pinfo)) {
while (ListCount(&setList[set_num-1]) > 1)
{
char *sub;
char newpath[PATHMAX];

```

```

// 파일명 추출
sub = strrchr(pinfo->path, '/');
// 휴지통으로 연결된 경로 생성
strcpy(newpath, trash);
strcat(newpath, sub);

// 현재 파일이 최근 수정된 파일인 경우
if (max < pinfo->mtime) {
    max = pinfo->mtime;
    temp = pinfo;
}
else { // 수정 시간이 오래된 경우
    // 중복 파일 리스트에서 수정
시간이 오래된 파일 제거

ListRemove(&setList[set_num-1]);

휴지통으로 변경, 기존 위치의 파일 삭제

시 계속 진행(해시값, 이름 모두 동일)

newpath) < 0) {

continue;
}
unlink(pinfo->path);
}
ListNext(&setList[set_num-1],
&pinfo);
}

}

// 삭제 결과 출력
printf("All files in #%d have moved to Trash except
W"%sW" (%s)WnWn", set_num, temp->path, printTime(temp->mtime));

// 삭제 후 중복 리스트 내 파일이 1개만 존재
// 중복파일리스트에서 제거
if (ListCount(&setList[set_num-1]) < 2) {
    if (ListCount(&setList[set_num-1]) == 1)
        ListRemove(&setList[set_num-1]);
    for (int i = set_num; i < setIdx; i++) {
        setList[i-1] = setList[i];
    }
    setIdx--;
}
// 휴지통으로 파일 이동 후 출력
for (int i = 0; i < setIdx; i++) {
    showFileList(&setList[i], i);

```

```

    }
}
else if (strcmp(token[1], "i") == 0) // i옵션 설정
{
    Info *rminfo; // 삭제된 파일 정보
    int except; // y/n 이외의 입력이 들어온

    if (ListFirst(&setList[set_num-1], &pinfo)) {
        for (int i = 0; i < set_num; i++) {
            char ans[2]; // 삭제

            // 현재 파일 삭제 여부 결정
            memset(ans, 0, 1);
            printf("Delete W"%sW"? [y/n] ",
                pinfo->path);

            fgets(ans, 1, stdin);
            ans[1] = '\0';
            if (ans[0] == 'y' || ans[0] == 'Y') {

                // 파일 삭제

                rminfo =
                    ListRemove(&setList[set_num-1]);

                unlink(rminfo->path);
                except = FALSE;

                ListNext(&setList[set_num-1], &pinfo);
            }
            else if (ans[0] == 'n' || ans[0] ==
                'N') { // 파일 유지

                except = FALSE;

                ListNext(&setList[set_num-1], &pinfo);

                continue;
            }
            else { // 이외의 입력이 들어온 경우
                except = TRUE;
                break;
            }
        }
    }

    // y/n 이외의 입력이 들어온 경우
    if (except == TRUE)
        break;
    else {
        // 삭제 후 중복 리스트 내 파일이 1개만 존재
        // 중복파일리스트에서 제거
        if (ListCount(&setList[set_num-1]) < 2) {
            if (ListCount(&setList[set_num-1]) ==

```

```

ListRemove(&setList[set_num-1]);

