

1. 과제 개요

본 과제는 중복 파일을 검색해 중복 데이터를 제거하는 'fdupes' 명령어를 확장하여 직접 구현해 보는 것이다. Fdupes는 리눅스에서 중복된 파일을 찾고 삭제하는데 사용하는 명령어이다.

ssu_sdup에서는 fdupes와 다르게 사용자가 입력한 조건(파일 확장자, 파일 크기 범위)에 맞춰 동일한(중복된) 정규 파일을 찾고 삭제하는 프로그램이다. 또한 md5와 sha1 탐색을 비교하여 두 해시함수의 탐색시간을 비교할 수도 있다. 또한 다양한 옵션을 통한 삭제를 지원하여 사용자가 편리하게 동일(중복)한 파일을 찾고 삭제할 수 있다.

사용자가 삭제해야 할 중복 파일의 개수가 많지 않다면 굳이 ssu_sdup 프로그램을 사용할 필요는 없다. 그러나 만약 중복 파일이 수백, 수천개가 된다면 시간 소요가 많을 것이다. 이때 사용자가 하나하나 찾아서 삭제하는 대신 ssu_sdup 프로그램을 사용하면 손쉽게 원하는 명령을 실행하여 파일과 디렉토리를 찾아내고 그 차이점을 확인할 수 있다. 본 과제는 Linux fdupes와 달리 입력 조건(파일 확장자, 파일 크기 범위)에 맞춰 정규 파일을 찾고 입력한 옵션에 맞춰 파일을 삭제함으로써 시스템 내 존재하는 동일한 파일을 찾고 삭제하는 과정을 이해하고 구현할 수 있다.

2. 구현 플랫폼

- OS : Linux Ubuntu 버전 18.04
- 컴파일러 : gcc 7.5
- 커널 버전 : Linux ip-172-31-13-105 5.4.0-1071-aws #76~18.04.1-Ubuntu SMP Mon Mar 28 17:49:57 UTC 2022 x86_64 x86_64 x86_64 GNU/Linux

3. 상세 설계

3-1) 목표 설정

본 과제의 목표 구현 프로그램인 ssu_sdup는 linux의 fdupes를 명세에 맞게 직접 구현해 보는 것이다. 먼저 fmd5 또는 fsha1이 아닌 다른 명령어인 exit와 help의 구현이 필요하다. fmd5, fsha1에서는 우선적으로 사용자가 입력한 조건에 속하는 파일의 재귀적 탐색이 필요하다. 일치하는 파일들을 저장해 둔 뒤, 해시 값이 일치하는 파일을 찾고 링크드 리스트에 넣어주어야 한다. 그 후 파일을 크기순으로 정렬하여 출력해주어야 한다. 사용자가 입력한 조건들에 대한 처리 방법은 아래와 같다.

1. [FILE_EXTENSION] : 탐색할 파일의 확장자

- "*" 입력 시, 모든 정규 파일을 대상으로 중복 파일 탐색

- "*.(확장자)" 입력 시, (확장자)인 정규 파일에서만 중복 파일 탐색 (예. *.jpg : jpg 확장자를 가진 파일에서만 중복 파일 탐색)

- [FILE_EXTENSION] 인자에 입력이 없거나, "*", "*.(확장자)" 외 다른 입력이 들어오면 에러 처리 후 프롬프트 출력

2. [MINSIZE] : 탐색할 파일의 최소 크기

- [MINSIZE] 인자에 바이트, KB, MB, GB 단위가 가능하며, 단위 미입력 시 기본 설정은 바이트 단위임 (단위는 소문자도 가능해야 하며, 공백으로 구분하지 않음)

- [MINSIZE] 인자에 KB, MB, GB 단위는 실수도 가능해야 함 (실수의 소숫점 이하 자리는 KB는 3자리 이상, MB는 6자리 이상, GB는 9자리 이상이 표시되면 됨. 그 이상의 자리는 의미 없음)

- [MINSIZE] 인자에 '~' 입력할 경우, [MAXSIZE] 이하인 모든 파일에서 중복 파일 탐색

- [MINSIZE] 인자에 입력이 없거나, 숫자(실수 포함)나 '~' 외 다른 입력이 들어오면 에러 처리 후 프롬프트 출력

3. [MAXSIZE] : 탐색할 파일의 최대 크기

- [MINSIZE] 인자에 바이트, KB, MB, GB 단위가 가능하며, 단위 미입력 시 기본 설정은 바이트 단위임 (단위는 소문자도 가능해야 하며, 공백으로 구분하지 않음)

- [MINSIZE] 인자에 KB, MB, GB 단위는 실수도 가능해야 함 (실수의 소숫점 이하 자리는 KB는 3자리 이상, MB는 6자리 이상, GB는 9자리 이상이 표시되면 됨. 그 이상의 자리는 의미 없음)

- [MAXSIZE] 인자에 '~'를 입력할 경우, [MINSIZE] 이상인 모든 파일에서 중복 파일 탐색

- [MAXSIZE] 인자에 입력이 없거나, 숫자(실수 포함)나 '~' 외 다른 입력이 들어오면 에러 처리 후 프롬프트 출력

- [MINSIZE]와 [MAXSIZE] 모두 '~'이면 크기 제한 없이 중복 파일 탐색

4. [TARGET_DIRECTORY] : 탐색할 디렉토리 경로

- 절대경로와 "~(홈 디렉토리)"를 포함한 상대경로 모두 가능해야 함

- 인자에 루트("/")를 입력할 경우, 파일 탐색 시 권한 문제가 발생할 수 있으므로 root 권한으로 실행해야 함

- 루트 디렉토리부터 탐색 시, "proc"과 "run", "sys" 디렉토리는 제외하여 탐색

- [TARGET_DIRECTORY] 인자에 입력이 없거나, 디렉토리가 아니거나, 존재하지 않는 디렉토리인 경우, 에러 처리 후 프롬프트 출력

중복 파일의 출력 방법은 다음과 같다.

- 지정한 디렉토리 내에 중복 파일이 있는 경우, 각 중복 파일 리스트를 구분하여 한 세트씩 중복 파일 리스트 출력

- 첫째 줄에는 각 중복 파일 리스트의 세트 번호와 파일 크기(바이트 단위), 해시값을 출력. 예시 7 참고
 - 파일 크기는 천 단위마다 ','로 구분해야 함
- 다음 줄부터, 중복 파일 리스트 내의 인덱스 번호와 파일의 절대 경로, 마지막 수정 시간, 마지막 접근 시간을 출력
 - 세트 내 리스트 파일들의 출력순서는 파일 절대 경로가 짧은 것부터, 동일한 절대 경로는 아스키 코드 순서
 - 마지막 수정 시간과 접근 시간은 stat 구조체 이용
 - 위 과정을 반복하여 전체 중복 파일 리스트 출력함
- 지정한 디렉토리 내에 중복 파일이 없는 경우, "No duplicates in (TARGET_DIRECTORY의 절대경로)" 출력 후 프롬프트 출력
- 중복 파일 탐색 및 출력을 마치면 마지막 줄에 탐색 소요 시간 출력.
- 중복 파일 리스트가 존재하는 경우 ">>"를 출력하고 사용자 입력 대기

이렇게 중복 파일을 출력해준 후 사용자로부터 삭제 옵션을 입력 받는다. 옵션의 내용은 다음과 같다.

d [LIST_IDX] : 선택한 세트에서 [LIST_IDX]에 해당하는 파일 삭제

- [LIST_IDX] 인자에 입력이 없거나 범위를 벗어난 경우, 에러 처리 후 ">>" 출력하고 사용자 입력 대기
- 삭제 후 삭제한 파일의 절대 경로 출력

i : 선택한 세트의 중복 파일 리스트에 있는 파일의 절대 경로를 하나씩 보여주면서 삭제 여부 확인 후 파일 삭제 또는 유지

- 중복 파일 리스트 원소의 절대경로를 하나씩 보여주면서 삭제 여부 확인 메시지 출력
- "y"나 "Y" 입력 시, 해당 파일 삭제, "n"나 "N" 입력 시, 해당 파일 유지
 - 이외의 입력이 들어오면 예외 처리 후 ">>" 출력하고 사용자 입력 대기
- 모든 파일에 대해 삭제 여부 확인 완료하면 전체 중복 파일 리스트 출력
 - 선택한 세트의 중복 파일 리스트에 있는 파일을 모두 삭제하거나 하나만 남는 경우 중복 파일 리스트에서 제거됨

f : 가장 최근에 수정한 파일을 남겨두고 나머지 중복 파일을 삭제

- 삭제 후 가장 최근에 수정한 파일의 절대 경로와 수정 시간 출력

- f 옵션 사용 시, 파일이 하나만 남기 때문에 중복 파일 리스트에서는 제거됨

t : 가장 최근에 수정한 파일을 남겨두고 나머지 중복 파일을 휴지통으로 이동

- 삭제 후 가장 최근에 수정한 파일의 절대 경로와 수정 시간 출력

- t 옵션 사용 시, 파일이 하나만 남기 때문에 중복 파일 리스트에서는 제거됨

a : 모든 세트의 입력한 인덱스에 해당하는 파일 삭제

- 입력한 인덱스가 없는 세트의 경우 패스

본 과제의 목표는 다음과 같은 조건을 만족하여 중복파일을 찾아 삭제할 수 있는 ssu_sdup를 만드는 것이다.

3-2) 분석

먼저 ssu_sdup가 실행되면 fgets를 통해 명령어를 입력받고 memmove와 strtok를 사용해 입력한 명령어를 공백 단위로 분리해준다. 이를 통해 fmd5, fsha1, help, exit 명령을 구분할 수 있다. exit의 경우 ssu_sdup를 종료시킨다. help를 입력한 경우 fork()와 execl()을 통해 자식 프로세스로 ssu_help를 실행한다. 부모 프로세스인 ssu_sdup는 wait()로 자식 프로세스가 종료될 때까지 대기한다. ssu_help에서는 명령어 사용법을 출력해준다.

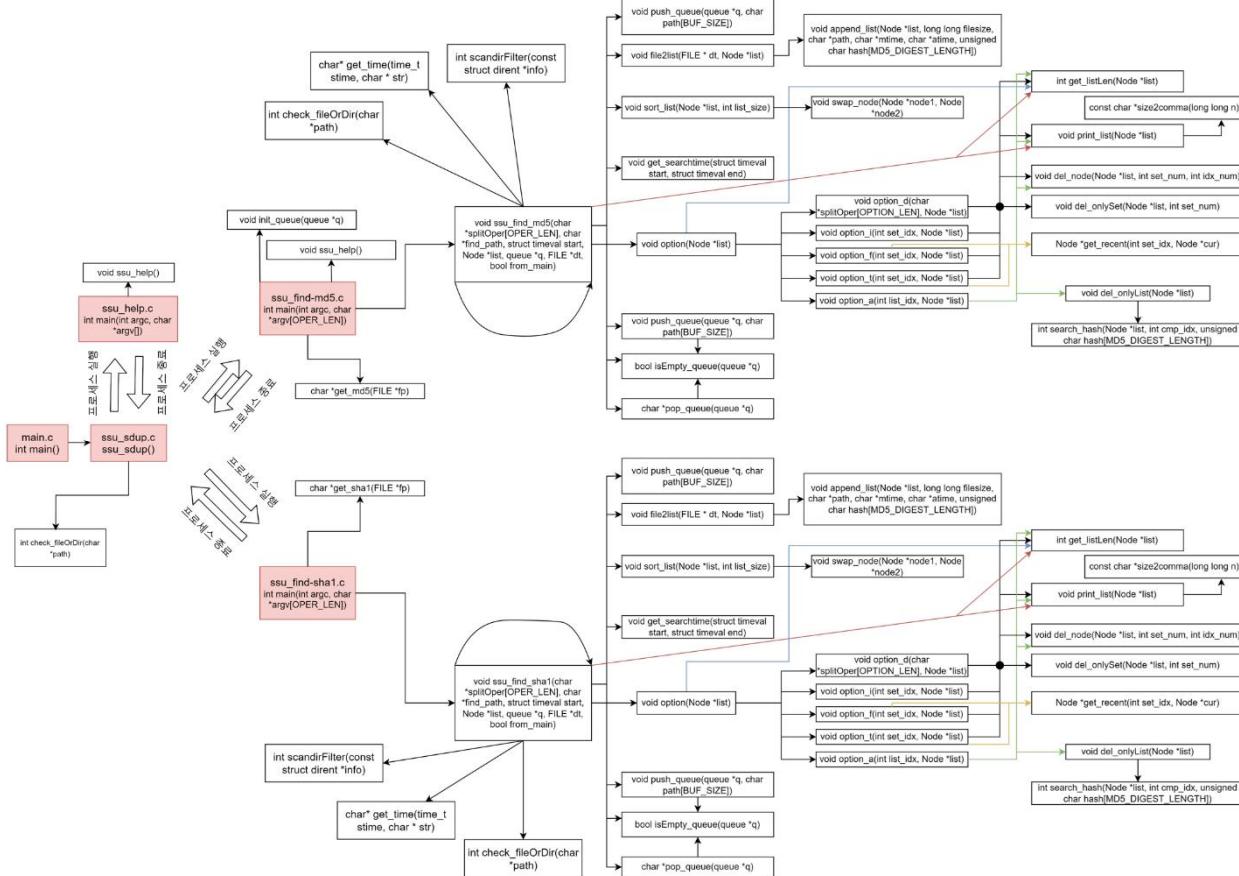
fmd5, fsha1 역시 마찬가지로 각각의 프로세스를 실행한다. (각각 함수의 자세한 내용은 구현방법에서 설명) fmd5에서는 execl()로부터 전달받은 입력 인자에 따라 사용자가 입력한 범위 내에 있는 파일들만 비교 대상으로 추가해주어야 한다. scandir() 함수를 사용하여 디렉토리 내 모든 파일을 조회한 뒤, for문으로 각각의 파일들을 검사한다. 파일은 정규파일인 경우와 디렉토리인 경우로 나누어준다. 정규파일인 경우 사용자가 입력한 조건(확장자, 최소 크기, 최대 크기)에 맞는지 확인 한 후, 맞지 않은 경우 패스한다. 맞는 경우 텍스트 파일에 그 파일에 대한 정보를 적어준다. (텍스트 파일은 일시적으로 생성해준다.) 디렉토리인 경우 하위 파일들을 bfs로 탐색해야 하므로 큐에 삽입해준다. for문이 끝난 후 큐가 빌 때 까지 해당 디렉토리를 다시 검사해주면 된다.

이 과정이 모두 끝나면 텍스트파일에는 사용자가 입력한 조건에 맞는 파일들이 모두 담아져 있을 것이다. 링크드 리스트로 중복 파일 리스트를 관리해야 하므로 텍스트 파일에서 중복 파일을 찾아 링크드 리스트에 추가해 준다. 이렇게 링크드 리스트에 중복 파일 추가 과정이 끝나면 텍스트파일을 삭제한다.

이제 링크드 리스트에는 중복파일들이 담겨져 있다. 그러나 파일 크기에 따라 정렬할 필요가 있다. 링크드 리스트 내 파일들을 크기를 기준으로 버블 소트를 실행하면 링크드 리스트는 파일 크기 순서에 따라 정렬되어 있다. 이렇게 정렬

된 링크드 리스트를 사용자가 입력한 옵션에 맞게 각각의 함수를 실행해 삭제해주면 된다. 사용자가 exit를 입력한다면 프로세스가 종료되고, 부모 프로세스(ssu_sdup)로 이동한다.

3-3) 흐름도



4. 구현 방법 설명

◆ int main() – main.c

메인함수이다. ssu_sdup()를 실행하고 ssu_sdup가 종료되면 Prompt End를 출력한 후 종료된다.

◆ void ssu_sdup()

ssu_sdup 프로그램을 담당하는 함수이다. fgets()를 통해 명령어를 입력 받고 memmove()와 strtok()를 사용해 입력한 명령어를 공백 단위로 분리해준다. 입력한 명령어에 따라 에러처리, fmd5, fsha1, help, exit의 경우를 분리한다. exit의 경우 ssu_sdup() 함수를 종료한다. fmd5(), fsha1(), help의 경우 fork()를 통해 부모 프로세스는 wait()를 통해 자식 프로세스가 종료될때까지 대기하고 execl()을 통해 각각의 프로세스를 실행시킨다. fmd5(), fsha1()에서는 추가적인 처리가 있는데, 올바

른 확장자를 입력했는지 먼저 검사해준다. 그 후 만약 TARGET_DIRECTORY에 ~(홈 디렉토리)를 입력할 경우 getpwuid()와 getuid()를 사용해 사용자의 홈 디렉토리를 알아낸다. 그 뒤 memmove(), strcpy(), sprint()를 사용해 홈디렉토리 경로로 문자열을 만들어준다. 이렇게 TARGET_DIRECTORY를 확정한 후 check_fileOrDir() 함수를 통해 입력한 경로가 디렉토리인지 체크한다. (디렉토리가 아닌 경우 예러 처리) 디렉토리인 경우 fork()와 execl()을 통해 각각의 프로세스를 실행한다.

- ◆ int main(int argc, char *argv[OPER_LEN]) – ssu_find-md5.c

ssu_find-md5.c의 메인함수이다. 우선 fmd5 명령을 실행하기 위해서는 중복 파일을 관리할 링크드 리스트, BFS를 관리할 큐, 사용자의 입력 조건에 맞는 정규 파일을 저장해 둘 텍스트 파일이 필요하다. 따라서 헤더 파일에 먼저 Qnode 구조체를 정의해 둔다. Qnode는 디렉토리의 경로인 path와 포인터 next를 정의하였다. 그 후 queue를 구조체로 정의한다. queue에는 처음을 나타내는 front, 마지막을 나타내는 rear를 Qnode 구조체로 정의한다. 그리고 큐 안의 노드 개수를 나타내는 cnt를 정의한다.

이렇게 정의한 큐를 선언해주고, 초기화를 해주는 함수인 init_queue()를 실행한다. 링크드 리스트 역시 선언해준다. 실행시간을 측정하기 위해 gettimeofday()를 실행하고 fmd5를 담당하는 함수인 ssu_find_md5() 함수를 실행한다. ssu_find_md5()가 끝나면 delete_list() 함수를 호출해 링크드 리스트를 제거한다.

- ◆ void ssu_find_md5(char *splitOper[OPER_LEN], char *find_path, struct timeval start, Node *list, queue *q, FILE *dt, bool from_main)

fmd5를 담당하는 함수이다. 우선 인자에 대해 알아보면, splitOper는 사용자가 입력한 명령어를 공백 단위로 분리해둔 문자열이다. find_path는 scandir로 조회할 디렉토리 경로이다. start는 시작 시간, list는 링크드 리스트. q는 큐, dt는 링크드 리스트에 넣어줄 파일들을 임시로 저장해둘 파일 포인터, 마지막으로 from_main은 main함수에서 실행됐는지 확인하는 변수이다.

먼저 디렉토리 내 파일 목록을 조회하기 위해 관련 변수를 선언해준 뒤, scandir를 실행한다. 그 후 조회한 파일만큼 for문을 돌려 각각의 파일을 검사한다. 첫 번째로 check_fileOrDir() 함수를 실행해 파일이 정규파일인지 디렉토리인지 체크한다.

정규파일인 경우, 처음으로 사용자가 입력한 조건의 확장자인지 검사한다. strrchr을 이용해 .뒤의 문자열을 읽어온다. (없을 경우 NULL) 그 후 strcmp로 사용자가 입력한 조건의 확장자와 일치하는지 확인하고 아닐 경우 패스한다. (이전의

strrchr 결과가 NULL인 경우도 패스한다.) 이제 lstat을 사용해 파일 정보를 얻어온다. 만약 파일의 읽기 권한이 없으면 패스한다. 이를 통해 파일의 크기를 구할 수 있다. 이때 파일의 크기가 0바이트인경우 패스한다.

다음으로 파일의 크기가 사용자가 입력한 조건 내인지 검사해 주어야 한다. 먼저 최소 크기 이상인지 확인한다. 사용자가 ~를 입력한 경우는 확인하지 않는다. 여기서 KB, MB, GB 단위의 경우 실수도 가능해야 한다. 따라서 최소 크기 를 비교할 변수를 long double로 선언하고 strtold 함수를 사용해 문자열을 long double형으로 변환해준다. 만약 수만 입력한 경우 modf로 실수부와 소수부로 분리해 올바른 입력을 했는지 검사한다. KB, MB, GB를 입력한 경우 byte단위로 변환해준다. 그 후 파일 크기가 최소 크기 이상인지 비교한다. 최대 크기 이하 검사도 동일한 프로세스로 진행한다.

이제 모든 조건을 통과했으니 텍스트 파일에 정규 파일 정보를 추가해 주어야 한다. (만약 파일의 읽기 권한이 없으면 패스한다.) fopen()으로 파일을 r모드로 열고 get_md5() 함수를 사용해 해시 값을 읽어온다. 작업이 끝나면 fclose()로 닫아준다. 시간 정보는 get_time()함수를 만들어 명세에 맞는 문자열 포맷으로 변환하였다. 이제 텍스트 파일에 fputs()로 써주는데, 저장 양식은 다음과 같다. (중복 체크 여부 -> 처음 써줄 때 *로 저장함|파일 크기|파일 절대경로|mtime|atime|해시값) 이렇게 정규 파일의 프로세스가 끝났다.

디렉토리인 경우, bfs로 진행하므로 큐에 디렉토리의 정보를 push해주어야 한다. 루트에서부터 탐색 시 proc, run, sys를 제외해야 하므로 strcmp()와 if문으로 처리하였다. 그 후 push_queue() 함수를 통해 파일 경로를 담은 노드를 큐에 추가해준다.

이렇게 조회한 파일을 모두 처리하고 for문이 끝나면 큐에 들어가 있는 디렉토리들을 처리해 주어야 한다. from_main 변수를 사용해 재귀호출한 경우는 함수를 끝내고 메인 함수의 경우만 다음 프로세스를 진행한다. 큐를 처리하기 위해 while()문의 조건을 !isEmpty_queue()로 둔다. (isEmpty_queue는 큐가 비었을 경우 true 리턴) 이제 큐가 빌 때까지 재귀적으로 탐색해주기 위해 ssu_find_md5()를 재귀호출한다. 이때 파일 경로를 나타내는 인자인 find_path에는 큐를 pop 해 경로를 가져오는 함수인 pop_queue() 함수를 사용한다. 이 과정이 모두 끝나면 지정한 디렉토리의 하위 디렉토리를 재귀적으로 탐색하여 사용자가 입력한 조건에 맞는 모든 정규파일들이 텍스트 파일에 한 줄씩 저장되어 있을 것이다.

이제 텍스트 파일 내 저장된 정규파일들 중 중복파일만 링크드 리스트에 추가해 주어야 한다. 이는 file2list() 함수를 호출하여 처리한다. 이렇게 링크드 리스트에 모든 중복 파일이 추가되면 일시적으로 생성했던 텍스트 파일은 unlink() 함수를 통해 삭제해준다. 그 후 gettimeofday()로 종료 시간을 측정한다.

만약 중복 파일이 없다면 "No duplicates in (TARGET_DIRECTORY의 절대경로)"를 출력해 주어야 한다. 따라서 get_listLen() 함수를 통해 링크드 리스트의 길이를 구하고, 길이가 0이면 해당 메시지를 출력하고 함수를 종료한다.

링크드 리스트에 중복 파일이 들어가 있으니 파일 크기 순서대로 정렬해주면 모든 과정이 완료된다. `sort_list()` 함수를 통해 파일 크기 순서대로 정렬해준다. 이 과정이 모두 끝나면 `print_list()` 함수를 통해 중복 파일 리스트를 출력해주고 `get_searchtime()` 함수를 통해 탐색 소요 시간을 출력해준다. 이제 사용자로부터 옵션을 입력 받고 처리하기 위해 `option()` 함수를 실행한다.

◆ `void file2list(FILE * dt, Node *list)`

중복 파일을 텍스트 파일로부터 링크드 리스트에 추가해주는 함수이다. 여기서 모든 정규 파일들이 적혀 있는 텍스트 파일로부터 중복 파일을 추출해야 한다. 이는 텍스트 파일에 저장했던 양식인 (중복 체크 여부|파일 크기|파일 절대경로|mtime|atime|해시값) 중 중복 여부를 통해 처리할 수 있다. `strtok()` 함수를 사용해 | 기준으로 문자열을 분리해 저장해줄 수 있었다. 처음 텍스트 파일에 데이터를 써줄 때 *로 써주었는데, 가장 위의 줄을 기준 라인으로 잡는다. 기준 라인부터 파일의 끝까지 반복하여 한 줄씩 `fgets()`로 읽는다.(비교 라인 탐색) 만약 기준 라인과 비교 라인의 해시 값이 일치할 경우(중복 파일인 경우) 해당 라인의 정보를 `append_list()` 함수를 통해 링크드 리스트에 추가해 준다. 그 후 *로 적어주었던 데이터를 **로 변경해준다. 이것이 중복 체크를 완료했다는 표시이다. 이렇게 표시를 해주면 다음 기준 라인을 잡을 때 중복 체크 부분의 데이터가 **이면, 이미 링크드 리스트에 추가를 해주었다는 의미이므로 패스하면 된다. `ftell()`과 `fseek()`를 사용하여 데이터를 읽고 쓸 위치를 자유롭게 변경할 수 있었다. 이 과정이 모두 종료되면(기준 라인이 파일의 끝에 도달하면) 모든 중복 파일들이 링크드 리스트에 추가되었으므로 함수를 종료한다.

◆ `int scandirFilter(const struct dirent *info)`

`scandir()`로 하위 파일을 가져올 때 .과 ..을 무시해주는 함수이다. `strcmp()`을 사용해 .이나 ..일 경우 0을 리턴 하고, 아닐 경우 1을 리턴해준다.

◆ `int check_fileOrDir(char*path)`

인자로 받은 path가 파일인지 디렉토리인지 판별해주는 함수이다. `stat`을 통해 path의 정보를 얻고 정규파일인 경우 1, 디렉토리일 경우 2를 리턴 해준다.

◆ `char *get_md5(FILE *fp)`

md5의 값을 가져오는 함수이다. MD5_Init(), MD5_Update, MD5_Final()을 통해 값을 가져온 후 sprint()를 사용해 문자열 형식으로 변환해 리턴 한다.

- ◆ `char* get_time(time_t stime, char * str)`

명세의 시간 정보 포맷에 맞게 변환해주는 함수이다. 인자로 받은 stime을 strftime()을 사용해 포맷에 맞게 변환해준뒤, str에 저장해 리턴한다.

- ◆ `void get_searchtime(struct timeval start, struct timeval end)`

탐색 소요 시간을 계산에 출력해주는 함수이다. gettimeofday()로 종료 시간을 받아온 후, 먼저 end.tv_sec -= start.tv_sec 를 통해 초 단위 부분을 계산해준다. 그 후 ms단위 부분을 계산해주는데, 이 때 결과가 마이너스인 경우 (start.tv_usec가 end.tv_usec보다 큰 경우)는 초 단위를 1초 빼주고 ms 단위에 1000000을 더해준다. 구한 탐색 소요 시간을 출력해 준 후 함수는 종료된다.

- ◆ `int get_listLen(Node *list)`

링크드 리스트의 크기를 구해주는 함수이다. cur 노드를 선언하고 cur != NULL 조건으로 while문을 돌려 카운트를 증가시키며 리스트를 순회한다. 반복문이 끝나면 카운트를 리턴 한다.

- ◆ `const char *size2comma(long long n)`

파일 크기를 천 단위마다 ','로 구분한 문자열로 변환해 주는 함수이다.

- ◆ `void append_list(Node *list, long long filesize, char *path, char *mtime, char *atime, unsigned char hash[MD5_DIGEST_LENGTH])`

링크드 리스트 끝에 정규 파일 관련 정보가 들어있는 노드를 추가해주는 함수이다. 입력인자로 리스트와 파일 정보들을 받는다. 만약 리스트가 비어 있는 경우 노드 크기만큼 malloc으로 할당해 새로운 노드를 만들어 준 뒤, 파일의 정보를 노드에 담아준다. 그 후 list->next를 새로운 노드로 연결해 주면 된다.

리스트가 비어 있지 않은 경우는 리스트의 마지막 노드까지 while문을 돌린다. (조건 : cur->next != NULL) 그 후

위와 같이 노드를 만들어 파일 정보를 담아준 뒤, cur->next를 새로운 노드로 연결해 준다.

◆ void sort_list(Node *list, int list_size)

링크드 리스트를 파일 크기 순서대로 정렬 해주는 함수이다. 정렬은 버블 정렬을 사용하였다. head의 다음으로 cur 노드의 시작 위치를 잡아준다. 버블 정렬 알고리즘으로 for문을 2중으로 돌려 두 파일의 파일 크기를 비교한다. 만약 앞 노드의 파일 크기가 뒤 노드의 파일 크기보다 크다면, swap_node() 함수를 통해 두 노드의 파일 정보를 바꾸어 준다.

◆ void swap_node(Node *node1, Node *node2)

인자로 받은 두개의 노드에 담겨있는 파일 정보를 서로 맞바꾸어 준다.

◆ void print_list(Node *list)

중복 파일 리스트를 출력해주고, 해당 노드가 몇 번째 중복 세트의 몇 번째 인덱스에 위치해 있는지 라벨링을 해주는 함수이다. head 다음 노드인 cur, 세트 카운터인 cnt, 세트 내 인덱스 카운터인 small_cnt를 선언해준다. 이전 노드의 해시 값을 따로 저장해두기 위해 미리 선언을 해준다. cur != NULL 조건으로 while문을 실행해 만약 현재 노드의 해시 값과 이전 노드의 해시 값이 다르면 cnt를 증가시키고 small_cnt를 1로 설정한다. 그 후 Identical files... 메시지를 출력해준다. (여기서 size2comma() 함수를 사용해 천 단위마다 ','로 구분해 출력한다.) 출력이 끝나면 이전 노드의 해시 값을 현재 노드의 해시 값으로 업데이트한다.

이전, 현재 노드의 해시 값이 같거나 위의 과정이 끝나면 노드의 set_num, idx_num에 각각 세트 번호, 세트내 인덱스 번호를 넣어준다. 그리고 중복 파일을 출력해주어야 하므로 파일 정보를 출력해준다. small_cnt의 카운트를 1 증가시키고 다음 노드로 넘어간다. 이렇게 링크드 리스트가 끝날 때 까지 반복한다.

◆ void init_queue(queue *q)

큐를 초기화 해주는 함수이다. 큐의 처음을 나타내는 front, 마지막을 나타내는 rear를 NULL로 설정하고, 큐의 개수를 나타내는 cnt를 0으로 설정한다.

◆ bool isEmpty_queue(queue *q)

큐가 비어있으면(q->cnt == 0) true를 리턴한다.

- ◆ void push_queue(queue *q, char path[BUF_SIZE])

큐에 데이터를 push해주는 함수이다. malloc으로 Qnode struct의 노드를 하나 만들어주고 인자의 path를 노드에 넣어준다. 큐의 마지막에 push하므로 노드의 next는 NULL로 설정한다.

만약 큐가 비어있다면 q->front에 노드를 설정하고, 비어있지 않다면 q->rear의 다음을 노드로 설정한다. 큐의 카운트 증가, rear를 노드로 설정해주고 함수를 종료한다.

- ◆ char *pop_queue(queue *q)

큐를 pop해 해당 노드의 디렉토리 경로를 리턴 해주는 함수이다. 큐가 비어 있다면 에러 처리를 해준다. 노드(ptr)를 하나 만들어 front로 설정해준다. ptr의 경로를 추출한 뒤 삭제를 해야 하므로 ptr의 다음 노드를 front로 설정하고 ptr은 free해준다. 큐의 카운트를 1 감소시키고 추출한 경로를 리턴한다.

- ◆ void option(Node *list)

사용자가 입력한 옵션 부분을 담당하는 함수이다. fgets()로 사용자로부터 명령어를 입력받고 strtok(), memmove()를 사용하여 공백 단위로 문자열을 분리해 저장해준다. 입력이 없거나 입력 개수가 초과된 경우 등 상황에 따른 에러처리를 해준다. 에러 처리가 끝나면 각각의 옵션 별 함수들을 실행해준다.

- ◆ void option_d(char *splitOper[OPTION_LEN], Node *list)

d옵션을 처리하는 함수이다. 먼저 세트 인덱스와 리스트 인덱스는 이전에 노드에 저장해 뒀다. 따라서 같은 세트 인덱스가 나올 때까지 다음 노드로 이동한다. 같은 세트 인덱스가 나오면 다시 같은 리스트 인덱스가 나올 때까지 다음 노드로 이동한다. 이 과정을 통해 삭제할 노드로 이동할 수 있다.

이제 unlink로 파일을 삭제해주고 del_node() 함수를 통해 링크드 리스트에서 해당 노드를 삭제한다. 다음으로 del_onlySet() 함수를 사용해 해당 세트의 인덱스가 하나만 남았을 경우, 링크드 리스트에서 제거해준다. print_list()를 통해 중복 파일 리스트를 출력해주고 함수를 종료한다.

◆ void option_i(int set_idx, Node *list)

i옵션을 처리하는 함수이다. 선택한 세트의 리스트에 있는 파일을 하나씩 보여주어야 하므로 우선 같은 세트 인덱스가 나올 때까지 다음 노드로 이동한다. (d옵션과 동일) d옵션과 다른 점은 나중에 삭제 후 cur의 위치를 복구시켜 줄 노드가 필요하므로 pre노드를 선언해준다. (pre노드는 cur의 이전)

이제 while(cur->set_num == set_idx)를 통해 세트 내의 인덱스들을 순회시킨다. 중복 파일 리스트 원소의 절대경로를 하나씩 보여주면서 삭제 여부 확인 메시지 출력하고 fgets()로 사용자의 입력을 대기한다. 사용자가 N, n을 입력했다면 pre에 cur을 넣어주고 cur을 다음으로 이동한다. Y, y를 입력했다면 unlink()와 del_node()를 통해 파일을 삭제시키고 해당 노드를 링크드 리스트에서 제거한다. cur의 위치를 복구하기 위해 cur = pre->next로 지정해준다. 이렇게 선택한 세트 내 중복 파일들을 하나씩 보여주며 파일을 삭제 시킬 수 있었다. d옵션과 마찬가지로 del_OnlySet(), print_list()를 통해 세트의 인덱스가 하나만 남은 경우를 처리하고 중복 파일을 출력해준다.

◆ void option_f(int set_idx, Node *list)

f옵션을 처리하는 함수이다. 이전과 동일하게 입력한 세트로 이동한다. 가장 최근에 수정한 파일을 구하기 위해 recent 노드를 만들어주고, get_recent() 함수를 통해 가장 최근에 수정한 파일의 노드를 구해 recent 노드에 설정해준다. 이제 세트 내에서 while문으로 하나하나 탐색하며 현재 노드(cur)가 가장 최근 수정한 파일의 노드(recent)가 아니라면 unlink()와 del_node()를 사용해 제거해준다.(i옵션과 삭제 프로세스 동일) 만약 cur == recent의 경우 패스한다. (pre = cur, cur = cur->next)

이렇게 삭제 과정이 모두 끝나면 Left file 메시지를 출력해주고 del_OnlySet(), print_list()를 통해 세트의 인덱스가 하나만 남은 경우를 처리하고 중복 파일을 출력해준다.

◆ void option_t(int set_idx, Node *list)

t옵션을 처리하는 함수이다. f옵션에서 삭제 대신 휴지통으로 이동하는 점 빼고는 모두 동일하다. 이 프로그램에서는 현재 디렉토리에 trash라는 휴지통 디렉토리를 만들어주고 휴지통 디렉토리로 이동해주는 프로세스로 진행하였다.

먼저 mkdir로 휴지통 디렉토리를 만들어준다. (errno!=EEXIST를 조건에 추가해 이미 trash 디렉토리가 존재한 경우는 에러를 발생하지 않도록 처리하였다.) f옵션과 동일하게 진행하다가 삭제하는 부분 대신 link를 사용해 trash 내로 파일을 복사하였다. 여기서 추가적으로 처리해준 경우가 있는데, 이미 휴지통 내에 이동할 파일과 같은 이름의 파일이 있는 경

우이다. `strrchr()`과 `sprintf`를 사용해서 정규 파일 이름 앞에 `cp[번호 인덱스]`로 바꿔 복사한다. 만약 `cp[번호 인덱스]`도 존재할 경우, `cp[번호 인덱스+1]`로 바꿔 복사한다. (즉, 유일한 이름이 존재할 때 까지 `while`문 반복)

이렇게 휴지통 디렉토리로 복사가 완료되면 기존의 파일은 `unlink()`로 제거해주고, `del_node()` 함수를 통해 링크드 리스트에서도 제거한다. 이후 부분은 `f옵션과 동일하다.`

- ◆ `void option_a(int list_idx, Node *list)`

추가기능으로 구현한 함수이며 `a`옵션을 처리하는 함수이다. 사용자가 `[LIST_IDX] a`를 입력하면 모든 세트의 `LIST_IDX`를 삭제한다. (`LIST_IDX`가 존재하지 않는 세트는 패스) 먼저 현재 노드를 나타내는 `cur`은 `list->next`로, 이전 노드를 나타내는 `pre`는 `list`로 선언해준다. `cur!=NULL`까지 `while`문을 돌려 반복하면서 입력인자로 받은 `list_idx`와 노드의 `idx_num`이 같으면 `unlink()`와 `del_node()`를 사용해 파일과 노드를 삭제해준다. 같지 않을 경우 `pre = cur`로 이전 노드를 현재 노드로 맞춰주고, `cur = cur->next`로 현재 노드를 다음으로 이동한다. 모든 과정이 끝나면 `del_onlyList()`를 사용해 모든 링크드 리스트에서 하나만 남은 노드를 제거해준다. 그 후 `print_list()`를 사용해 링크드 리스트를 출력하고 함수를 종료한다.

- ◆ `void delete_list(Node *list)`

모든 링크드 리스트 노드를 제거하는 함수이다. 현재 노드를 나타내는 `cur`을 `list`로 설정해주고, 다음을 나타내는 노드 `next`를 선언한다. `while(cur != NULL)`과 `next`를 이용해 링크드 리스트 끝까지 반복하며 하나씩 `cur`을 `free`해준다. 모든 노드가 `free`되면 함수가 종료된다.

- ◆ `void del_node(Node *list, int set_num, int idx_num)`

리스트의 `set_num` 세트의 `idx_num` 인덱스에 위치한 노드를 제거해주는 함수이다. 현재 노드(`cur`)을 `list->next`로 설정해준다. 만약 빈 리스트인 경우 함수를 종료한다. `while(cur != NULL)`과 `pre`노드를 활용해 리스트 전체를 반복하고 (`cur->set_num == set_num && cur->idx_num == idx_num`로 세트 넘버와 인덱스 넘버가 같은 노드를 찾는다. 노드에 도달하면 이전 노드(`pre`)의 `next`를 삭제할 노드의 다음 노드(`cur->next`)로 설정해주고 `cur`노드를 `free`해준다. (만약 제거할 노드가 링크드 리스트의 마지막 노드였다면 `pre, cur`의 `next`를 `NULL`로 설정) 모든 과정이 끝나면 함수를 종료한다.