{
    for (int i = set_num; i < setIdx; i++)
        setList[i-1] = setList[i];
    setIdx--;
}
else { // 현재 중복리스트에 남은 파일이 2개
    showFileList(&setList[set_num-1],
set_num-1);
}
}
else if (strcmp(token[1], "f") == 0) { // f옵션 설정
    time_t max = 0;
    Info * temp; // 최근 수정 파일 정보 임시
    // 최종 수정시간 동일할 시 이전의 값 유지
    if (ListFirst(&setList[set_num-1], &pinfo)) {
        while (ListCount(&setList[set_num-1]) > 1)
        {
            char *sub;
            char newpath[PATHMAX];
            // 현재 파일이 최근 수정된 파일인 경우
            if (max < pinfo->mtime) {
                max = pinfo->mtime;
                temp = pinfo;
            }
            else { // 수정 시간이 오래된 경우
                // 중복 파일 리스트에서 수정
시간이 오래된 파일 제거
            }
        }
        ListRemove(&setList[set_num-1]);
        unlink(pinfo->path);
    }
    ListNext(&setList[set_num-1],
&pinfo);
}
}
// 삭제 결과 출력
printf("Left file in #%d : %s (%s)\n\n", set_num,
temp->path, printTime(temp->mtime));
// 삭제 후 중복 리스트 내 파일이 1개만 존재
// 중복파일리스트에서 제거
if (ListCount(&setList[set_num-1]) < 2) {
    if (ListCount(&setList[set_num-1]) == 1)

```

```

        ListRemove(&setList[set_num-1]);
        for (int i = set_num; i < setIdx; i++) {
            setList[i-1] = setList[i];
        }
        setIdx--;
    }
    // 나머지 중복 파일 리스트 출력
    for (int i = 0; i < setIdx; i++) {
        showFileList(&setList[i], i);
    }
}
else {
    fprintf(stderr, "option error\n");
}
}

// 파일 리스트에 할당된 메모리 해제
if (ListFirst(&list, &pinfo)) {
    ListRemove(&list);
    while (ListNext(&list, &pinfo))
        ListRemove(&list);
}
// 중복 파일 리스트의 각 세트별로 할당된 메모리 해제
for (int i = 0; i < setIdx; i++) {
    if (ListFirst(&setList[i], &pinfo)) {
        pinfo = ListRemove(&setList[i]);
        free(pinfo);
        while (ListNext(&setList[i], &pinfo)) {
            pinfo = ListRemove(&setList[i]);
            free(pinfo);
        }
    }
}

}
exit(0);
}

// 파일 1개의 sha1해시값을 구하는 함수
void get_sha1Hash(char *filename, unsigned char *sh)
{
    SHA_CTX c;
    int i, err = 0;
    int fd;
    static unsigned char buf[BUFSIZE];
    FILE *IN;
    // 인자로 지정된 파일 오픈
    IN = fopen(filename, "r");
    if (IN == NULL)
    {
        // 읽어들 파일에 대한 파일 디스크립터
        // 읽어들 파일의 데이터
    }
}

```



```

        fprintf(stderr, "fopen error for %s\n", filename);
        return;
    }
    fd = fileno(IN); // 파일 디스크립터 생성
    SHA1_Init(&c); // sha1_ctx 구조체 초기화
    for (;;)
    {
        i = read(fd, buf, BUFSIZE); // BUFSIZE만큼 파일 데이터 패딩
        if (i <= 0)
            break;
        SHA1_Update(&c, buf, (unsigned long)i); // 패딩된 데이터에 대한
해시값 부여
    }
    SHA1_Final(&(sh[0]),&c); // 해시 추출 결과를 sh에 저장
    fclose(IN);
}

void put_sha1Hash(unsigned char *sh) // sha1 해시값 출력
{
    for (int i = 0; i < SHA_DIGEST_LENGTH; i++) {
        printf("%02x", sh[i]);
    }
}

// 중복 파일 리스트 탐색: 해시값이 바뀌는 지점에서 분할
int get_identicalFiles(List *plist, int setIdx)
{
    Info * pinfo;
    if (ListFirst(plist, &pinfo)) {
        // 현재 파일 = 첫번째 파일, 다음 파일의 해시값도 동일
        if (memcmp(pinfo->hash, plist->cur->next->data->hash, HASHMAX) ==
0) {
            ListInsert(&setList[setIdx], pinfo);
        }
        while (ListNext(plist, &pinfo) && plist->cur->next != NULL) {
            // 직전 파일 해시값 != 현재 파일 해시값이고 현재 해시값 == 다음
해시값(중복 리스트의 시작)
            if (memcmp(pinfo->hash, plist->before->data->hash,
HASHMAX) != 0
&& memcmp(pinfo->hash,
plist->cur->next->data->hash, HASHMAX) == 0) {
                ListInsert(&setList[setIdx], pinfo);
            }
            // 직전 해시값 == 현재 해시값 == 다음 해시값
            else if (memcmp(pinfo->hash, plist->before->data->hash,
HASHMAX) == 0
&& memcmp(pinfo->hash,
plist->cur->next->data->hash, HASHMAX) == 0) {
                ListInsert(&setList[setIdx], pinfo);
            }
            // 직전 해시값 == 현재 해시값이고 현재 해시값 != 다음
해시값(중복 리스트의 끝)

```