- ◆ `void del_onlySet(Node *list, int set_num)`

d, i, f, t옵션에서 삭제를 마치고 해당 세트에 노드가 하나만 남았을 경우를 처리하기 위해 호출하는 함수이다. 먼저 입력인자인 set_num과 일치하는 세트가 나올 때까지 while문을 돌린다. while문 내에서 cur->next가 NULL이라면 세트가 완전히 삭제된 경우이므로 함수를 종료한다. while문이 종료된 후 cur->next가 NULL이라면 링크드 리스트의 마지막 파일이 하나 남아 있는 경우이므로 free해준다.

위 사항에 해당하지 않으면 set_num과 같은 첫번째 노드가 cur로 되어있을 것이다. 만약 세트에 노드가 하나만 남아있지 않다면 cur의 다음 노드의 해시 값도 cur의 해시 값과 같을 것이다. 따라서 해시 값을 비교해 같지 않다면 노드가 하나만 남은 경우이므로 free해준다. 모든 과정이 끝나면 함수를 종료한다.

- ◆ void del_onlyList(Node *list)

a옵션(추가 기능)에서 삭제를 마치고 하나만 남은 노드를 처리하기 위해 호출하는 함수이다. 모든 노드를 while문으로 반복하며 search_hash() 함수를 사용하여 본인을 제외한 같은 노드가 있는지 탐색한다. 결과에 따라 노드를 free해주거나 넘어간다. (free해주는 프로세스는 del_onlySet()와 같다.) 모든 과정이 끝나면 함수를 종료한다.

- ◆ int search_hash(Node *list, int cmp_idx, unsigned char hash[MD5_DIGEST_LENGTH])

본인 노드를 제외한 해시 값이 일치하는 노드를 찾으면 해당 인덱스를 반환하고, 못 찾은 경우 -1을 리턴 하는 함수이다. 모든 링크드 리스트를 while문으로 반복해가며 인자로 받은 cmp_idx(본인 노드 인덱스)가 아니고 노드의 해시 값과 인자로 받은 해시 값이 일치하면 해당 인덱스를 리턴 한다.

- ◆ Node *get_recent(int set_idx, Node *cur)

가장 최근 수정한 파일 정보를 담은 노드를 반환하는 함수이다. 먼저 가장 최근 수정할 파일 정보를 담을 노드를 저장해둘 recent와 최근 수정 시간을 저장해둘 recent_time을 선언한다. 그 후 인자로 받은 cur노드부터 같은 세트까지 반복하며 다음 프로세스를 진행한다. lstat으로 파일의 정보를 얻고 recent_time과 현재 노드의 mtime을 비교한다. 만약 현재 노드의 mtime이 더 최근이면 recent_time과 recent를 갱신해준다. 이 과정을 반복해 가장 최근 수정한 파일 정보를 담은 노드를 구할 수 있다. 과정이 끝나면 구한 노드를 리턴 한다.

- ◆ int main(int argc, char *argv[OPER_LEN]) – ssu_find-sha1.c

ssu_find-sha1.c의 메인 함수이다. ssu_find-md5.c와 동일한 프로세스로 진행된다.

- ◆ void ssu_find_sha1(char *splitOper[OPER_LEN], char *find_path, struct timeval start, Node *list, queue *q, FILE *dt, bool from_main)

ssu_find-sha1.c 내에 있으며 ssu_find_md5와 기능은 동일하다.

- ◆ char *get_sha1(FILE *fp)

sha1 값을 조회하는 함수이다. get_md5와 다른점은 SHA_CTX로 변수를 선언하고 SHA1_Init(), SHA1_Update(), SHA1_Final()을 통해 해시 값을 구한다.

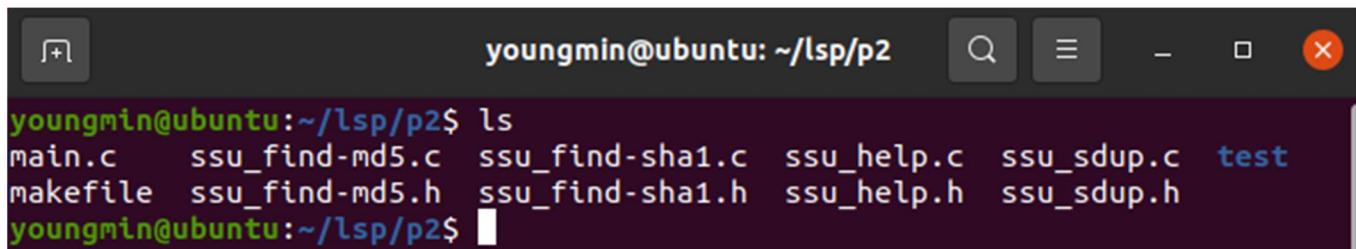
- ◆ int main(int argc, char *argv[]) – ssu_help.c

ssu_help.c의 메인 함수이다. ssu_help()를 실행한다. 함수가 끝나면 종료한다.

- ◆ void ssu_help()

명령어 사용법을 출력해주는 함수이다.

5. 실행 결과



```
youngmin@ubuntu:~/lsp/p2$ ls
main.c      ssu_find-md5.c  ssu_find-sha1.c  ssu_help.c  ssu_sdup.c  test
makefile    ssu_find-md5.h  ssu_find-sha1.h  ssu_help.h  ssu_sdup.h
youngmin@ubuntu:~/lsp/p2$
```

- 현재 디렉토리

4-1. make

```
youngmin@ubuntu:~/lsp/p2$ ls
main.c      ssu_find-md5.c  ssu_find-sha1.c  ssu_help.c  ssu_sdup.c  test
makefile    ssu_find-md5.h  ssu_find-sha1.h  ssu_help.h  ssu_sdup.h
youngmin@ubuntu:~/lsp/p2$ make
gcc -c main.c
gcc -c ssu_sdup.c
gcc main.o ssu_sdup.o -o ssu_sdup
gcc -c ssu_find-md5.c
gcc -Wall ssu_find-md5.o -o ssu_find-md5 -lcrypto -lssl
gcc -c ssu_find-sha1.c
gcc -Wall ssu_find-sha1.o -o ssu_find-sha1 -lcrypto -lssl
gcc -c ssu_help.c
gcc ssu_help.o -o ssu_help
youngmin@ubuntu:~/lsp/p2$ ls
main.c      ssu_find-md5.c  ssu_find-sha1.c  ssu_help.c  ssu_sdup.c
main.o      ssu_find-md5.h  ssu_find-sha1.h  ssu_help.h  ssu_sdup.h
makefile    ssu_find-md5.o  ssu_find-sha1.o  ssu_help.o  ssu_sdup.o
ssu_find-md5  ssu_find-sha1  ssu_help        ssu_sdup      test
youngmin@ubuntu:~/lsp/p2$
```

- make 정상적으로 실행
- ssu_sdup, ssu_find-md5, ssu_find-sha1, ssu_help 각각 따로 실행 파일 생성

4-2. 프롬프트

```
youngmin@ubuntu:~/lsp/p2$ ./ssu_sdup
20182615> █
```

```
youngmin@ubuntu:~/lsp/p2$ ./ssu_sdup
20182615>
20182615> █
```

- 엔터키만 입력시 프롬프트 재출력

4-3. help

```
youngmin@ubuntu:~/lsp/p2$ ./ssu_sdup
20182615>
20182615> 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
  [IDX] a : delete the [IDX] file for all sets
> help
> exit

20182615> 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
  [IDX] a : delete the [IDX] file for all sets
> help
> exit

20182615> test
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
  [IDX] a : delete the [IDX] file for all sets
> help
> exit

20182615>
```

- 몇 칸 띄우고 입력해도 정상 출력
- 이외 명령어 실행 시 자동으로 help를 실행시킨 것과 동일한 결과 출력
- 추가 옵션(a옵션) help에도 설명

4-4. exit

```
20182615> exit
Prompt End
youngmin@ubuntu:~/lsp/p2$ ./ssu_sdup
20182615> exit
Prompt End
youngmin@ubuntu:~/lsp/p2$
```

- 몇 칸 띄우고 입력해도 정상적으로 종료

4-5. fmd5

```
youngmin@ubuntu:~/lsp/p2$ ./ssu_sdup
20182615> fmd5
명령어를 맞게 입력해주세요
20182615> fmd5 * ~ ~
명령어를 맞게 입력해주세요
20182615> fmd5 test ~ ~ /
올바른 확장자 입력이 아님
20182615> fmd5 test x ~ /
올바른 확장자 입력이 아님
20182615> fmd5 test ~ X /
올바른 확장자 입력이 아님
20182615> fmd5 test ~ ~ iii
올바른 확장자 입력이 아님
20182615> fmd5 test ~ ~ ~ ~ ~
명령어를 맞게 입력해주세요
20182615> █
```

- 입력 잘못된 경우 에러 처리

```
20182615> fmd5 *.png ~ ~ .
No duplicates in /home/youngmin/lsp/p2
20182615> fmd5 *.txt ~ ~ .
---- Identical files #1 (13 bytes - 2d31d97199fb287d6fcb4f82ebd1b3f2) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/6.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[3] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

---- Identical files #2 (20 bytes - be497c2168e374f414a351c49379c01a) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[3] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[4] /home/youngmin/lsp/p2/test/b/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

---- Identical files #3 (513 bytes - a1c2c741136a7bf3149ff40775d23d4f) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

Searching time: 0:000325(sec:usec)
>> █
```

- 확장자 검색
- 중복 파일 없을 경우 메시지 출력
- 탐색 마치면 >> 출력 후 사용자 입력 대기

```

20182615> fmd5 * ~ ~ ~
--- Identical files #1 (1 bytes - 68b329da9893e34099c7d8ad5cb9c940) ----
[1] /home/youngmin/.config/pulse/838b1e08dd2d94289a3f8accdab8b0df03-default-sink (mtime : 2022-04-17 04:41:47) (atime : 2022-04-17 04:41:47)
[2] /home/youngmin/.config/pulse/838b1e08dd2d94289a3f8accdab8b0df03-default-source (mtime : 2022-04-17 04:41:47) (atime : 2022-04-17 04:41:47)

--- Identical files #2 (5 bytes - 71095c56cc41f2ca4f189b9dfcd7a38) ----
[1] /home/youngmin/.config/user-dirs.locale (mtime : 2022-04-17 03:51:00) (atime : 2022-04-17 03:51:00)
[2] /home/youngmin/snap/snap-store/558/.config/user-dirs.locale (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)

--- Identical files #3 (5 bytes - 1e0438b6c1e0c17304345e249cb74abe) ----
[1] /home/youngmin/lsp/p1/diri1/diri3/cc.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)
[2] /home/youngmin/lsp/p1/diri2/diri1/diri3/dd.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)

--- Identical files #4 (13 bytes - 2d31d97199fb287df6fc4f82ebdb1b3f2) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/6.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[3] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

--- Identical files #5 (17 bytes - 2046ce9668ec32e4b9f7bddb3c2ae60a) ----
[1] /home/youngmin/lsp/p1/diri/e.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)
[2] /home/youngmin/lsp/p1/dirz/e.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)

--- Identical files #6 (5 bytes - be497c2168e374f414a351c49379cb01a) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[3] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[4] /home/youngmin/lsp/p2/test/b/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

--- Identical files #7 (200 bytes - eddad2301cde055e29b0d3b8d2f4b724) ----
[1] /home/youngmin/.cache/fontconfig/CACHEDIR.TAG (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)
[2] /home/youngmin/snap/snap-store/common/.cache/fontconfig/CACHEDIR.TAG (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)

--- Identical files #8 (236 bytes - dd0458514c9a922b45da6a8bebbe47320) ----
[1] /home/youngmin/.cache/mozilla/firefox/mprt4603.default-release/safefrowsing/allow-flashallow-digest256.sbstore (mtime : 2022-04-17 03:53:49) (atime : 2022-04-17 03:53:49)
[2] /home/youngmin/.cache/mozilla/firefox/mprt4603.default-release/safefrowsing/except-flashallow-digest256.sbstore (mtime : 2022-04-17 03:53:49) (atime : 2022-04-17 03:53:49)

--- Identical files #9 (401 bytes - 236260b598cad10657e7250ce2379ede) ----
[1] /home/youngmin/lsp/p1/diri/g.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)
[2] /home/youngmin/lsp/p1/dirz/g.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)

--- Identical files #10 (476 bytes - 3354a777ff2e9799993092b920d76338) ----
[1] /home/youngmin/lsp/p1/a.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)
[2] /home/youngmin/lsp/p1/test.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)
[3] /home/youngmin/lsp/p1/diri/a.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)
[4] /home/youngmin/lsp/p1/dirz/diri/a.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)

--- Identical files #11 (476 bytes - e3e4c8c9747350472b075510657d96e1) ----
[1] /home/youngmin/lsp/p1/diri/b.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)
[2] /home/youngmin/lsp/p1/dirz/diri/b.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)

--- Identical files #12 (513 bytes - a1c2c741136a7bf3149ff40775d3d4f) ----
[1] /home/youngmin/lsp/p2/test/4/test_copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

```

- 모든 정규 파일 대상 탐색
- 홈 디렉토리(~) 탐색

```

youngmin@ubuntu:~/lsp/p2$ sudo su
[sudo] password for youngmin:
root@ubuntu:/home/youngmin/lsp/p2# ./ssu_sdup
20182615> fmd5 * ~ ~ ~
No duplicates in /root
20182615>

```

- 다른 계정 로그인 시, 그 계정의 홈 디렉토리(~)로 적용

```

20182615> fmd5 * 4kb ~ ~
--- Identical files #1 (48,814 bytes - 3d2c7d0c693332fac563bfacac565d94) ----
[1] /home/youngmin/.cache/mozilla/firefox/mprt4603.default-release/thumbnails/008ec4453ff31513f43893cba7aa31c8.png (mtime : 2022-04-17 04:42:12) (atime : 2022-04-17 04:42:12)
[2] /home/youngmin/.cache/mozilla/firefox/mprt4603.default-release/thumbnails/b37c4cfbdeae68780dea877a19bcd5b9.png (mtime : 2022-04-17 04:42:12) (atime : 2022-04-17 04:42:12)

Searching time: 0:029550(sec:usec)
>>

```

- 최소 크기 조건 탐색

```
20182615> fmd5 * 1.2KB ~
---- Identical files #1 (48,814 bytes - 3d2c7d0c693332fac563bfacac565d94) ----
[1] /home/youngmin/.cache/mozilla/firefox/mprt4603.default-release-thumbnails/008ec4453ff31513f43893cba7aa31c8.png (mtime : 2022-04-17 04:42:12) (atime : 2022-04-17 04:42:12)
[2] /home/youngmin/.cache/mozilla/firefox/mprt4603.default-release-thumbnails/b37c4cfbdeae68780dea877a19bcd5b9.png (mtime : 2022-04-17 04:42:12) (atime : 2022-04-17 04:42:12)

Searching time: 0:032645(sec:usec)
```

```
>> █
```

- KB, MB, GB의 경우 소수점 지원

```
20182615> fmd5 * 0 5 ~
---- Identical files #1 (1 bytes - 68b329da9893e34099c7d8ad5cb9c940) ----
[1] /home/youngmin/.config/pulse/838b1e08d2d94289a3f8acdda8b6df63-default-sink (mtime : 2022-04-17 04:41:47) (atime : 2022-04-17 04:41:47)
[2] /home/youngmin/.config/pulse/838b1e08d2d94289a3f8acdda8b6df63-default-source (mtime : 2022-04-17 04:41:47) (atime : 2022-04-17 04:41:47)

---- Identical files #2 (5 bytes - 71095c56c641f2c4a4f189b9dfcd7a38) ----
[1] /home/youngmin/.config/user-dirs.locale (mtime : 2022-04-17 03:51:00) (atime : 2022-04-17 03:51:00)
[2] /home/youngmin/snap/snap-store/558/.config/user-dirs.locale (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)

---- Identical files #3 (5 bytes - 1e0438b6c1e0c17304345e249cb74abe) ----
[1] /home/youngmin/lsp/p1/dir1/dir3/cc.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)
[2] /home/youngmin/lsp/p1/dir2/dir1/dir3/dd.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)

Searching time: 0:003484(sec:usec)
```

```
>> █
```

- 최대 크기 조건 탐색

```

youngmin@ubuntu: ~/lsp/p2
[2] ./snap/gnome-3-38-2004/99/usr/lib/x86_64-linux-gnu/dri/iris_dri.so (mtime : 2021-11-22 00:23:04) (atime : 2021-11-22 00:23:04)
[3] ./snap/gnome-3-38-2004/99/usr/lib/x86_64-linux-gnu/dri/kms_swrast_dri.so (mtime : 2021-11-22 00:23:04) (atime : 2021-11-22 00:23:04)
[4] ./snap/gnome-3-38-2004/99/usr/lib/x86_64-linux-gnu/dri/nouveau_dri.so (mtime : 2021-11-22 00:23:04) (atime : 2021-11-22 00:23:04)
[5] ./snap/gnome-3-38-2004/99/usr/lib/x86_64-linux-gnu/dri/r300_dri.so (mtime : 2021-11-22 00:23:04) (atime : 2021-11-22 00:23:04)
[6] ./snap/gnome-3-38-2004/99/usr/lib/x86_64-linux-gnu/dri/r600_dri.so (mtime : 2021-11-22 00:23:04) (atime : 2021-11-22 00:23:04)
[7] ./snap/gnome-3-38-2004/99/usr/lib/x86_64-linux-gnu/dri/radeonsi_dri.so (mtime : 2021-11-22 00:23:04) (atime : 2021-11-22 00:23:04)
[8] ./snap/gnome-3-38-2004/99/usr/lib/x86_64-linux-gnu/dri/swrast_dri.so (mtime : 2021-11-22 00:23:04) (atime : 2021-11-22 00:23:04)
[9] ./snap/gnome-3-38-2004/99/usr/lib/x86_64-linux-gnu/dri/virtio_gpu_dri.so (mtime : 2021-11-22 00:23:04) (atime : 2021-11-22 00:23:04)
[10] ./snap/gnome-3-38-2004/99/usr/lib/x86_64-linux-gnu/dri/vmwgfx_dri.so (mtime : 2021-11-22 00:23:04) (atime : 2021-11-22 00:23:04)
[11] ./snap/gnome-3-38-2004/99/usr/lib/x86_64-linux-gnu/dri/zink_dri.so (mtime : 2021-11-22 00:23:04) (atime : 2021-11-22 00:23:04)

---- Identical files #42737 (26,655,984 bytes - bedb295211cf957aa64751c2d0c88410) ----
[1] ./usr/lib/x86_64-linux-gnu/dri/d3d12_dri.so (mtime : 2022-04-08 07:46:47) (atime : 2022-04-08 07:46:47)
[2] ./usr/lib/x86_64-linux-gnu/dri/iris_dri.so (mtime : 2022-04-08 07:46:47) (atime : 2022-04-08 07:46:47)
[3] ./usr/lib/x86_64-linux-gnu/dri/kms_swrast_dri.so (mtime : 2022-04-08 07:46:47) (atime : 2022-04-08 07:46:47)
[4] ./usr/lib/x86_64-linux-gnu/dri/nouveau_dri.so (mtime : 2022-04-08 07:46:47) (atime : 2022-04-08 07:46:47)
[5] ./usr/lib/x86_64-linux-gnu/dri/r300_dri.so (mtime : 2022-04-08 07:46:47) (atime : 2022-04-08 07:46:47)
[6] ./usr/lib/x86_64-linux-gnu/dri/r600_dri.so (mtime : 2022-04-08 07:46:47) (atime : 2022-04-08 07:46:47)
[7] ./usr/lib/x86_64-linux-gnu/dri/radeonsi_dri.so (mtime : 2022-04-08 07:46:47) (atime : 2022-04-08 07:46:47)
[8] ./usr/lib/x86_64-linux-gnu/dri/swrast_dri.so (mtime : 2022-04-08 07:46:47) (atime : 2022-04-08 07:46:47)
[9] ./usr/lib/x86_64-linux-gnu/dri/virtio_gpu_dri.so (mtime : 2022-04-08 07:46:47) (atime : 2022-04-08 07:46:47)
[10] ./usr/lib/x86_64-linux-gnu/dri/vmwgfx_dri.so (mtime : 2022-04-08 07:46:47) (atime : 2022-04-08 07:46:47)
[11] ./usr/lib/x86_64-linux-gnu/dri/zink_dri.so (mtime : 2022-04-08 07:46:47) (atime : 2022-04-08 07:46:47)

---- Identical files #42738 (28,046,894 bytes - 2b0z0adbs519437a9ea977cabfd71993) ----
[1] ./usr/lib/x86_64-linux-gnu/liblicutadata.so.66.1 (mtime : 2022-04-08 03:40:17) (atime : 2022-04-08 03:40:17)
[2] ./snap/gnome-3-38-2004/99/usr/lib/x86_64-linux-gnu/liblicutadata.so.66.1 (mtime : 2021-11-23 06:50:52) (atime : 2021-11-23 06:50:52)

---- Identical files #42739 (45,703,168 bytes - a7db5b5fb734527e317c1116ce0c4473) ----
[1] ./var/lib/snappy/snaps/snappy_14978.snap (mtime : 2022-04-08 03:53:18) (atime : 2022-04-08 03:53:18)
[2] ./var/lib/snappy/seed/snaps/snappy_14978.snap (mtime : 2022-04-08 04:39:26) (atime : 2022-04-08 04:39:26)

---- Identical files #42740 (56,872,960 bytes - 424674c0e21ea3f03e125b6e0c271749) ----
[1] ./var/lib/snappy/snaps/store_558.snap (mtime : 2022-04-08 03:53:18) (atime : 2022-04-08 03:53:18)
[2] ./var/lib/snappy/seed/snaps/store_558.snap (mtime : 2022-04-08 04:39:26) (atime : 2022-04-08 04:39:26)

---- Identical files #42741 (64,917,508 bytes - 7aa38fd66c3447ac1ce8bd60c8ab5999) ----
[1] ./var/lib/snappy/snaps/core20_1328.snap (mtime : 2022-04-08 03:53:18) (atime : 2022-04-08 03:53:18)
[2] ./var/lib/snappy/seed/snaps/core20_1328.snap (mtime : 2022-04-08 04:39:26) (atime : 2022-04-08 04:39:26)

---- Identical files #42742 (68,378,624 bytes - 2f782abf8f52bc92c03ec9644004f28a) ----
[1] ./var/lib/snappy/snaps/gtk-common-themes_1519.snap (mtime : 2022-04-08 03:53:18) (atime : 2022-04-08 03:53:18)
[2] ./var/lib/snappy/seed/snaps/gtk-common-themes_1519.snap (mtime : 2022-04-08 04:39:26) (atime : 2022-04-08 04:39:26)

---- Identical files #42743 (260,841,472 bytes - 89c6bf5327f5a5a19fecfa5de3f4c009) ----
[1] ./var/lib/snappy/snaps/gnome-3-38-2004_99.snap (mtime : 2022-04-08 03:53:18) (atime : 2022-04-08 03:53:18)
[2] ./var/lib/snappy/seed/snaps/gnome-3-38-2004_99.snap (mtime : 2022-04-08 04:39:26) (atime : 2022-04-08 04:39:26)

Searching time: 3336:364271(sec:usec)
>>

```

- 루트 탐색

- proc, run, sys 제외
- 천 단위마다 , 로 구분

4-6) fmd5 옵션

4-6-1) 에러처리, exit

```

youngmin@ubuntu:~/lsp/p2$ ./ssu_sdup
20182615> fmd5 * ~ ~ .
T---- Identical files #1 (13 bytes - 2d31d97199fb287d6fc4f82ebd1b3f2) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/6.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[3] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

---- Identical files #2 (20 bytes - be497c2168e374f414a351c49379c01a) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[3] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[4] /home/youngmin/lsp/p2/test/b/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

---- Identical files #3 (513 bytes - a1c2c741136a7bf3149ff40775d23d4f) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

Searching time: 0:000775(sec:usec)

>>
index 입력이 없음
>> 1 z
올바른 옵션을 입력해주세요
>> 1 d 2 2 2
option 입력 초과
올바른 옵션을 입력해주세요
>> 2
옵션 입력 x
>> exit
>> Back to Prompt
20182615>

```

- 에러 처리

- exit 입력 시 메시지 출력 후 프롬프트 출력

4-6-2) d옵션

```

youngmin@ubuntu:~/lsp/p2$ ./ssu_sdup
20182615> fmd5 * ~ ~ .
T---- Identical files #1 (13 bytes - 2d31d97199fb287d6fc4f82ebd1b3f2) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/6.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[3] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

---- Identical files #2 (20 bytes - be497c2168e374f414a351c49379c01a) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[3] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[4] /home/youngmin/lsp/p2/test/b/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

---- Identical files #3 (513 bytes - a1c2c741136a7bf3149ff40775d23d4f) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

Searching time: 0:000832(sec:usec)

>> 1 d 2
"/home/youngmin/lsp/p2/test/6.txt" has been deleted in #1

---- Identical files #1 (13 bytes - 2d31d97199fb287d6fc4f82ebd1b3f2) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

---- Identical files #2 (20 bytes - be497c2168e374f414a351c49379c01a) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[3] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[4] /home/youngmin/lsp/p2/test/b/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

---- Identical files #3 (513 bytes - a1c2c741136a7bf3149ff40775d23d4f) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

>> 5 d 1
세트 범위 벗어남
>> 1 d 3
인덱스 범위 벗어남
>>

```

- 삭제 후 메시지 출력

- 에러 처리

```
---- Identical files #1 (13 bytes - 2d31d97199fb287d6fcb4f82ebd1b3f2) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

---- Identical files #2 (20 bytes - be497c2168e374f414a351c49379c01a) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[3] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[4] /home/youngmin/lsp/p2/test/b/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

---- Identical files #3 (513 bytes - a1c2c741136a7bf3149ff40775d23d4f) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

>> 5 d 1
세트 범위 벗어남
>> 1 d 3
인덱스 범위 벗어남
>> 1 d 1
"/home/youngmin/lsp/p2/test/3.txt" has been deleted in #1

---- Identical files #1 (20 bytes - be497c2168e374f414a351c49379c01a) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[3] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[4] /home/youngmin/lsp/p2/test/b/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

---- Identical files #2 (513 bytes - a1c2c741136a7bf3149ff40775d23d4f) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

>>
```

- 삭제 후 세트에 인덱스가 하나만 남으면 중복 리스트에서 제거

4-6-3) i옵션

```
---- Identical files #1 (13 bytes - 2d31d97199fb287d6fcb4f82ebd1b3f2) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 05:42:17) (atime : 2022-04-17 05:42:17)
[2] /home/youngmin/lsp/p2/test/6.txt (mtime : 2022-04-17 05:42:20) (atime : 2022-04-17 05:42:20)
[3] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

---- Identical files #2 (20 bytes - be497c2168e374f414a351c49379c01a) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[3] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[4] /home/youngmin/lsp/p2/test/b/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

---- Identical files #3 (513 bytes - a1c2c741136a7bf3149ff40775d23d4f) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

Searching time: 0:000812(sec:usec)

>> 2 i
Delete "/home/youngmin/lsp/p2/test/1.txt"? [y/n] N
Delete "/home/youngmin/lsp/p2/test/5.txt"? [y/n] n
Delete "/home/youngmin/lsp/p2/test/a/1.txt"? [y/n] 2
y, N, n 중 하나 입력
>> 2 i
Delete "/home/youngmin/lsp/p2/test/1.txt"? [y/n] n
Delete "/home/youngmin/lsp/p2/test/5.txt"? [y/n] N
Delete "/home/youngmin/lsp/p2/test/a/1.txt"? [y/n] Y
Delete "/home/youngmin/lsp/p2/test/b/1.txt"? [y/n] n

---- Identical files #1 (13 bytes - 2d31d97199fb287d6fcb4f82ebd1b3f2) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 05:42:17) (atime : 2022-04-17 05:42:17)
[2] /home/youngmin/lsp/p2/test/6.txt (mtime : 2022-04-17 05:42:20) (atime : 2022-04-17 05:42:20)
[3] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

---- Identical files #2 (20 bytes - be497c2168e374f414a351c49379c01a) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[3] /home/youngmin/lsp/p2/test/b/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

---- Identical files #3 (513 bytes - a1c2c741136a7bf3149ff40775d23d4f) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

>>
```

- 절대 경로 하나씩 보여주면서 삭제 여부 확인

- n, N, y, Y 모두 적용

- 이외 입력 들어오면 >> 출력 후 사용자 입력 대기
- y나 Y 입력 시 삭제

```
---- Identical files #1 (13 bytes - 2d31d97199fb287dfcb4f82ebd1b3f2) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 05:42:17) (atime : 2022-04-17 05:42:17)
[2] /home/youngmin/lsp/p2/test/6.txt (mtime : 2022-04-17 05:42:20) (atime : 2022-04-17 05:42:20)
[3] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

---- Identical files #2 (20 bytes - be497c2168e374f414a351c49379c01a) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[3] /home/youngmin/lsp/p2/test/b/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

---- Identical files #3 (513 bytes - a1c2c741136a7bf3149ff40775d23d4f) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

>> 2 i
Delete "/home/youngmin/lsp/p2/test/1.txt"? [y/n] exit
y, Y, n, N 중 하나 입력
>> 6 i
세트 범위 벗어남
>> ■
```

- 세트 벗어나면 에러 처리 후 입력 대기

4-6-4) f 옵션

```
youngmin@ubuntu:~/lsp/p2$ ./ssu_sdup
20182615> fmd5 * ~ ~ .
---- Identical files #1 (14 bytes - 6a4ee0a0a52ed8274c154b2e623045fa) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)
[2] /home/youngmin/lsp/p2/test/6.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)
[3] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 05:59:02) (atime : 2022-04-17 05:59:02)

---- Identical files #2 (20 bytes - be497c2168e374f414a351c49379c01a) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)
[3] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)
[4] /home/youngmin/lsp/p2/test/b/1.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)

---- Identical files #3 (513 bytes - a1c2c741136a7bf3149ff40775d23d4f) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

Searching time: 0:000808(sec:usec)

>> 4 f
세트 범위 벗어남
>> 1 f
Left file in #1 : /home/youngmin/lsp/p2/test/7.txt (2022-04-17 05:59:02)

---- Identical files #1 (20 bytes - be497c2168e374f414a351c49379c01a) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)
[3] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)
[4] /home/youngmin/lsp/p2/test/b/1.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)

---- Identical files #2 (513 bytes - a1c2c741136a7bf3149ff40775d23d4f) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

>>
```

- 가장 최근 파일 제외 모두 삭제
- ms단위 까지 비교
- 입력 오류 시 에러 처리
- 삭제 후 메시지 출력

4-6-5) t옵션

```
youngmin@ubuntu:~/lsp/p2$ ./ssu_sdup
20182615> fmd5 * ~ ~ .
---- Identical files #1 (14 bytes - 6a4ee0a0a52ed8274c154b2e623045fa) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 06:00:13) (atime : 2022-04-17 06:00:13)
[2] /home/youngmin/lsp/p2/test/6.txt (mtime : 2022-04-17 06:00:08) (atime : 2022-04-17 06:00:08)
[3] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 05:59:19) (atime : 2022-04-17 05:59:19)

---- Identical files #2 (20 bytes - be497c2168e374f414a351c49379c01a) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)
[3] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)
[4] /home/youngmin/lsp/p2/test/b/1.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)

---- Identical files #3 (513 bytes - a1c2c741136a7bf3149ff40775d23d4f) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

Searching time: 0:000807(sec:usec)

>> 4 t
세트 범위 벗어남
>> 1 t
All files in #1 have moved to Trash except "/home/youngmin/lsp/p2/test/3.txt" (2022-04-17 06:00:13)

---- Identical files #1 (20 bytes - be497c2168e374f414a351c49379c01a) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)
[3] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)
[4] /home/youngmin/lsp/p2/test/b/1.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)

---- Identical files #2 (513 bytes - a1c2c741136a7bf3149ff40775d23d4f) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

>> █
```

- 가장 최근 수정 파일 제외 모두 휴지통 이동
- ms 단위까지 비교
- 입력 오류 시 에러 처리
- 완료 후 메시지 출력

```
youngmin@ubuntu:~/lsp/p2$ ls
main.c    ssu_fnd-md5  ssu_find-md5.o  ssu_find-sha1.h  ssu_help.c  ssu_sdup   ssu_sdup.o
main.o    ssu_fnd-md5.c  ssu_find-sha1  ssu_find-sha1.o  ssu_help.h  ssu_sdup.c  test
makefile  ssu_fnd-md5.h  ssu_find-sha1.c  ssu_help      ssu_help.o  ssu_sdup.h  trash
youngmin@ubuntu:~/lsp/p2$ cd trash
youngmin@ubuntu:~/lsp/p2/trash$ ls
6.txt  7.txt
youngmin@ubuntu:~/lsp/p2/trash$
```

- trash 디렉토리 생성 후 이동 확인

4-6-6) 추가기능 : a옵션

```

youngmin@ubuntu:~/lsp/p2$ ./ssu_sdup
20182615> fnd5 * ~ ~ .
---- Identical files #1 (14 bytes - 6a4ee0a0a52ed8274c154b2e623045fa) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 06:00:26) (atime : 2022-04-17 06:00:26)
[2] /home/youngmin/lsp/p2/test/6.txt (mtime : 2022-04-17 06:03:20) (atime : 2022-04-17 06:03:20)
[3] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 06:03:20) (atime : 2022-04-17 06:03:20)

---- Identical files #2 (20 bytes - be497c2168e374f414a351c49379c01a) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)
[3] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)

---- Identical files #3 (513 bytes - a1c2c741136a7bf3149ff40775d23d4f) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)

Searching time: 0:000799(sec:usec)

>> 2 a
---- Identical files #1 (14 bytes - 6a4ee0a0a52ed8274c154b2e623045fa) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 06:00:26) (atime : 2022-04-17 06:00:26)
[2] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 06:03:20) (atime : 2022-04-17 06:03:20)

---- Identical files #2 (20 bytes - be497c2168e374f414a351c49379c01a) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)

>> 3 a
---- Identical files #1 (14 bytes - 6a4ee0a0a52ed8274c154b2e623045fa) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 06:00:26) (atime : 2022-04-17 06:00:26)
[2] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 06:03:20) (atime : 2022-04-17 06:03:20)

---- Identical files #2 (20 bytes - be497c2168e374f414a351c49379c01a) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)

>>

```

- 2 a 입력 시, 모든 세트의 2번째 인덱스 제거됨
- 3번째 세트는 제거 후 하나만 남았으므로 링크드 리스트에서 제거
- 3 a에서 3번째 인덱스는 없으므로 패스

4-7. sha1

```

youngmin@ubuntu:~/lsp/p2$ ./ssu_sdup
20182615> fsha1
명령어를 맞게 입력해주세요
20182615> fsha1 * ~ ~ .
명령어를 맞게 입력해주세요
20182615> fsha1 test ~ ~ /
올바른 확장자 입력이 아님
20182615> fsha1 test x ~ /
올바른 확장자 입력이 아님
20182615> fsha1 test ~ x /
올바른 확장자 입력이 아님
20182615> fsha1 test ~ ~ iii
올바른 확장자 입력이 아님
20182615> fsha1 test ~ ~ ~ ~ ~
명령어를 맞게 입력해주세요
20182615>

```

- 입력 잘못된 경우 예러 처리

```

20182615> fsha1 *.png ~ ~ .
No duplicates in /home/youngmin/lsp/p2
20182615> fsha1 *.txx ~ ~ .
No duplicates in /home/youngmin/lsp/p2
20182615> fsha1 *.txt ~ ~ .
---- Identical files #1 (14 bytes - 013d7ae67f6415221a1820061243b105f53aca3d) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 06:00:26) (atime : 2022-04-17 06:00:26)
[2] /home/youngmin/lsp/p2/test/6.txt (mtime : 2022-04-17 06:06:37) (atime : 2022-04-17 06:06:37)
[3] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 06:03:20) (atime : 2022-04-17 06:03:20)

---- Identical files #2 (20 bytes - 88a7b59d2e9172960b72b5f7839b9da453f3e9) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 06:06:44) (atime : 2022-04-17 06:06:44)
[3] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)

---- Identical files #3 (513 bytes - 1f36f2b80b8c9d45aec9b79828292acb7e42478c) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 06:06:54) (atime : 2022-04-17 06:06:54)

Searching time: 0:000352(sec:usec)

>>