```

else if (memcmp(pinfo->hash, plist->before->data->hash,
HASHMAX) == 0
&& memcmp(pinfo->hash,
plist->cur->next->data->hash, HASHMAX) != 0) {
    ListInsert(&setList[setIdx], pinfo);
    setIdx++;
}
else;
}
}
return setIdx;
}
// 시작 디렉토리의 모든 하위 파일 및 디렉토리 순회
// 명령행 인자의 조건을 충족하는 전체 파일 리스트 생성
void searchFiles(char *dirpath, int depth, List *plist)
{
    int count, idx = 0;
    char *ptr;
    char curpath[PATHMAX];          // 현재 경로 저장
    unsigned char sha1[HASHMAX];    // md5 해시값 저장용
    Queue listq;                    // 너비우선탐색용 큐
    Info *pinfo;                    // 리스트에 저장할 파일 정보
    struct stat statbuf;
    struct dirent **namelist;       // 현재 디렉토리 내 하위 파일 리스트
    int start = 0;

    QueueInit(&listq);
    // 시작 디렉토리의 값을 큐에 저장
    pinfo = (Info *)malloc(sizeof(Info));
    strcpy(pinfo->path, dirpath);
    pinfo->depth = depth;
    Enqueue(&listq, pinfo);

    // 현재 큐가 비어 있는지 확인
    while (!QEmpty(&listq)) {
        // 큐에 저장된 가장 오래된 데이터 가져오기
        Info *curInfo = Dequeue(&listq);
        if (access(curInfo->path, F_OK) != 0) {
            // for Linux Permission denied
            if (errno == 13)
                return;
            fprintf(stderr, "access error for %s\n", dirpath);
            exit(1);
        }

        // 디렉토리 내 하위 파일 이름을 사전순 정렬
        if ((count = scandir(curInfo->path, &namelist, NULL, alphasort)) < 0) {
            fprintf(stderr, "scandir error for %s\n", curInfo->path);
            exit(1);
        }
    }
}

```

```

    }
    else if (count == 0)
        return;

    // 파일 개수만큼 하위 파일 탐색
    for (int i = start; i < count; i++) {
        if (!strcmp(namelist[i]->d_name, ".") ||
            !strcmp(namelist[i]->d_name, ".."))
            continue;
        // 시작 디렉토리가 루트 디렉토리인지 여부에 따른 하위 경로 설정
        방식
        if (strcmp(curInfo->path, "/") == 0) {
            if (strcmp(namelist[i]->d_name, "proc") == 0 ||
                strcmp(namelist[i]->d_name, "run")
                == 0 ||
                strcmp(namelist[i]->d_name, "sys")
                == 0) {
                continue;
            }
            sprintf(curpath, "%s%s", curInfo->path,
namelist[i]->d_name);
        }
        else {
            sprintf(curpath, "%s/%s", curInfo->path,
namelist[i]->d_name);
        }
        // 현재 파일의 정보 가져오기
        if ((lstat(curpath, &statbuf) < 0) && (!access(curpath, F_OK))) {
            fprintf(stderr, "lstat error for %s\n", curpath);
            exit(1);
        }