```

- 확장자 검색

- 중복 파일 없을 경우 메시지 출력
- 탐색 마치면 >> 출력 후 사용자 입력 대기

```

20182615> fsha1 * ~ ~
---- Identical files #1 (1 bytes - adc83b19e793491b1c6ea0fd8b46cd9f32e592fc) ----
[1] /home/youngmin/.config/pulse/838b1e08d2d94289a3f8acddaa8b0df63-default-sink (mtime : 2022-04-17 04:41:47) (atime : 2022-04-17 04:41:47)
[2] /home/youngmin/.config/pulse/838b1e08d2d94289a3f8acddaa8b0df63-default-source (mtime : 2022-04-17 04:41:47) (atime : 2022-04-17 04:41:47)

---- Identical files #2 (5 bytes - fa7398e89c5c67747184b362f7d8d86a939eca) ----
[1] /home/youngmin/.config/user-dirs.locale (mtime : 2022-04-17 03:51:08) (atime : 2022-04-17 03:51:08)
[2] /home/youngmin/snap-store/558/.config/user-dirs.locale (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)

---- Identical files #3 (5 bytes - @aab1b3456784040cc4026fcf408221abedf5885) ----
[1] /home/youngmin/lsp/pl1/dir1/dir3/cc.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)
[2] /home/youngmin/lsp/pl1/dir2/dir1/dl3/dd.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)

---- Identical files #4 (14 bytes - 013d7ae67f6415221a1820061243b105f53aca3d) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 06:00:26) (atime : 2022-04-17 06:00:26)
[2] /home/youngmin/lsp/p2/test/6.txt (mtime : 2022-04-17 06:12:46) (atime : 2022-04-17 06:12:46)
[3] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 06:03:20) (atime : 2022-04-17 06:03:20)

---- Identical files #5 (17 bytes - 36d812a6e12918feb95ae10d720ca63543679c03) ----
[1] /home/youngmin/lsp/pl1/dir1/e.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)
[2] /home/youngmin/lsp/pl2/dir1/e.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)

---- Identical files #6 (20 bytes - 88a7b59d2e9172960b72b5f7839b9da2453f3e9) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 06:12:46) (atime : 2022-04-17 06:12:46)
[3] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)

---- Identical files #7 (200 bytes - b9945c60549c2d1bd5cce64ffca2dec7d28bda31) ----
[1] /home/youngmin/.cache/fontconfig/CACHEDIR.TAG (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)
[2] /home/youngmin/.cache/snap-store/Common/.cache/fontconfig/CACHEDIR.TAG (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)

---- Identical files #8 (236 bytes - 2ceec1d5be670877cf360e47f82f7e172d73e5311) ----
[1] /home/youngmin/.cache/mozilla/firefox/nprt4603.default-release/safefrowsing/allow-flashallow-digest256.sbstore (mtime : 2022-04-17 03:53:49)
[2] /home/youngmin/.cache/mozilla/firefox/nprt4603.default-release/safefrowsing/except-flashallow-digest256.sbstore (mtime : 2022-04-17 03:53:49)
[3] /home/youngmin/.cache/mozilla/firefox/nprt4603.default-release/safefrowsing/except-flashallow-digest256.sbstore (mtime : 2022-04-17 03:53:49)

---- Identical files #9 (#401 bytes - 8fd8282f317eb5c45d0e520cf2f7d9bc8c8a909) ----
[1] /home/youngmin/lsp/pl1/dtr1/g.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)
[2] /home/youngmin/lsp/pl1/dtr2/drl1/c2.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)

---- Identical files #10 (#476 bytes - 735d4ebed7446e0d69a0fs98abd6845469899hd2) ----
[1] /home/youngmin/lsp/pl1/p1/a.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)
[2] /home/youngmin/lsp/pl1/p1/test.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)
[3] /home/youngmin/lsp/pl1/dtr1/a.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)
[4] /home/youngmin/lsp/pl1/dtr2/drl1/a.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)

---- Identical files #11 (#476 bytes - 82fa86ceb826dd76cb1d9683c2a3b6512e78fibb) ----
[1] /home/youngmin/lsp/pl1/dlr1/b.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)
[2] /home/youngmin/lsp/pl1/dlr2/drl1/b.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)

---- Identical files #12 (513 bytes - 1f36f2b80b8c9d45aec9b79828292acb7e42478c) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 06:12:46) (atime : 2022-04-17 06:12:46)

---- Identical files #13 (48,814 bytes - b5df21605bcad00cd6a196c6d4c30d520537a9a) ----

```

- 모든 정규 파일 대상 탐색
- 홈 디렉토리(~) 탐색

```
youngmin@ubuntu:~/lsp/p2$ sudo su
[sudo] password for youngmin:
root@ubuntu:/home/youngmin/lsp/p2# ./ssu_sdup
20182615> fsha1 * ~ ~ ~
No duplicates in /root
20182615> 
```

- 다른 계정 로그인 시, 그 계정의 홈 디렉토리(~)로 적용

```
youngmin@ubuntu:~/lsp/p2$ ./ssu_sdup
20182615> fsha1 * 4KB ~ ~
---- Identical files #1 (48,814 bytes - b5df21605bcadc00cd6a196c6d4c30d520537a9a) ----
[1] /home/youngmin/.cache/mozilla/firefox/mprt4603.default-release-thumbnails/008ec4453ff31513f43893cba7aa31c8.png (mtime : 2022-04-17 04:42:12) (atime : 2022-04-17 04:42:12)
[2] /home/youngmin/.cache/mozilla/firefox/mprt4603.default-release-thumbnails/b37c4cfbdeae68780dea877a19bcd5b9.png (mtime : 2022-04-17 04:42:12) (atime : 2022-04-17 04:42:12)

Searching time: 0:025303(sec:usec)
>> 
```

- 최소 크기 조건 탐색

```
20182615> fsha1 * 1.2kb ~ ~
---- Identical files #1 (48,814 bytes - b5df21605bcadc00cd6a196c6d4c30d520537a9a) ----
[1] /home/youngmin/.cache/mozilla/firefox/mprt4603.default-release-thumbnails/008ec4453ff31513f43893cba7aa31c8.png (mtime : 2022-04-17 04:42:12) (atime : 2022-04-17 04:42:12)
[2] /home/youngmin/.cache/mozilla/firefox/mprt4603.default-release-thumbnails/b37c4cfbdeae68780dea877a19bcd5b9.png (mtime : 2022-04-17 04:42:12) (atime : 2022-04-17 04:42:12)

Searching time: 0:029768(sec:usec)
>> 
```

- KB, MB, GB의 경우 소수점 지원

```
20182615> fsha1 * 0 5 ~
---- Identical files #1 (1 bytes - adc83b19e793491b1c6ea0fd8b46cd9f32e592fc) ----
[1] /home/youngmin/.config/pulse/838b1e08d2d94289a3f8acdda8b6df63-default-sink (mtime : 2022-04-17 04:41:47) (atime : 2022-04-17 04:41:47)
[2] /home/youngmin/.config/pulse/838b1e08d2d94289a3f8acdda8b6df63-default-source (mtime : 2022-04-17 04:41:47) (atime : 2022-04-17 04:41:47)

---- Identical files #2 (5 bytes - fa73905e89c5c67747184ab362f7d0d86a939eca) ----
[1] /home/youngmin/.config/user-dirs.locale (mtime : 2022-04-17 03:51:00) (atime : 2022-04-17 03:51:00)
[2] /home/youngmin/snap/snap-store/558/.config/user-dirs.locale (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)

---- Identical files #3 (5 bytes - 0aa1bb3456784040cc4026fcf408221abedf5885) ----
[1] /home/youngmin/lsp/p1/dir1/dir3/cc.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)
[2] /home/youngmin/lsp/p1/dir2/dir1/dir3/dd.txt (mtime : 2022-04-17 03:57:40) (atime : 2022-04-17 03:57:40)

Searching time: 0:003562(sec:usec)
>> 
```

- 최대 크기 조건 탐색

```

youngmin@ubuntu: ~/lsp/p2
[1] /usr/lib/python3.8/subprocess.py (mtime : 2022-04-08 06:46:49) (atime : 2022-04-08 06:46:49)
[2] /snap/core20/1328/usr/lib/python3.8/subprocess.py (mtime : 2021-11-26 12:14:08) (atime : 2021-11-26 1
2:14:08)
[3] /snap/core20/1405/usr/lib/python3.8/subprocess.py (mtime : 2021-11-26 12:14:08) (atime : 2021-11-26 1
2:14:08)

---- Identical files #41804 (77,366 bytes - 1f7216f59ff75a23f79ca143c2da6793c03be180) ----
[1] /snap/core20/1328/usr/lib/python3.8/unittest/_pycache__/mock.cpython-38.pyc (mtime : 2022-01-13 22:4
5:08) (atime : 2022-01-13 22:45:08)
[2] /snap/core20/1405/usr/lib/python3.8/unittest/_pycache__/mock.cpython-38.pyc (mtime : 2022-03-17 22:4
6:27) (atime : 2022-03-17 22:46:27)

---- Identical files #41805 (77,397 bytes - 1b9a162c74ebb6d2c48ffbc01b88492e3dc9ef62) ----
[1] /usr/src/linux-hwe-5.13.0-30/include/linux/mfd/wm831x/regulator.h (mtime : 2022-04-08 04
:14:15) (atime : 2022-04-08 04:14:15)
[2] /usr/src/linux-hwe-5.13.0-39/include/linux/mfd/wm831x/regulator.h (mtime : 2022-04-08 04
:14:37) (atime : 2022-04-08 04:14:37)

---- Identical files #41806 (77,956 bytes - 35b62f5ab6e7642237303f4ab78b09ab7cc836ec) ----
[1] /usr/lib/python3.8/logging/__init__.py (mtime : 2022-04-08 06:46:49) (atime : 2022-04-08 06:46:49)
[2] /snap/core20/1328/usr/lib/python3.8/logging/__init__.py (mtime : 2021-11-26 12:14:08) (atime : 2021-1
1-26 12:14:08)
[3] /snap/core20/1405/usr/lib/python3.8/logging/__init__.py (mtime : 2021-11-26 12:14:08) (atime : 2021-1
1-26 12:14:08)

```

- 루트 탐색
- proc, run, sys 제외
- 천 단위마다 , 로 구분

4-8) sha1 옵션

4-8-1) 에러처리, exit

```

youngmin@ubuntu:~/lsp/p2$ ./ssu_sdup
20182615> fsha1 * ~ ~ .
---- Identical files #1 (14 bytes - 013d7ae67f6415221a1820061243b105f53aca3d) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 06:00:26) (atime : 2022-04-17 06:00:26)
[2] /home/youngmin/lsp/p2/test/6.txt (mtime : 2022-04-17 06:12:46) (atime : 2022-04-17 06:12:46)
[3] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 06:03:20) (atime : 2022-04-17 06:03:20)

---- Identical files #2 (20 bytes - 88a7b59d2e9172960b72b65f7839b9da2453f3e9) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 06:12:46) (atime : 2022-04-17 06:12:46)
[3] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)

---- Identical files #3 (513 bytes - 1f36f2b80b8c9d45aec9b79828292acb7e42478c) ----
[1] /home/youngmin/lsp/p2/test/4/test_copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 06:12:46) (atime : 2022-04-17 06:12:46)

Searching time: 0:000696(sec:usec)

>>
index 입력이 없음
>> 1 z
올바른 옵션을 입력해주세요
>> 1 d 2 2 2
option 입력 초과
올바른 옵션을 입력해주세요
>> 2
옵션 입력 x
>> exit
>> Back to Prompt
20182615>

```

- 에러 처리
- exit 입력 시 메시지 출력 후 프롬프트 출력

4-8-2) d옵션

```
youngmin@ubuntu:~/lsp/p2$ ./ssu_sdup
20182615> fsha1 * ~ ~ .
---- Identical files #1 (14 bytes - 013d7ae67f6415221a1820061243b105f53aca3d) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 06:00:26) (atime : 2022-04-17 06:00:26)
[2] /home/youngmin/lsp/p2/test/6.txt (mtime : 2022-04-17 06:12:46) (atime : 2022-04-17 06:12:46)
[3] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 06:03:20) (atime : 2022-04-17 06:03:20)

---- Identical files #2 (20 bytes - 88a7b59d2e9172960b72b65f7839b9da2453f3e9) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 06:12:46) (atime : 2022-04-17 06:12:46)
[3] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)

---- Identical files #3 (513 bytes - 1f36f2b80b8c9d45aec9b79828292acb7e42478c) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 06:12:46) (atime : 2022-04-17 06:12:46)

Searching time: 0:000669(sec:usec)

>> 1 d 2
"/home/youngmin/lsp/p2/test/6.txt" has been deleted in #1

---- Identical files #1 (14 bytes - 013d7ae67f6415221a1820061243b105f53aca3d) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 06:00:26) (atime : 2022-04-17 06:00:26)
[2] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 06:03:20) (atime : 2022-04-17 06:03:20)

---- Identical files #2 (20 bytes - 88a7b59d2e9172960b72b65f7839b9da2453f3e9) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 06:12:46) (atime : 2022-04-17 06:12:46)
[3] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)

---- Identical files #3 (513 bytes - 1f36f2b80b8c9d45aec9b79828292acb7e42478c) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 06:12:46) (atime : 2022-04-17 06:12:46)

>> 5 d 1
세트 범위 벗어남
>> 1 d 3
인덱스 범위 벗어남
>> █
```

- 삭제 후 메시지 출력

- 에러 처리

```
youngmin@ubuntu:~/lsp/p2$ ./ssu_sdup
20182615> fsha1 * ~ ~ .
---- Identical files #1 (14 bytes - 013d7ae67f6415221a1820061243b105f53aca3d) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 06:00:26) (atime : 2022-04-17 06:00:26)
[2] /home/youngmin/lsp/p2/test/6.txt (mtime : 2022-04-17 06:21:00) (atime : 2022-04-17 06:21:00)
[3] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 06:03:20) (atime : 2022-04-17 06:03:20)

---- Identical files #2 (20 bytes - 88a7b59d2e9172960b72b65f7839b9da2453f3e9) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 06:12:46) (atime : 2022-04-17 06:12:46)
[3] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)

---- Identical files #3 (513 bytes - 1f36f2b80b8c9d45aec9b79828292acb7e42478c) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 06:12:46) (atime : 2022-04-17 06:12:46)

Searching time: 0:000713(sec:usec)

>> 3 d 2
"/home/youngmin/lsp/p2/test/4/test.txt" has been deleted in #3

---- Identical files #1 (14 bytes - 013d7ae67f6415221a1820061243b105f53aca3d) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 06:00:26) (atime : 2022-04-17 06:00:26)
[2] /home/youngmin/lsp/p2/test/6.txt (mtime : 2022-04-17 06:21:00) (atime : 2022-04-17 06:21:00)
[3] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 06:03:20) (atime : 2022-04-17 06:03:20)

---- Identical files #2 (20 bytes - 88a7b59d2e9172960b72b65f7839b9da2453f3e9) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 06:12:46) (atime : 2022-04-17 06:12:46)
[3] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)

>> █
```

- 삭제 후 세트에 인덱스가 하나만 남으면 중복 리스트에서 제거

4-8-3) i 옵션

```
youngmin@ubuntu:/lsp/p2$ ./ssu_sdup
20182615> fsha1 * ~ ~ .
---- Identical files #1 (14 bytes - 013d7ae67f6415221a1820061243b105f53aca3d) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 06:00:26) (atime : 2022-04-17 06:00:26)
[2] /home/youngmin/lsp/p2/test/6.txt (mtime : 2022-04-17 06:21:11) (atime : 2022-04-17 06:21:11)
[3] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 06:03:20) (atime : 2022-04-17 06:03:20)

---- Identical files #2 (20 bytes - 88a7b59d2e9172960b72b65f7839b9da2453f3e9) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 06:12:46) (atime : 2022-04-17 06:12:46)
[3] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 05:58:29) (atime : 2022-04-17 05:58:29)

---- Identical files #3 (513 bytes - 1f36f2b80b8c9d45aec9b79828292acb7e42478c) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 06:21:54) (atime : 2022-04-17 06:21:54)

Searching time: 0:000732(sec:usec)

>> 2 i
Delete "/home/youngmin/lsp/p2/test/1.txt"? [y/n] N
Delete "/home/youngmin/lsp/p2/test/5.txt"? [y/n] n
Delete "/home/youngmin/lsp/p2/test/a/1.txt"? [y/n] 2
y, Y, n, N 중 하나 입력
>> 2 i
Delete "/home/youngmin/lsp/p2/test/1.txt"? [y/n] n
Delete "/home/youngmin/lsp/p2/test/5.txt"? [y/n] y
Delete "/home/youngmin/lsp/p2/test/a/1.txt"? [y/n] Y

---- Identical files #1 (14 bytes - 013d7ae67f6415221a1820061243b105f53aca3d) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 06:00:26) (atime : 2022-04-17 06:00:26)
[2] /home/youngmin/lsp/p2/test/6.txt (mtime : 2022-04-17 06:21:11) (atime : 2022-04-17 06:21:11)
[3] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 06:03:20) (atime : 2022-04-17 06:03:20)

---- Identical files #2 (513 bytes - 1f36f2b80b8c9d45aec9b79828292acb7e42478c) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 06:21:54) (atime : 2022-04-17 06:21:54)

>>
```

- 절대 경로 하나씩 보여주면서 삭제 여부 확인
- n, N, y, Y 모두 적용
- 이외 입력 들어오면 >> 출력 후 사용자 입력 대기
- y나 Y 입력 시 삭제
- 삭제 후 세트 내 인덱스 하나만 남은 경우 링크드 리스트에서 제거

```
---- Identical files #1 (14 bytes - 013d7ae67f6415221a1820061243b105f53aca3d) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 06:00:26) (atime : 2022-04-17 06:00:26)
[2] /home/youngmin/lsp/p2/test/6.txt (mtime : 2022-04-17 06:21:11) (atime : 2022-04-17 06:21:11)
[3] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 06:03:20) (atime : 2022-04-17 06:03:20)

---- Identical files #2 (513 bytes - 1f36f2b80b8c9d45aec9b79828292acb7e42478c) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 06:21:54) (atime : 2022-04-17 06:21:54)

>> 4 i
세트 벗어나면 뒤에 처리 후 입력 대기
>> █
```

- 세트 벗어나면 예상 처리 후 입력 대기

4-8-4) f 옵션

```

youngmin@ubuntu:~/lsp/p2$ ./ssu_sdup
20182615> fsha1 * ~ ~ .
---- Identical files #1 (14 bytes - 013d7ae67f6415221a1820061243b105f53aca3d) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 06:24:50) (atime : 2022-04-17 06:24:50)
[2] /home/youngmin/lsp/p2/test/6.txt (mtime : 2022-04-17 06:24:57) (atime : 2022-04-17 06:24:57)
[3] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 06:25:03) (atime : 2022-04-17 06:25:03)

---- Identical files #2 (20 bytes - 88a7b59d2e9172960b72b65f7839b9da2453f3e9) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 06:24:34) (atime : 2022-04-17 06:24:34)
[3] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 06:24:34) (atime : 2022-04-17 06:24:34)
[4] /home/youngmin/lsp/p2/test/b/1.txt (mtime : 2022-04-17 06:24:34) (atime : 2022-04-17 06:24:34)

---- Identical files #3 (513 bytes - 1f36f2b80b8c9d45aec9b79828292acb7e42478c) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 06:22:13) (atime : 2022-04-17 06:22:13)

Searching time: 0:000756(sec:usec)

>> 4 f
세트 범위 벗어남
>> 1 f
Left file in #1 : /home/youngmin/lsp/p2/test/7.txt (2022-04-17 06:25:03)

---- Identical files #1 (20 bytes - 88a7b59d2e9172960b72b65f7839b9da2453f3e9) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/5.txt (mtime : 2022-04-17 06:24:34) (atime : 2022-04-17 06:24:34)
[3] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 06:24:34) (atime : 2022-04-17 06:24:34)
[4] /home/youngmin/lsp/p2/test/b/1.txt (mtime : 2022-04-17 06:24:34) (atime : 2022-04-17 06:24:34)

---- Identical files #2 (513 bytes - 1f36f2b80b8c9d45aec9b79828292acb7e42478c) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 06:22:13) (atime : 2022-04-17 06:22:13)

>>

```

- 가장 최근 파일 제외 모두 삭제
- ms단위 까지 비교
- 입력 오류 시 예러 처리
- 삭제 후 메시지 출력

4-8-5) t옵션

```

youngmin@ubuntu:~/lsp/p2$ ./ssu_sdup
20182615> fsha1 * ~ ~ .
---- Identical files #1 (14 bytes - 013d7ae67f6415221a1820061243b105f53aca3d) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 06:26:26) (atime : 2022-04-17 06:26:26)
[2] /home/youngmin/lsp/p2/test/6.txt (mtime : 2022-04-17 06:27:48) (atime : 2022-04-17 06:27:48)
[3] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 06:27:48) (atime : 2022-04-17 06:27:48)

---- Identical files #2 (21 bytes - 1f4066c41d37723852362966cfea7a473e4e93e9) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 06:29:17) (atime : 2022-04-17 06:29:17)
[2] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 06:28:57) (atime : 2022-04-17 06:28:57)
[3] /home/youngmin/lsp/p2/test/b/1.txt (mtime : 2022-04-17 06:28:57) (atime : 2022-04-17 06:28:57)

---- Identical files #3 (513 bytes - 1f36f2b80b8c9d45aec9b79828292acb7e42478c) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 06:22:13) (atime : 2022-04-17 06:22:13)

Searching time: 0:000653(sec:usec)

>> 4 t
세트 범위 벗어남
>> 2 t
All files in #2 have moved to Trash except "/home/youngmin/lsp/p2/test/1.txt" (2022-04-17 06:29:17)

---- Identical files #1 (14 bytes - 013d7ae67f6415221a1820061243b105f53aca3d) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 06:26:26) (atime : 2022-04-17 06:26:26)
[2] /home/youngmin/lsp/p2/test/6.txt (mtime : 2022-04-17 06:27:48) (atime : 2022-04-17 06:27:48)
[3] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 06:27:48) (atime : 2022-04-17 06:27:48)

---- Identical files #2 (513 bytes - 1f36f2b80b8c9d45aec9b79828292acb7e42478c) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 06:22:13) (atime : 2022-04-17 06:22:13)

```

- 가장 최근 수정 파일 제외 모두 휴지통 이동

- ms 단위까지 비교
- 입력 오류 시 에러 처리
- 완료 후 메시지 출력

```
youngmin@ubuntu:~/lsp/p2$ ls
main.c    ssu_find-md5.o  ssu_find-sha1.h  ssu_help.c  ssu_sdup    ssu_sdup.o
main.o    ssu_find-md5.c  ssu_find-sha1    ssu_find-sha1.o  ssu_help.h  ssu_sdup.c  test
makefile   ssu_find-md5.h  ssu_find-sha1.c  ssu_help      ssu_help.o  ssu_sdup.h  trash
youngmin@ubuntu:~/lsp/p2$ cd trash
youngmin@ubuntu:~/lsp/p2/trash$ ls
1.txt  cp1.txt
youngmin@ubuntu:~/lsp/p2/trash$
```

- trash 디렉토리 생성 후 이동 확인
- 같은 이름의 파일이 여러 개이면, 앞에 cp[IDX]를 붙여 이동

4-8-6) 추가기능 : a옵션

```
youngmin@ubuntu:~/lsp/p2$ ./ssu_sdup
20182615> fsha1 * ~ ~ .
---- Identical files #1 (14 bytes - 013d7ae67f6415221a1820061243b105f53aca3d) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 06:26:26) (atime : 2022-04-17 06:26:26)
[2] /home/youngmin/lsp/p2/test/6.txt (mtime : 2022-04-17 06:27:48) (atime : 2022-04-17 06:27:48)
[3] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 06:27:48) (atime : 2022-04-17 06:27:48)

---- Identical files #2 (21 bytes - 1f4066c41d37723852362966cfea7a473e4e93e9) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 06:29:24) (atime : 2022-04-17 06:29:24)
[2] /home/youngmin/lsp/p2/test/a/1.txt (mtime : 2022-04-17 06:31:18) (atime : 2022-04-17 06:31:18)
[3] /home/youngmin/lsp/p2/test/b/1.txt (mtime : 2022-04-17 06:31:18) (atime : 2022-04-17 06:31:18)

---- Identical files #3 (513 bytes - 1f36f2b80b8c9d45aec9b79828292acb7e42478c) ----
[1] /home/youngmin/lsp/p2/test/4/test copy.txt (mtime : 2022-04-17 03:56:31) (atime : 2022-04-17 03:56:31)
[2] /home/youngmin/lsp/p2/test/4/test.txt (mtime : 2022-04-17 06:22:13) (atime : 2022-04-17 06:22:13)

Searching time: 0:000729(sec:usec)

>> 2 a
---- Identical files #1 (14 bytes - 013d7ae67f6415221a1820061243b105f53aca3d) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 06:26:26) (atime : 2022-04-17 06:26:26)
[2] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 06:27:48) (atime : 2022-04-17 06:27:48)

---- Identical files #2 (21 bytes - 1f4066c41d37723852362966cfea7a473e4e93e9) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 06:29:24) (atime : 2022-04-17 06:29:24)
[2] /home/youngmin/lsp/p2/test/b/1.txt (mtime : 2022-04-17 06:31:18) (atime : 2022-04-17 06:31:18)

>> 3 a
---- Identical files #1 (14 bytes - 013d7ae67f6415221a1820061243b105f53aca3d) ----
[1] /home/youngmin/lsp/p2/test/3.txt (mtime : 2022-04-17 06:26:26) (atime : 2022-04-17 06:26:26)
[2] /home/youngmin/lsp/p2/test/7.txt (mtime : 2022-04-17 06:27:48) (atime : 2022-04-17 06:27:48)

---- Identical files #2 (21 bytes - 1f4066c41d37723852362966cfea7a473e4e93e9) ----
[1] /home/youngmin/lsp/p2/test/1.txt (mtime : 2022-04-17 06:29:24) (atime : 2022-04-17 06:29:24)
[2] /home/youngmin/lsp/p2/test/b/1.txt (mtime : 2022-04-17 06:31:18) (atime : 2022-04-17 06:31:18)

>>
```

- 2 a 입력 시, 모든 세트의 2번째 인덱스 제거됨
- 3번째 세트는 제거 후 하나만 남았으므로 링크드 리스트에서 제거
- 3 a에서 3번째 인덱스는 없으므로 패스

6. 소스코드

<makefile>

all : ssu_sdup ssu_find-md5 ssu_find-sha1 ssu_help

ssu_sdup : main.o ssu_sdup.o

```
gcc main.o ssu_sdup.o -o ssu_sdup
```

ssu_find-md5 : ssu_find-md5.o

```
gcc -Wall ssu_find-md5.o -o ssu_find-md5 -lcrypto -lssl
```

ssu_find-sha1 : ssu_find-sha1.o

```
gcc -Wall ssu_find-sha1.o -o ssu_find-sha1 -lcrypto -lssl
```

ssu_help : ssu_help.o

```
gcc ssu_help.o -o ssu_help
```

main.o : main.c ssu_sdup.h

```
gcc -c main.c
```

ssu_sdup.o : ssu_sdup.c ssu_sdup.h

```
gcc -c ssu_sdup.c
```

ssu_help.o : ssu_help.c ssu_help.h

```
gcc -c ssu_help.c
```

ssu_find-md5.o : ssu_find-md5.c ssu_find-md5.h

```
gcc -c ssu_find-md5.c
```

```
ssu_find-sha1.o : ssu_find-sha1.c ssu_find-sha1.h
```

```
gcc -c ssu_find-sha1.c
```

```
clean :
```

```
rm *.o
```

```
rm ssu_sdup
```

```
rm ssu_find-md5
```

```
rm ssu_find-sha1
```

```
rm ssu_help
```

```
<main.c>

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

int main(){
    ssu_sdup();
    fprintf(stdout, "Prompt End\n");
    exit(0);
}
```

```
<ssu_sdup.c>

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <dirent.h> // scandir 사용
#include <ctype.h>
#include <stdbool.h>
#include <openssl/md5.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <pwd.h>
#include "ssu_sdup.h"

void ssu_sdup(){
    while (1){
        char *oper = malloc(sizeof(char) * BUF_SIZE);
        fprintf(stdout, "20182615> "); // 프롬프트 출력
        fgets(oper, BUF_SIZE, stdin); // 명령어 입력
        oper[strlen(oper)-1] = '\0'; // 공백 제거
        // 시작 공백 제거
        while(oper[0] == ' '){

```

```

    memmove(oper, oper + 1, strlen(oper));

}

char *splitOper[OPER_LEN] = {NULL, }; // 명령어 split
char *ptr = strtok(oper, " "); // 공백 기준으로 문자열 자르기

int idx = 0;
while (ptr != NULL){

    if(idx < OPER_LEN) splitOper[idx] = ptr;

    idx++;
    ptr = strtok(NULL, " ");
}

// fmd5 or fsha1 명령 시
if(splitOper[0] != NULL && (!strcmp(splitOper[0], "fmd5") || !strcmp(splitOper[0], "fsha1"))){

    // 명령어 인자 틀리면 에러 처리
    if(idx != OPER_LEN)

        fprintf(stderr, "명령어를 맞게 입력해주세요\n");

    // 확장자 에러 검사 (*.확장자)만 ok
    else if(strcmp(splitOper[1], "*") != 0 && (strlen(splitOper[1]) > 1 && splitOper[1][1] != '.'))

        fprintf(stderr, "올바른 확장자 입력이 아닙니다\n");

    // 최소, 최대 검사는 함수 내에서 진행
    else{

        // 파일경로에 ~입력시 홈디렉토리로 바꿈
        if(splitOper[4][0] == '~'){

            struct passwd *pwd;

```

기

```
if((pwd = getpwuid(getuid()) == NULL){ // 사용자 아이디, 홈 디렉토리 경로 얻
    fprintf(stderr, "user id error");

}

memmove(splitOper[4], splitOper[4] + 1, strlen(splitOper[4])); // 맨 처음 ~ 제거
char *str = malloc(sizeof(char) * BUF_SIZE); // 임시로 저장해둘 문자열
strcpy(str, splitOper[4]);
sprintf(splitOper[4], "%s%s", pwd->pw_dir, str); // 홈디렉토리/하위경로로 합쳐줌
free(str);

}

int fileOrDir = check_fileOrDir(splitOper[4]); // 파일 or 디렉토리인지 체크
// TARGET_DIRECTORY 검사 (디렉토리 아닌 경우)
if(fileOrDir != 2){

    fprintf(stderr, "디렉토리가 아님\n");
}

else{
    // 프로세스 생성 및 실행
    int pid, status;
    pid = fork();
    if(pid < 0){

        fprintf(stderr, "fork error :");
        exit(1);
    }

    else if(pid == 0){ // 0인 경우 자식 프로세스
        // fmd5 또는 sha1 실행
    }
}
```

```

        !strcmp(splitOper[0], "fmd5") ?

                execl("./ssu_find-md5",     splitOper[0],     splitOper[1],     splitOper[2],
splitOper[3], splitOper[4], NULL)

                :

                execl("./ssu_find-sha1",     splitOper[0],     splitOper[1],     splitOper[2],
splitOper[3], splitOper[4], NULL);

                exit(0);

        }

        else{ // 부모 프로세스

                wait(&status); // child 종료때까지 대기

        }

}

}

}

}

// exit 입력 시 종료

else if(splitOper[0] != NULL && !strcmp(oper, "exit")){

        // exit 뒤 인자 붙으면 help와 동일한 결과 출력

        if(splitOper[1] != NULL){

                // 프로세스 생성 및 실행

                int pid, status;

                pid = fork();

                if(pid < 0){

                        fprintf(stderr, "fork error :");

                        exit(1);

                }

                else if(pid == 0){ // 0인 경우 자식 프로세스

```

```
        execl("./ssu_help", "", NULL);

        exit(0);

    }

else{ // 부모 프로세스

    wait(&status); // child 종료때까지 대기

}

else break;

}

else if(splitOper[0] != NULL){ // 엔터키 입력 아닌 경우 명령어 사용법 출력

    // 프로세스 생성 및 실행

    int pid, status;

    pid = fork();

    if(pid < 0){

        fprintf(stderr, "fork error :");

        exit(1);

    }

    else if(pid == 0){ // 0인 경우 자식 프로세스

        execl("./ssu_help", "", NULL);

        exit(0);

    }

    else{ // 부모 프로세스

        wait(&status); // child 종료때까지 대기

    }

}

free(oper);
```

```
    }

}

// 파일, 디렉토리 판별(파일 : 1, 디렉토리 : 2 리턴)

int check_fileOrDir(char *path){

    struct stat st;

    int fileOrDir = 0;

    // 파일 정보 얻기

    if(lstat(path, &st) == -1){

        fprintf(stderr, "stat error -> checkfile\n");

        return -1;

    }

    // 파일 형식

    switch (st.st_mode & S_IFMT){

        case S_IFREG:

            fileOrDir = 1;

            break;

        case S_IFDIR:

            fileOrDir = 2;

            break;

        case S_IFIFO:

            fileOrDir = 0;

            break;

        case S_IFLNK:

            fileOrDir = 0;

            break;

    }

}
```

```
        break;
```

```
}
```

```
    return fileOrDir;
```

```
}
```

```
<ssu.sdup.h>

#ifndef MAIN_H
#define MAIN_H

#ifndef BUF_SIZE
#define BUF_SIZE 1024
#endif

#ifndef OPER_LEN
#define OPER_LEN 5
#endif

void ssu_sdup();
int check_fileOrDir(char *path);

#endif
```

```
<ssu_help.c>

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

int main(int argc, char *argv[]){
    ssu_help();
    exit(0);
}

void ssu_help(){
    printf("Usage:\n");
    printf("  > fmd5/fsha1  [FILE_EXTENSION] [MINSIZE] [MAXSIZE] [TARGET_DIRECTORY]\n");
    printf("  >>  [SET_INDEX] [OPTION ... ]\n");
    printf("          [OPTION ... ]\n");
    printf("  d  [LIST_IDX] : delete [LIST_IDX] file\n");
    printf("  i : ask for confirmation before delete\n");
    printf("  f : force delete except the recently modified file\n");
    printf("  t : force move to Trash except the recently modified file\n");
    printf("  [IDX] a : delete the [IDX] file for all sets\n");
    printf("  > help\n");
    printf("  > exit\n\n");
}
```

```
<ssu_help.h>
```

```
#ifndef SSU_HELP_H
```

```
#define SSU_HELP_H
```

```
void ssu_help();
```

```
#endif
```

```
<ssu_find-md5.c>

#include <stdio.h>

#include <sys/types.h> // stat 사용

#include <sys/stat.h> // stat 사용

#include <sys/time.h>

#include <unistd.h> // stat 사용

#include <fcntl.h>

#include <string.h> // string 관련 함수 사용

#include <stdlib.h>

#include <stdbool.h>

#include <time.h>

#include <dirent.h> // scandir 사용

#include <math.h> // modf 사용

#include <openssl/md5.h>

#include <sys/time.h> // gettimeofday 사용

#include <errno.h>

#include "ssu_find-md5.h"

int main(int argc, char *argv[OPER_LEN]){

    // 큐 선언

    queue q;

    init_queue(&q);

    // 링크드리스트 head 선언

    Node *head = malloc(sizeof(Node));

    head->next = NULL;
```

```

// 찾은 파일 저장해둘 fp선언

FILE *dt = fopen("writeReadData.txt", "w+");

if(dt == NULL){

    fprintf(stderr, "data file create error\n");

    exit(1);

}

struct timeval start;

gettimeofday(&start, NULL);

ssu_find_md5(argv, argv[4], start, head, &q, dt, true);

delete_list(head); // 링크드리스트 제거

exit(0);

}

// md5 관련 함수 실행

// 입력인자 : 명령어 split, 찾을 디렉토리 경로, 현재 링크드리스트, 현재 큐, 파일 포인터, 메인에서 왔는지

// 인자배열 : fmd5, 파일 확장자, 최소크기, 최대크기, 디렉토리

void ssu_find_md5(char *splitOper[OPER_LEN], char *find_path, struct timeval start, Node *list, queue *q, FILE *dt, bool from_main){

    struct dirent **filelist; // scandir 파일목록 저장 구조체

    int cnt; // return 값

    // scandir로 파일목록 가져오기 (디렉토리가 아닐 경우 예러)

    if((cnt = scandir(find_path, &filelist, scandirFilter, alphasort)) == -1){

        fprintf(stderr, "%s error, ERROR : %s\n", find_path, strerror(errno));

        return;

    }

}

```

```

// 절대경로 변환

char dir_path[BUF_SIZE];

if(realpath(find_path, dir_path) == NULL){

    fprintf(stderr, "realpath error : %s\n", strerror(errno));

    return;

}

char pathname[BUF_SIZE]; // 합성할 path이름

strcpy(pathname, dir_path);

strcat(pathname, "/"); // 처음에 / 제거하고 시작하므로 붙여줌

// 조건에 맞으면, 파일 : 리스트 조회 | 디렉토리 : 큐에 삽입

for(int i = 0; i < cnt; i++){

    // 합쳤던 이전 하위파일명 문자열 제거

    char* ptr = strrchr(pathname, '/'); // 합쳤던 /하위파일명 포인터 연결

    if(ptr)  strncpy(ptr, "", 1); // 합쳤던 문자열 제거

    // 현재 하위파일명 문자열 붙이기

    strcat(pathname, "/");

    strcat(pathname, filelist[i]->d_name); // 경로 + '/파일이름'

}

int fileOrDir = check_fileOrDir(pathname); // 파일 or 디렉토리인지 체크

// 파일일경우

if(fileOrDir == 1){


```

```

// *.(확장자)인 경우

if(strlen(splitOper[1]) > 1 ){

    char* ori_fname = strrchr(splitOper[1], '.'); // 지정 파일 확장자

    char* cmp_fname = strrchr(filelist[i]->d_name, '.'); // 가져온 파일 확장자

    // 확장자가 없거나 확장자가 다르면 패스

    if((cmp_fname == NULL) || strcmp(ori_fname, cmp_fname) != 0)

        continue;

}

// 파일 정보 조회

struct stat st;

// 파일 정보 얻기

if(lstat(pathname, &st) == -1){

    // 파일 읽기 권한 없으면 패스

    if(!st.st_mode & S_IRWXU) continue;

    fprintf(stderr, "stat error : %s\n", strerror(errno));

    continue;

}

long double filesize = (long double) st.st_size; // 파일크기 구하기

if(filesize == 0) continue; // 0바이트인경우 패스

// 최소 크기 이상인지 확인

if(strcmp(splitOper[2], "~"){

    long double min_byte = 0; // 비교할 최소 크기(byte)

    char *pos = NULL;

```

```

min_byte = strtold(splitOper[2], &pos); // 문자열 변환

// 수만 입력한 경우

if(!strcmp(pos, ""){

    double integer, fraction;

    fraction = modf(min_byte, &integer); // 정수부 소수부 분리

    // 실수 입력한 경우 예외

    if(fraction != 0){

        fprintf(stderr, "min size error : %s\n", strerror(errno));

        return;

    }

}

// KB = 1024byte

else if(!strcmp(pos, "kb") || !strcmp(pos, "KB")){

    min_byte *= 1024;

}

// MB = 1024KB

else if(!strcmp(pos, "mb") || !strcmp(pos, "MB")){

    min_byte *= (1024 * 1024);

}

// GB = 1024MB

else if(!strcmp(pos, "gb") || !strcmp(pos, "GB")){

    min_byte *= (1024 * 1024 * 1024);

}

else{

    fprintf(stderr, "min size error : %s\n", strerror(errno));

    return;
}

```

```

    }

    // 최소 크기보다 작은 경우 패스

    if(filesize < min_byte) continue;

}

// 최대 크기 이하인지 확인

if(strcmp(splitOper[3], "~")){
    long double max_byte = 0; // 비교할 최소 크기(byte)

    char *pos = NULL;

    max_byte = strtod(splitOper[3], &pos); // 실수, 문자열 분리


    // 수만 입력한 경우

    if(!strcmp(pos, "")){
        double integer, fraction;

        fraction = modf(max_byte, &integer);

        // 실수 입력한 경우 예러

        if(fraction != 0){

            fprintf(stderr, "min size error\n");

            return;
        }
    }

    // KB = 1024byte

    else if(!strcmp(pos, "kb") || !strcmp(pos, "KB")){
        max_byte *= 1024;
    }

    // MB = 1024KB

    else if(!strcmp(pos, "mb") || !strcmp(pos, "MB")){