        // 정규파일인 경우 파일의 크기 및 해시값을 리스트에 저장
        if (S_ISREG(statbuf.st_mode)) {
            // 파일의 크기가 지정된 범위에 해당
            if (statbuf.st_size >= minsize && statbuf.st_size <=
maxsize) {
                // 확장자명을 따로 지정한 경우
                if (strcmp(ext, "") != 0) {
                    // 파일의 확장자명이 일치하는 경우
                    if ((ptr = strrchr(curpath, '.')) !=
NULL && strcmp(ext, ptr) == 0) {
                        pinfo = (Info
                        *)malloc(sizeof(Info));

                        strcpy(pinfo->path, curpath);
                        get_sha1Hash(curpath, sha1);
                        memcpy(pinfo->hash, sha1,
sizeof(unsigned char) * HASHMAX);
                    }
                }
            }
        }
    }
}

```

```

        pinfo->size = statbuf.st_size;
        pinfo->mtime =
statbuf.st_mtime;
        pinfo->atime =
statbuf.st_atime;
        pinfo->depth =
curInfo->depth + 1;
        // 연결리스트에 저장
        ListInsert(plist, pinfo);
    }
}
else { // 확장자명을 따로 지정하지 않은 경우
    get_sha1Hash(curpath, sha1);
    pinfo = (Info *)malloc(sizeof(Info));
    strcpy(pinfo->path, curpath);
    memcpy(pinfo->hash, sha1,
sizeof(unsigned char) * HASHMAX);
    pinfo->size = statbuf.st_size;
    pinfo->mtime = statbuf.st_mtime;
    pinfo->atime = statbuf.st_atime;
    pinfo->depth = curInfo->depth + 1;
    ListInsert(plist, pinfo);
}
}
}
else if (S_ISDIR(statbuf.st_mode)) { // 디렉토리인 경우 큐에 추가
    pinfo = (Info *)malloc(sizeof(Info));
    strcpy(pinfo->path, curpath);
    pinfo->depth = curInfo->depth + 1;
    Enqueue(&listq, pinfo);
}
else
    ;
// 동일 파일 내 하위 디렉토리가 다른 하위디렉토리의 절대경로에
연결되는 것 방지
    strcpy(curpath, curInfo->path);
}
}
}
// 파일리스트 출력
void showFileList(List *pset, int setnum)
{
    struct stat statbuf;
    int index = 1;
    Info * pinfo;
    if (ListFirst(pset, &pinfo)) {
        // 각 중복 파일 리스트의 세트 번호, 파일 크기, 해시값 출력
        printf("--- Identical files #%d (", setnum+ 1);
        putCommaToSize(pinfo->size);

```

```

        printf(" bytes - ");
        put_shalHash(pinfo->hash);
        printf(" ----\n");
        // 중복 파일 리스트에 저장된 파일 정보 출력
        printf("[%d] %s (mtime : %s) (atime : %s)\n", index++, pinfo->path,
                printTime(pinfo->mtime), printTime(pinfo->atime));
        while (ListNext(pset, &pinfo)) {
            printf("[%d] %s (mtime : %s) (atime : %s)\n", index++,
pinfo->path,
                printTime(pinfo->mtime),
printTime(pinfo->atime));
        }
        printf("\n");
    }
}

// 정렬 기준 설정(오름차순)
int sort_identical(LData a, LData b)
{
    // 정렬 우선순위: 파일 크기>해시값>절대경로 길이>파일명>최종 수정시간>접근시간 순
    // 크기 정렬
    if (a->size > b->size)
        return 1;          // 내림차순
    else if (a->size < b->size)
        return -1;         // 오름차순
    else {
        // 해시값 기준 정렬: 문자열처럼 사전순
        if (memcmp(a->hash, b->hash, HASHMAX) > 0)
            return 1;
        else if (memcmp(a->hash, b->hash, HASHMAX) < 0)
            return -1;
        else {
            // 절대경로 길이 기준 정렬
            if (a->depth > b->depth)
                return 1;
            else if (a->depth < b->depth)
                return -1;
            else { // 파일명 기준 정렬
                if (strcmp(a->path, b->path) > 0)
                    return 1;
                else if (strcmp(a->path, b->path) < 0)
                    return -1;
            }
        }
    }
}