```

```

        max_byte *= (1024 * 1024);

    }

// GB = 1024MB

else if(!strcmp(pos, "gb") || !strcmp(pos, "GB")){
    max_byte *= (1024 * 1024 * 1024);

}

else{
    fprintf(stderr, "max size error\n");
    return;
}

// 최대 크기보다 큰 경우 패스

if(filesize > max_byte) continue;

}

// 파일 읽기 권한 없으면 패스

if(!st.st_mode & S_IRWXU) continue;

// md5값 구하기

FILE *fp = fopen(pathname, "r");

if (fp == NULL) continue; // fopen 에러시 패스

unsigned char filehash[MD5_DIGEST_LENGTH * 2 + 1]; // 해시값 저장할 문자열

strcpy(filehash, get_md5(fp)); // 해시값 구해서 저장

fclose(fp);

// 동적할당 후 시간 포맷 구하기

```

```
char* mstr = (char*)malloc(sizeof(char) * BUF_SIZE);
char* astr = (char*)malloc(sizeof(char) * BUF_SIZE);
strcpy(mstr, get_time(st.st_mtime, mstr));
strcpy(astr, get_time(st.st_atime, mstr));

// 파일에 저장
if(dt != NULL){

    char size2str[BUF_SIZE];
    sprintf(size2str, "%lld", (long long)filesize);
    fputs("*", dt); // 체크 여부
    fputs("|", dt);
    fputs(size2str, dt); // 파일 크기
    fputs("|", dt);
    fputs(pathname, dt); // 파일 절대경로
    fputs("|", dt);
    fputs(mstr, dt); // mtime
    fputs("|", dt);
    fputs(astr, dt); // atime
    fputs("|", dt);
    fputs(filehash, dt);
    fputs("|", dt);
    fputs("\n", dt);
}

free(mstr);
free(astr);
}
```

```

// 디렉토리일 경우

else if(fileOrDir == 2){

    // 루트에서부터 탐색시, proc, run, sys 제외

    if(!strcmp(find_path, "/")){

        if((!strcmp(filelist[i]->d_name,      "proc")      ||      !strcmp(filelist[i]->d_name,      "run"))
|| !strcmp(filelist[i]->d_name, "sys"))

            continue;

    }

    push_queue(q, pathname); // 찾은 디렉토리경로 큐 추가

}

}

for(int i = 0; i < cnt; i++){

    free(filelist[i]);

}

free(filelist);

if(!from_main) return; // 처음 메인에서 실행한게 아니라면 리턴 (재귀 종료)

// 큐 빌때까지 bfs탐색(bfs이므로 절대경로, 아스키 순서로 정렬되어있음)

while (!isEmpty_queue(q)) {

    ssu_find_md5(splitOper, pop_queue(q), start, list, q, dt, false);

}

fclose(dt); // w모드 종료

dt = fopen("writeReadData.txt", "r+t"); // r+모드 실행 (체크 표시 남겨야 하므로)

if(dt == NULL) fprintf(stderr, "fopen read error\n");

// 중복파일 리스트 추가

```

```

file2list(dt, list);

fclose(dt);

// 데이터 파일 삭제
if(unlink("writeReadData.txt") == -1)
    fprintf(stderr, "writeReadData delete error\n");

struct timeval end;
gettimeofday(&end, NULL); // 종료 시간 측정

int list_size = get_listLen(list);

if(list_size == 0){
    if(realpath(find_path, dir_path) == NULL){
        fprintf(stderr, "realpath error : %s\n", strerror(errno));
        return;
    }
    fprintf(stdout, "No duplicates in %s\n", dir_path);
    return;
}

// 파일크기대로 정렬 (bfs이므로 파일크기 같을 경우 절대경로 짧은 순 -> 임의(아스키 코드 순))
sort_list(list, list_size);

print_list(list); // 리스트 출력
get_searchtime(start, end); // 탐색 시간 출력
option(list); // 옵션 실행

```

}

// 중복파일 리스트에 추가 (체크한 파일 : **, 체크 x : *)

void file2list(FILE * dt, Node *list){

char *line;

char *cmpline;

while (!feof(dt)){

char buf[BUF_SIZE * FILEDATA_SIZE]; // 한 라인 읽기

line = fgets(buf, BUF_SIZE * FILEDATA_SIZE, dt);

if(line == NULL) break; // 파일 끝인경우 종료

char *splitFile[FILEDATA_SIZE] = {NULL, }; // 파일 크기, 파일 경로, hash 분리

char *ptr = strtok(buf, "|"); // | 기준으로 문자열 자르기

int idx = 0;

while (ptr != NULL){

if(idx < FILEDATA_SIZE) splitFile[idx] = ptr;

idx++;

ptr = strtok(NULL, "|");

}

if(!strcmp(splitFile[0], "**")) continue; // 이미 중복 체크 됐다면, 패스

int now_ftell = ftell(dt); // 돌아갈 위치 저장

bool is_first = true; // 기준 파일 추가해줘야 하는지

// 중복 파일 있는지 체크

while (!feof(dt)){

int cmp_ftell = ftell(dt); // 체크 표시 위해 위치 저장

char cmp_buf[BUF_SIZE * FILEDATA_SIZE]; // 한 라인 읽기

```

cmpline = fgets(cmp_buf, BUF_SIZE * FILEDATA_SIZE, dt);

if(cmpline == NULL) break; // 파일 끝인경우 종료

char *cmp_split[FILEDATA_SIZE] = {NULL, }; // 파일 크기, 파일 경로, hash 분리

char *cmp_ptr = strtok(cmp_buf, "|"); // | 기준으로 문자열 자르기

int cmp_idx = 0;

while (cmp_ptr != NULL){

    if(cmp_idx < FILEDATA_SIZE) cmp_split[cmp_idx] = cmp_ptr;

    cmp_idx++;

    cmp_ptr = strtok(NULL, "|");

}

if(!strcmp(cmp_split[0], "**")) continue; // 이미 중복 체크 됐다면, 패스

// 파일 크기 다르면 패스

if(strcmp(splitFile[1], cmp_split[1])){

    continue;

// 해시값 같으면 리스트에 추가(처음 찾은 경우 기준 라인부터 추가)

    if(!strcmp(splitFile[5], cmp_split[5])){

        if(is_first){

            long long filesize = atol(splitFile[1]);

            append_list(list, filesize, splitFile[2], splitFile[3], splitFile[4], splitFile[5]); // 리스트

에 추가

        is_first = false;

    }

    // 리스트에 추가

    long long filesize = atol(cmp_split[1]);

    append_list(list, filesize, cmp_split[2], cmp_split[3], cmp_split[4], cmp_split[5]); // 리스트에
}

```

추가

```
fseek(dt, cmp_ftell, SEEK_SET); // 체크 위치로 이동  
fputs("**|", dt); // **으로 체크 표시  
fseek(dt, cmp_ftell, SEEK_SET); // 체크 위치로 이동  
}  
}  
fseek(dt, now_ftell, SEEK_SET); // 다시 위치로 이동  
}
```

}

void option(Node *list){

```
while(1){
```

```
if(!get_listLen(list)) break; // 리스트 없으면 탈출
```

```
fprintf(stdout, ">> ");
```

```
char oper[BUF_SIZE];
```

```
fgets(oper, BUF_SIZE, stdin); // 명령어 입력
```

```
oper[strlen(oper)-1] = '\0'; // 공백 제거
```

```
// 시작 공백 제거
```

```
while(oper[0] == ' '){
```

```
memmove(oper, oper + 1, strlen(oper));
```

```
}
```

```
char *splitOper[OPTION_LEN] = {NULL, }; // 명령어 split
```

```
char *ptr = strtok(oper, " "); // 공백 기준으로 문자열 자르기

int idx = 0;
bool goNext = true;

while (ptr != NULL){

    if(idx < OPTION_LEN) splitOper[idx] = ptr;

    else{

        fprintf(stderr, "option 입력 초과\n");

        goNext = false; // 다시 프롬프트 출력

        break;

    }

    idx++;

    ptr = strtok(NULL, " ");

}

// INDEX 입력 없을 경우

if(splitOper[0] == NULL){

    fprintf(stderr, "index 입력이 없음\n");

}

else if(!strcmp(splitOper[0], "exit")){

    fprintf(stdout, ">> Back to Prompt\n");

    break;

}

else if(splitOper[1] == NULL){

    fprintf(stderr, "옵션 입력 x\n");

}

else if(goNext && !strcmp(splitOper[1], "d")){ // d옵션
```

```

        if(splitOper[2] == NULL)

            fprintf(stderr, "LIST_IDX 입력 x\n");

        else option_d(splitOper, list);

    }

else if(goNext && !strcmp(splitOper[1], "i")){ // i옵션

    option_i(atoi(splitOper[0]), list);

}

else if(goNext && !strcmp(splitOper[1], "f")){ // f옵션

    option_f(atoi(splitOper[0]), list);

}

else if(goNext && !strcmp(splitOper[1], "t")){ // t옵션

    option_t(atoi(splitOper[0]), list);

}

else if(goNext && !strcmp(splitOper[1], "a")){ // 추가기능 : a옵션

    option_a(atoi(splitOper[0]), list);

}

else{

    fprintf(stderr, "올바른 옵션을 입력해주세요\n");

}

}

}

// d옵션

void option_d(char *splitOper[OPTION_LEN], Node *list){

    Node *cur = list->next;

```

```

char *set_idx = malloc(sizeof(char) * BUF_SIZE);
strcpy(set_idx, splitOper[0]);

// 세트 같을때까지 탐색
while (cur->set_num != atoi(set_idx)){
    if(cur->next == NULL){

        printf(stderr, "세트 범위 벗어남\n");
        free(set_idx);

        return;
    }

    cur = cur->next;
}

char *list_idx = malloc(sizeof(char) * BUF_SIZE);
strcpy(list_idx, splitOper[2]);

// 인덱스 같을때까지 탐색
while (cur->idx_num != atoi(list_idx)){
    // 만약 같은 세트가 아닐경우 인덱스 범위 벗어남
    if(cur->next == NULL || cur->set_num != atoi(set_idx)){

        printf(stderr, "인덱스 범위 벗어남\n");
        free(set_idx);

        free(list_idx);

        return;
    }

    cur = cur->next;
}

```

```
}
```

```
free(set_idx);
```

```
free(list_idx);
```

```
int set_num = cur->set_num; // 삭제할 세트 번호
```

```
int idx_num = cur->idx_num; // 삭제할 인덱스 번호
```

```
// 파일 삭제
```

```
if(unlink(cur->path) == -1)
```

```
    fprintf(stderr, "%s delete error", cur->path);
```

```
else{
```

```
    fprintf(stdout, "\"%s\" has been deleted in #%d\n", cur->path, set_num);
```

```
    del_node(list, set_num, idx_num); // 해당 노드 연결 리스트에서 삭제
```

```
    del_onlySet(list, set_num); // 하나만 남은 경우 제거
```

```
    print_list(list); // 프린트, 넘버링(인덱스 재배치)
```

```
    if(get_listLen(list)) fprintf(stdout, "\n"); // 학번 프롬프트 출력 시 \n x
```

```
}
```

```
}
```

```
// i옵션
```

```
void option_i(int set_idx, Node *list){
```

```
    Node *cur = list->next;
```

```
    Node *pre = list; // 삭제 시 cur 위치 복구해줄 포인터
```

```
// 세트 같을때까지 탐색
```

```

while (cur->set_num != set_idx){

    if(cur->next == NULL){

        fprintf(stderr, "세트 범위 벗어남\n");

        return;

    }

    pre = cur;

    cur = cur->next;

}

// 세트 내에서 탐색

while (cur->set_num == set_idx){

    char *oper = malloc(sizeof(char) * BUF_SIZE);

    fprintf(stdout, "Delete %s? [y/n] ", cur->path);

    fgets(oper, BUF_SIZE, stdin); // yes or no

    oper[strlen(oper)-1] = '\0'; // 공백 제거


    // 시작 공백 제거

    while(oper[0] == ' '){

        memmove(oper, oper + 1, strlen(oper));

    }

    if(!strcasecmp(oper, "Y") || !strcasecmp(oper, "y")){

        //파일 삭제

        if(unlink(cur->path) == -1){

            fprintf(stderr, "%s delete error", cur->path);

        }

    }

}

```

```

        return;

    }

    else{
        del_node(list, cur->set_num, cur->idx_num); // 해당 노드 연결 리스트에서 삭제
        cur = pre->next; // 삭제했으므로 cur 위치 복구
    }

}

else if(!strcasecmp(oper, "N") && !strcasecmp(oper, "n")){
    pre = cur;
    cur = cur->next;
}

else{
    fprintf(stderr, "y, Y, n, N 중 하나 입력\n");
    return;
}

free(oper);

if(cur == NULL) break; // 마지막인 경우 종료

}

fprintf(stdout, "\n");

del_onlySet(list, set_idx); // 하나만 남은 경우 제거
print_list(list); // 프린트

if(get_listLen(list)) fprintf(stdout, "\n"); // 학번 프롬프트 출력 시 \n x

}

// f옵션

```

```

void option_f(int set_idx, Node *list){

    Node *cur = list->next;

    Node *pre = list; // 삭제 시 cur 위치 복구해줄 포인터

    // 세트 같을때까지 탐색

    while (cur->set_num != set_idx){

        if(cur->next == NULL){

            fprintf(stderr, "세트 범위 벗어남\n");

            return;

        }

        pre = cur;

        cur = cur->next;

    }

    Node *recent = get_recent(set_idx, cur); // 가장 최근 시간 노드 구하기

    // 세트 내에서 탐색

    while (cur->set_num == set_idx){

        // 가장 최근 수정 노드 아니라면

        if(cur != recent){

            //파일 삭제

            if(unlink(cur->path) == -1){

                fprintf(stderr, "%s delete error", cur->path);

                return;

            }

            else{


```

```

        del_node(list, cur->set_num, cur->idx_num); // 해당 노드 연결 리스트에서 삭제
        cur = pre->next; // 삭제했으므로 cur 위치 복구
    }

}

else{ // 최근 수정 노드인 경우
    pre = cur;
    cur = cur->next;
}

if(cur == NULL) break; // 마지막인 경우 종료
}

```

fprintf(stdout, "Left file in #%-d : %s (%-15s)\n\n", recent->set_num, recent->path, recent->mtime);

del_onlySet(list, set_idx); // 하나만 남은 경우 제거

print_list(list); // 프린트

if(get_listLen(list)) fprintf(stdout, "\n"); // 학번 프롬프트 출력 시 \n x

}

// t옵션

```

void option_t(int set_idx, Node *list){
    // 휴지통 경로 생성 (이미 존재한 경우는 에러x)
    if(mkdir("trash", 0776) == -1 && errno != EEXIST){
        fprintf(stderr, "directory create error: %s\n", strerror(errno));
        exit(1);
    }
}

```

Node *cur = list->next;

Node *pre = list; // 삭제 시 cur 위치 복구해줄 포인터

```

// 세트 같을때까지 탐색

while (cur->set_num != set_idx){

    if(cur->next == NULL){

        fprintf(stderr, "세트 범위 벗어남\n");

        return;

    }

    pre = cur;

    cur = cur->next;

}

```

```
Node *recent = get_recent(set_idx, cur); // 가장 최근 시간 노드 구하기
```

```

// 세트 내에서 탐색

while (cur->set_num == set_idx){

    // 가장 최근 수정 노드 아니라면

    if(cur != recent){

        // 파일 이동을 위해 먼저 복사하기

        char *str = malloc(sizeof(char) * PATH_SIZE);

        sprintf(str, "trash%ss", strrchr(cur->path, '/') ); // trash 파일 경로 만들어주기

        if(link(cur->path, str) == -1){

            if(errno != EEXIST){

                fprintf(stderr, "link error\n");

                exit(1);

            }

            // 같은 이름의 파일이 존재할 경우 -> cp1, cp2, ... 이름붙여 휴지통으로 이동

```

```

else{

    int cpnum = 1;

    while(1){

        char str2[PATH_SIZE];

        if(strrchr(cur->path, '.') == NULL) // 확장자 없는 파일인경우

            sprintf(str2, "trash/cp%d", cpnum);

        else

            sprintf(str2, "trash/cp%d%s", cpnum, strrchr(cur->path, '.));

        cpnum++;

        if(!link(cur->path, str2) == -1 && errno == EEXIST) break; // 중복파일
    }
}

```

있으면 다음 숫자 이름붙임

```

}

}

}

free(str);

// 그 후 기존 파일 삭제

if(unlink(cur->path) == -1){

    fprintf(stderr, "%s delete error", cur->path);

    return;

}

else{

    del_node(list, cur->set_num, cur->idx_num); // 해당 노드 연결 리스트에서 삭제

    cur = pre->next; // 삭제했으므로 cur 위치 복구

}

}

```

```

else{ // 최근 수정 노드인 경우

    pre = cur;

    cur = cur->next;

}

if(cur == NULL) break; // 마지막인 경우 종료

}

fprintf(stdout, "All files in #%"PRIu64" have moved to Trash except %"PRIu64" (%-15s)\n\n", recent->set_num, recent->path,
recent->mtime);

del_onlySet(list, set_idx); // 하나만 남은 경우 제거

print_list(list); // 프린트

if(get_listLen(list)) fprintf(stdout, "\n"); // 학번 프롬프트 출력 시 \n x

}

// 추가기능 : a옵션

// [LIST_IDX] a : 세트의 해당번째 인덱스 모두 삭제

// 해당 인덱스 없을 경우 삭제 x

void option_a(int list_idx, Node *list){

    Node *cur = list->next;

    Node *pre = list; // 삭제 시 cur 위치 복구해줄 포인터

    // 세트 같을때까지 탐색

    while (cur != NULL){

        if(cur->idx_num == list_idx){ // 인덱스 같으면 삭제

            //파일 삭제

            if(unlink(cur->path) == -1){

                fprintf(stderr, "%s delete error", cur->path);


```

```

        return;

    }

    else{
        del_node(list, cur->set_num, cur->idx_num); // 노드 삭제

        cur = pre->next;
    }

}

else{
    pre = cur;

    cur = cur->next;
}

}

del_onlyList(list); // 하나 남은 경우 삭제
print_list(list); // 프린트

if(get_listLen(list)) fprintf(stdout, "%n"); // 학번 프롬프트 출력 시 %n x
}

// scandir 필터(. 과 .. 제거)

int scandirFilter(const struct dirent *info){

    if(strcmp(info->d_name, ".") == 0 || strcmp(info->d_name, "..") == 0){

        return 0; // .이나 .. 이면 filter
    }

    else return 1;
}

// 파일, 디렉토리 판별(파일 : 1, 디렉토리 : 2 리턴)