return 0;
}

```

## 5) ssu\_functions.h

```
#ifndef FUNCTION_H
#define FUNCTION_H
#define TRUE 1
#define FALSE 0
#define ARGMAX 5          // ssu_sdup 실행 후 입력될 명령행 인자의 최대 개수
#define TOKMAX 3          // 삭제 명령 수행 시 명령행 인자의 최대 개수
#define BUFMAX 1024       // 버퍼의 크기
#define BUFSIZE 1024*16   // 해시 함수에서 한번에 읽어올 파일 데이터의 최대 길이
#define SEC_TO_MICRO 1000000 // 마이크로초 변환
#define FILEMAX 256       // 파일명의 길이
#define PATHMAX 4096      // 경로명의 길이
#define LISTMAX 10000
#ifdef HEADER_MD5_H
    #define HASHMAX 16    // openssl/md5.h가 정의되어 있으면 해시값 길이는 16
#else
    #define HASHMAX 20    // openssl/sha.h가 정의되어 있으면 해시값 길이는 20
#endif

// 파일 정보
typedef struct _fileinfo
{
    char path[BUFMAX];
    char hash[HASHMAX];
    time_t mtime;
    time_t atime;
    size_t size;
    int depth;
} Info;
typedef Info * LData;
// 연결리스트/큐를 구성할 노드
typedef struct _node
{
    LData data;
    struct _node * next;
} Node;
// 파일에 대한 연결리스트
typedef struct _linkedlist
{
    Node * head;
    Node * cur;
    Node * before;
    int (*comp)(LData d1, LData d2);
    int numOfData;
} LinkedList;
typedef LinkedList List;
// 너비우선탐색용 큐
typedef struct lqueue
{
    Node * front;
    Node * rear;
} LQueue;
```

```

typedef LQueue Queue;
int parseCmd(char *cmd, char *argv[]);           // 명령문 파싱
void ssu_searchTime(struct timeval *begin_t, struct timeval *end_t); // 경과 시간 측정
char *printTime(time_t ptime);                  // 시간 정보 출력
void putCommaToSize(long int size);             // 천단위 구분된 파일 크기 출력
void searchFiles(char *dirname, int depth, List * plist); // 시작 디렉토리 내 하위 파일
순회
int get_identicalFiles(List *setList, int setIdx); // 전체 파일 리스트 중 중복
파일 리스트 추출
int sort_identical(LData d1, LData d2);          // (중복)파일 리스트
정렬 기준
// 큐 구현
void QueueInit(Queue *pq);                      // 큐 초기화
int QEmpty(Queue * pq);                        // 비어 있는 큐인지 판별
void Enqueue(Queue * pq, LData data);          // 큐에 노드 추가
LData Dequeue(Queue * pq);                    // 큐에서 노드 제외
// 중복 파일 리스트 출력
void showFileList(List *pset, int setnum);
// 연결리스트 구현
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
void SetSortRule(List * plist, int (*comp)(LData d1, LData d2)); // 리스트 정렬 규칙
설정
#endif

```

## 6) ssu\_functions.c

```

// ssu_sdup.c, ssu_find-md5.c, ssu_find-sha1.c에서 공통으로 사용하는 함수 정의
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <dirent.h>
#include <errno.h>
#include <time.h>
#include <sys/time.h>
#include <sys/stat.h>
#include <sys/types.h>
#include "ssu_functions.h"
// 명령문 파싱
int parseCmd(char *cmd, char *argv[])
{
    int argc = 0;
    char *ptr = NULL;
    ptr = strtok(cmd, " ");
    while (ptr != NULL) {

```

```

        argv[argc++] = ptr;
        ptr = strtok(NULL, " ");
    }
    return argc;
}

// 총 검색시간 출력
void ssu_searchTime(struct timeval *begin_t, struct timeval *end_t)
{
    end_t->tv_sec -= begin_t->tv_sec;
    if (end_t->tv_usec < begin_t->tv_usec) {
        end_t->tv_sec--;
        end_t->tv_usec += SEC_TO_MICRO;
    }
    end_t->tv_usec -= begin_t->tv_usec;
    printf("Searching time: %ld:%06ld(sec:usec)\n\n", end_t->tv_sec,
end_t->tv_usec);
}

// 파일 크기 출력: 천 단위마다 구분, 바이트 단위 출력
void putCommaToSize(long int size)
{
    char sizebuf[BUFMAX];
    char *digits;
    int len;

    len = snprintf(sizebuf, sizeof(sizebuf), "%ld", size);
    digits = sizebuf;
    for (int i = len; i > 0; ) {
        printf("%c", *digits++);
        i--;
        if ((i % 3 == 0) && i > 0) {
            printf(",");
        }
    }
    return;
}

// 시간 출력
char *printTime(time_t ptime)
{
    char *time = (char *)malloc(sizeof(char) * BUFMAX);
    struct tm * t;
    t = localtime(&ptime);
    sprintf(time, "%04d-%02d-%02d %02d:%02d:%02d",
        (t->tm_year)+ 1900, (t->tm_mon)+ 1, t->tm_mday, t->tm_hour,
t->tm_min, t->tm_sec);
    return time;
}

// 큐 구현
// 큐 초기화
void QueueInit(Queue * pq)
{
    pq->front = NULL;
    pq->rear = NULL;
}

```



```

} // 큐가 비었는지 확인
int QEmpty(Queue * pq)
{
    if (pq->front == NULL)
        return TRUE;
    else
        return FALSE;
}

// 큐에 노드 추가
void Enqueue(Queue * pq, LData data)
{
    Node * newNode = (Node *)malloc(sizeof(Node));
    newNode->next = NULL;
    newNode->data = data;
    if (QEmpty(pq))
    {
        pq->front = newNode;
        pq->rear = newNode;
    }
    else {
        pq->rear->next = newNode;
        pq->rear = newNode;
    }
}

// 큐에서 노드 삭제
LData Dequeue(Queue *pq)
{
    Node * delNode;
    LData retData;
    if (QEmpty(pq))
    {
        printf("Queue Memory Error!Wn");
        exit(-1);
    }
    delNode = pq->front;
    retData = delNode->data;
    pq->front = pq->front->next;
    free(delNode);
    return retData;
}

// 연결리스트 초기화
void ListInit(List * plist)
{
    plist->head = (Node *)malloc(sizeof(Node));
    plist->head->next = NULL;
    plist->cur = plist->head;
    plist->comp = NULL;
    plist->numOfData = 0;
}

// 정렬 규칙 없이 리스트 노드 추가
void FreeInsert(List * plist, LData data)
{
    Node * newNode = (Node *)malloc(sizeof(Node));

```

```

        newNode->data = data;
        newNode->next = plist->head->next;
        plist->head->next = newNode;
        (plist->numOfData)++;
    }
    // 정렬 규칙을 적용하여 노드 추가
    void SortInsert(List * plist, LData data)
    {
        Node * newNode = (Node *)malloc(sizeof(Node));
        Node * pred = plist->head;
        newNode->data = data;
        while (pred->next != NULL && plist->comp(data, pred->next->data) != -1)
        {
            pred = pred->next;
        }
        newNode->next = pred->next;
        pred->next = newNode;
        (plist->numOfData)++;
    }
    // 노드 추가
    void ListInsert(List * plist, LData data)
    {
        if (plist->comp == NULL)
            FreeInsert(plist, data);
        else
            SortInsert(plist, data);
    }
    // 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;
}
// 정렬 규칙 수행용 함수 불러오기
void SetSortRule(List * plist, int (*comp)(LData d1, LData d2))
{
    plist->comp = comp;
}
```