```

```
int check_fileOrDir(char *path){

    struct stat st;

    int fileOrDir = 0;

    // 파일 정보 얻기

    if(lstat(path, &st) == -1){

        fprintf(stderr, "stat error -> checkfile\n");

        return -1;

    }

    // 파일 형식

    switch (st.st_mode & S_IFMT){

        case S_IFREG:

            fileOrDir = 1;

            break;

        case S_IFDIR:

            fileOrDir = 2;

            break;

        case S_IFIFO:

            fileOrDir = 0;

            break;

        case S_IFLNK:

            fileOrDir = 0;

            break;

    }

    return fileOrDir;

}
```

```
// md5 값 조회

char *get_md5(FILE *fp){

    MD5_CTX c;

    static unsigned char md[MD5_DIGEST_LENGTH];
    static unsigned char buf[BUF_MAX];

    int fd = fileno(fp);

    MD5_Init(&c);

    int i = read(fd, buf, BUF_MAX);

    MD5_Update(&c, buf, (unsigned long)i);

    MD5_Final(md,&c);

    static char md5string[MD5_DIGEST_LENGTH * 2 + 1];

    for(int i = 0; i < MD5_DIGEST_LENGTH; ++i)

        sprintf(&md5string[i*2], "%02x", (unsigned int)md[i]);

    return md5string;
}
```

```
// 시간 정보 포맷에 맞게 변환

char* get_time(time_t stime, char * str){

    struct tm *tm;

    tm = localtime(&stime);

    strftime(str, BUF_SIZE, "%Y-%m-%d %H:%M:%S", tm);
```

```

return str;

}

// 탐색 소요 시간 계산

void get_searchtime(struct timeval start, struct timeval end){

    end.tv_sec -= start.tv_sec; // 초 부분 계산

    if(end.tv_usec < start.tv_usec){ // ms 연산 결과가マイ너스인 경우 고려

        end.tv_sec--;
        end.tv_usec += 1000000;

    }

    end.tv_usec -= start.tv_usec;

    fprintf(stdout, "Searching time: %ld:%06ld(sec:usec)\n\n", end.tv_sec, end.tv_usec);

}

const char *size2comma(long long n){

    static char comma_str[COMMA_SIZE];

    char str[COMMA_SIZE];

    int idx, len, cidx = 0, mod;

    sprintf(str, "%lld", n);

    len = strlen(str);

    mod = len % 3;

    for(idx = 0; idx < len; idx++){

        if(idx != 0 && (idx) % 3 == mod){


```

```

        comma_str[cidx++] = ',';
    }

    comma_str[cidx++] = str[idx];
}

comma_str[cidx] = 0x00;
return comma_str;
}

// 리스트 끝에 추가

void append_list(Node *list, long long filesize, char *path, char *mtime, char *atime, unsigned char hash[MD5_DIGEST_LENGTH]){

    // 리스트 빌 경우(처음인경우 포함)

    if(list -> next == NULL){

        Node *newNode = malloc(sizeof(Node));

        newNode->next = list->next;

        newNode->filesize = filesize;
        strcpy(newNode->path, path);
        strcpy(newNode->mtime, mtime);
        strcpy(newNode->atime, atime);
        strcpy(newNode->hash, hash);

        list->next = newNode;
    }
    else{
        Node *cur = list;

```

```
while (cur->next != NULL)

    cur = cur->next;

Node *newNode = malloc(sizeof(Node));

newNode->next = cur->next;

newNode->filesize = filesize;

strcpy(newNode->path, path);

strcpy(newNode->mtime, mtime);

strcpy(newNode->atime, atime);

strcpy(newNode->hash, hash);

cur->next = newNode;

}
```

}

// 리스트 크기 구하기

```
int get_listLen(Node *list){

    int cnt = -1; // head 제외

    Node *cur = list;

    while(cur != NULL){

        cnt++;

        cur = cur->next;

    }
```

```

return cnt;

}

// 리스트 전체 데이터 출력 and 라벨링

void print_list(Node *list){

    Node *cur = list->next;

    int cnt = 0;

    int small_cnt = 1;

    unsigned char pre_hash[BUF_SIZE] = "";

    while (cur != NULL){

        // 해시값이 다르면 다른 리스트출력

        if(strcmp(pre_hash, cur->hash)){

            cnt++;

            small_cnt = 1;

            if(cnt != 1) fprintf(stdout, "%n"); // 2번째 파일부터는 한칸 씩 더 띄워줌

            fprintf(stdout, "---- Identical files # %d (%s bytes - ", cnt, size2comma(cur->filesize));

            fprintf(stdout, "%s", cur->hash);

            fprintf(stdout, ") ----%n");

            strcpy(pre_hash, cur->hash);

        }

        cur->set_num = cnt; // 현재 세트 번호 저장

        cur->idx_num = small_cnt; // 세트 내 인덱스 번호 저장
    }
}

```

```
    fprintf(stdout, "[%d] %s (mtime : %-15s) (atime : %-15s)\n", small_cnt, cur->path, cur->mtime, cur->atime);

    small_cnt++;

    cur = cur->next;
}
```

```
}
```

// 메모리 해제

```
void delete_list(Node *list){
```

```
    Node *cur = list;
```

```
    Node *next;
```

```
    while (cur != NULL){
```

```
        next = cur->next;
```

```
        free(cur);
```

```
        cur = next;
    }
```

```
}
```

// 큐 초기화

```
void init_queue(queue *q){
```

```
    q->front = q->rear = NULL;
```

```
    q->cnt = 0;
}
```

// 특정 파일 삭제

```
void del_node(Node *list, int set_num, int idx_num){
```

```

Node *cur = list->next; // head 다음

if(cur == NULL){

    fprintf(stderr, "node is NULL\n");

    return; // 빈 리스트일 경우 리턴

}

Node *pre = list;

while(cur != NULL){

    // 삭제할 인덱스 도달하면

    if(cur->set_num == set_num && cur->idx_num == idx_num){

        // 해당 인덱스 삭제

        if(cur->next != NULL){ // 중간 인덱스 삭제할 경우

            pre->next = cur->next; // 이전 노드의 다음 -> 삭제할 노드의 다음

            free(cur);

            return;

        }

        else{ // 마지막 인덱스 삭제할 경우

            pre->next = NULL;

            cur->next = NULL;

            free(cur);

            return;

        }

    }

    else{

        pre = cur;

```

```

        cur = cur->next;

    }

}

// 삭제 작업후 세트에 인덱스 하나만 남았으면 삭제

void del_onlySet(Node *list, int set_num){

    Node *cur = list->next; // head 다음

    if(cur == NULL) return; // 빈 리스트일 경우 리턴

    Node *pre = list; // 삭제 시 cur 위치 복구해줄 포인터

    // 세트 같을때까지 탐색

    while(cur->set_num != set_num){

        // 세트가 완전히 삭제된경우

        if(cur->next == NULL) return;

        pre = cur;

        cur = cur->next;

    }

    // 중복 없고 마지막 파일인 경우 삭제

    if(cur->next == NULL){

        pre->next = NULL;

        cur->next = NULL;

        free(cur);

        cur = NULL;

    }
}

```

```
    }

else if(strcmp(cur->hash, cur->next->hash)){ // 중복 없는 경우 삭제
    pre->next = cur->next; // 이전 노드의 다음 -> 삭제할 노드의 다음
    free(cur);
    cur = pre->next;
}

}
```

// 같은파일 있는지 찾고 없으면 삭제

```
void del_onlyList(Node *list){

    Node *cur = list->next; // head 다음
    if(cur == NULL) return; // 빈 리스트일 경우 리턴

    Node *pre = list;
    int idx = 1; // 현재 cur 인덱스
    int cmp_idx; // 비교할 인덱스

    while(cur != NULL){

        cmp_idx = search_hash(list, idx, cur->hash); // 본인 제외 같은 해시 존재하는지 탐색
        // 같은 해시 값 없으면

        if(cmp_idx == -1){

            // 해당 인덱스 삭제

            if(cur->next != NULL){ // 중간 인덱스 삭제할 경우
                pre->next = cur->next; // 이전 노드의 다음 -> 삭제할 노드의 다음
                free(cur);
                cur = pre->next;
            }
        }
    }
}
```

```

        }

    else{ // 마지막 인덱스 삭제할 경우

        pre->next = NULL;

        cur->next = NULL;

        free(cur);

        cur = NULL;

    }

}

else{

    pre = cur;

    idx++;

    cur = cur->next;

}

}

}

// 해시 일치할경우 인덱스 반환

int search_hash(Node *list, int cmp_idx, unsigned char hash[MD5_DIGEST_LENGTH]){

    Node *cur = list->next; // head 다음

    int idx = 1;

    while(cur != NULL){

        // 본인이 아닌 같은 해시 찾은 경우

        if((idx != cmp_idx) && !strcmp(cur->hash, hash))

            return idx;

    }

}

```

```

cur = cur->next;
idx++;
}

// 못 찾은 경우
return -1;
}

// 리스트 버블정렬
// 파일크기순 정렬 (bfs이므로 파일크기 같을 경우 절대경로 짧은 순 -> 임의(아스키 코드 순))

void sort_list(Node *list, int list_size){

    Node *cur = list->next; // head 다음

    for (int i = 0; i < list_size; i++){
        if(cur->next == NULL) break;

        for (int j = 0; j < list_size - 1 - i; j++){
            if(cur->filesize > cur->next->filesize)

                swap_node(cur, cur->next); // swap

            cur = cur->next;
        }

        cur = list->next;
    }
}

void swap_node(Node *node1, Node *node2){

    int fileSize;

    char path[BUF_SIZE];
    char mtime[BUF_SIZE];
}

```

```
char atime[BUF_SIZE];
unsigned char hash[BUF_SIZE];

fileSize = node1->filesize;
strcpy(path, node1->path);
strcpy(mtime, node1->mtime);
strcpy(atime, node1->atime);
strcpy(hash, node1->hash);

node1->filesize = node2->filesize;
strcpy(node1->path, node2->path);
strcpy(node1->mtime, node2->mtime);
strcpy(node1->atime, node2->atime);
strcpy(node1->hash, node2->hash);

node2->filesize = fileSize;
strcpy(node2->path, path);
strcpy(node2->mtime, mtime);
strcpy(node2->atime, atime);
strcpy(node2->hash, hash);
```

}

// 가장 최근 시간 노드 구하기

```
Node *get_recent(int set_idx, Node *cur){
    Node *recent;
```

```

// 파일 정보 조회

struct stat st;

time_t recent_time = -9999999;

// 세트 내에서 탐색

while (cur->set_num == set_idx){

    // 파일 정보 얻기

    if(lstat(cur->path, &st) == -1){

        fprintf(stderr, "stat error\n");

        exit(1);

    }

    // 가장 최근 시간 노드 구하기

    if(recent_time < st.st_mtime){

        recent_time = st.st_mtime;

        recent = cur;

    }

    cur = cur->next;

    if(cur == NULL) break; // 마지막인 경우 종료

}

return recent;

}

bool isEmpty_queue(queue *q){

    return q->cnt == 0;

}

```

```

// 큐에 데이터 삽입

void push_queue(queue *q, char path[BUF_SIZE]){
    Qnode *newNode = malloc(sizeof(Qnode)); // newNode 생성
    strcpy(newNode->path, path);
    newNode->next = NULL;

    if (isEmpty_queue(q)) // 큐가 비어있을 때
        q->front = newNode;
    else
        q->rear->next = newNode; // 맨 뒤의 다음을 newNode로 설정

    q->rear = newNode; // 맨 뒤를 newNode로 설정
    q->cnt++; // 큐 노드 개수 1 증가
}

// 큐 pop

char *pop_queue(queue *q){
    static char data[BUF_SIZE];
    Qnode *ptr;
    if (isEmpty_queue(q)){
        fprintf(stderr, "Error : Queue is empty!\n");
        return data;
    }
    ptr = q->front;
    strcpy(data, ptr->path);
    q->front = ptr->next; // ptr의 다음 노드를 front로 설정
}

```

```
free(ptr);

q->cnt--;

return data;

}
```

```
<ssu_find-md5.h>

#ifndef SSU_FIND_MD5_H
#define SSU_FIND_MD5_H

#ifndef BUF_MAX
#define BUF_MAX 1024 * 16
#endif

#ifndef BUF_SIZE
#define BUF_SIZE 1024
#endif

#ifndef PATH_SIZE
#define PATH_SIZE 4096
#endif

#ifndef FILEDATA_SIZE
#define FILEDATA_SIZE 7 // 마지막은 \n
#endif

#ifndef COMMA_SIZE
#define COMMA_SIZE 64
#endif

#ifndef OPER_LEN
#define OPER_LEN 5

```

```
#endif
```

```
#ifndef OPTION_LEN
```

```
#define OPTION_LEN 3
```

```
#endif
```

```
#include <openssl/md5.h>
```

```
// 동일 파일 링크드리스트
```

```
typedef struct Nodes{
```

```
    struct Nodes *next; // 다음 주소
```

```
    long long filesize; // 파일 크기(byte)
```

```
    char path[PATH_SIZE]; // 파일 경로
```

```
    char mtime[BUF_SIZE]; // mtime
```

```
    char atime[BUF_SIZE]; // atime
```

```
    unsigned char hash[BUF_SIZE]; // hash value
```

```
    int set_num; // 현재 세트 번호
```

```
    int idx_num; // 세트 내 인덱스 번호
```

```
}Node;
```

```
typedef struct QNode{
```

```
    char path[BUF_SIZE];
```

```
    struct QNode *next;
```

```
}Qnode;
```

```
// BFS 구현 용 큐
```

```

typedef struct Queue{
    Qnode *front;
    Qnode *rear;
    int cnt; // 큐 안의 노드 개수
}queue;

void ssu_find_md5(char *splitOper[OPER_LEN], char *find_path, struct timeval start, Node *list, queue *q, FILE *dt, bool from_main);

void file2list(FILE * dt, Node *list);

void option(Node *list);

void option_d(char *splitOper[OPTION_LEN], Node *list);

void option_i(int set_idx, Node *list);

void option_f(int set_idx, Node *list);

void option_t(int set_idx, Node *list);

void option_a(int list_idx, Node *list); // 추가기능

int scandirFilter(const struct dirent *info);

int check_fileOrDir(char *path);

char *get_md5(FILE *fp);

char* get_time(time_t stime, char * str);

void get_searchtime(struct timeval start, struct timeval end);

int get_listLen(Node *list);

const char *size2comma(long long n);

void append_list(Node *list, long long filesize, char *path, char *mtime, char *atime, unsigned char hash[MD5_DIGEST_LENGTH]);

```

```
void print_list(Node *list);

void delete_list(Node *list);

void del_node(Node *list, int set_num, int idx_num);

void del_onlySet(Node *list, int set_num);

void del_onlyList(Node *list);

int search_hash(Node *list, int cmp_idx, unsigned char hash[MD5_DIGEST_LENGTH]);

void sort_list(Node *list, int list_size);

void swap_node(Node *node1, Node *node2);

Node *get_recent(int set_idx, Node *cur);

void init_queue(queue *q);

bool isEmpty_queue(queue *q);

void push_queue(queue *q, char path[BUF_SIZE]);

char *pop_queue(queue *q);

#endif
```

```
<ssu_find-sha1.c>

#include <stdio.h>

#include <sys/types.h> // stat 사용

#include <sys/stat.h> // stat 사용

#include <sys/time.h>

#include <unistd.h> // stat 사용

#include <fcntl.h>

#include <string.h> // string 관련 함수 사용

#include <stdlib.h>

#include <stdbool.h>

#include <time.h>

#include <dirent.h> // scandir 사용

#include <math.h> // modf 사용

#include <openssl/sha.h>

#include <sys/time.h> // gettimeofday 사용

#include <errno.h>

#include "ssu_find-sha1.h"

int main(int argc, char *argv[OPER_LEN]){

    // 큐 선언

    queue q;

    init_queue(&q);

    // 링크드리스트 head 선언

    Node *head = malloc(sizeof(Node));

    head->next = NULL;
```

```

// 찾은 파일 저장해둘 fp선언

FILE *dt = fopen("writeReadData.txt", "w+");

if(dt == NULL){

    fprintf(stderr, "data file create error\n");

    exit(1);

}

struct timeval start;

gettimeofday(&start, NULL);

ssu_find_sha1(argv, argv[4], start, head, &q, dt, true);

delete_list(head); // 링크드리스트 제거

exit(0);

}

// sha1 관련 함수 실행

// 입력인자 : 명령어 split, 찾을 디렉토리 경로, 현재 링크드리스트, 현재 큐, 파일 포인터, 메인에서 왔는지

// 인자배열 : fsha1, 파일 확장자, 최소크기, 최대크기, 디렉토리

void ssu_find_sha1(char *splitOper[OPER_LEN], char *find_path, struct timeval start, Node *list, queue *q, FILE *dt, bool from_main){

    struct dirent **filelist; // scandir 파일목록 저장 구조체

    int cnt; // return 값

    // scandir로 파일목록 가져오기 (디렉토리가 아닐 경우 예러)

    if((cnt = scandir(find_path, &filelist, scandirFilter, alphasort)) == -1){

        fprintf(stderr, "%s error, ERROR : %s\n", find_path, strerror(errno));

        return;

    }

}

```

```

// 절대경로 변환

char dir_path[BUF_SIZE];

if(realpath(find_path, dir_path) == NULL){

    fprintf(stderr, "realpath error : %s\n", strerror(errno));

    return;

}

char pathname[BUF_SIZE]; // 합성할 path이름

strcpy(pathname, dir_path);

strcat(pathname, "/"); // 처음에 / 제거하고 시작하므로 붙여줌

// 조건에 맞으면, 파일 : 리스트 조회 | 디렉토리 : 큐에 삽입

for(int i = 0; i < cnt; i++){

    // 합쳤던 이전 하위파일명 문자열 제거

    char* ptr = strrchr(pathname, '/'); // 합쳤던 /하위파일명 포인터 연결

    if(ptr)  strncpy(ptr, "", 1); // 합쳤던 문자열 제거

    // 현재 하위파일명 문자열 붙이기

    strcat(pathname, "/");

    strcat(pathname, filelist[i]->d_name); // 경로 + '/파일이름'

}

int fileOrDir = check_fileOrDir(pathname); // 파일 or 디렉토리인지 체크

// 파일일경우

if(fileOrDir == 1){


```

```

// *.(확장자)인 경우

if(strlen(splitOper[1]) > 1 ){

    char* ori_fname = strrchr(splitOper[1], '.'); // 지정 파일 확장자

    char* cmp_fname = strrchr(filelist[i]->d_name, '.'); // 가져온 파일 확장자

    // 확장자가 없거나 확장자가 다르면 패스

    if((cmp_fname == NULL) || strcmp(ori_fname, cmp_fname) != 0)

        continue;

}

// 파일 정보 조회

struct stat st;

// 파일 정보 얻기

if(lstat(pathname, &st) == -1){

    // 파일 읽기 권한 없으면 패스

    if(!st.st_mode & S_IRWXU) continue;

    fprintf(stderr, "stat error : %s\n", strerror(errno));

    continue;

}

long double filesize = (long double) st.st_size; // 파일크기 구하기

if(filesize == 0) continue; // 0바이트인경우 패스

// 최소 크기 이상인지 확인

if(strcmp(splitOper[2], "~"){

    long double min_byte = 0; // 비교할 최소 크기(byte)

    char *pos = NULL;

```

```
min_byte = strtold(splitOper[2], &pos); // 실수, 문자열 분리

// 수만 입력한 경우

if(!strcmp(pos, ""){

    double integer, fraction;

    fraction = modf(min_byte, &integer);

    // 실수 입력한 경우 예외

    if(fraction != 0){

        fprintf(stderr, "min size error : %s\n", strerror(errno));

        return;

    }

}

// KB = 1024byte

else if(!strcmp(pos, "kb") || !strcmp(pos, "KB")){

    min_byte *= 1024;

}

// MB = 1024KB

else if(!strcmp(pos, "mb") || !strcmp(pos, "MB")){

    min_byte *= (1024 * 1024);

}

// GB = 1024MB

else if(!strcmp(pos, "gb") || !strcmp(pos, "GB")){

    min_byte *= (1024 * 1024 * 1024);

}

else{

    fprintf(stderr, "min size error : %s\n", strerror(errno));

    return;

}
```

```

    }

    // 최소 크기보다 작은 경우 패스

    if(filesize < min_byte) continue;

}

// 최대 크기 이하인지 확인

if(strcmp(splitOper[3], "~")){
    long double max_byte = 0; // 비교할 최소 크기(byte)

    char *pos = NULL;

    max_byte = strtod(splitOper[3], &pos); // 실수, 문자열 분리


    // 수만 입력한 경우

    if(!strcmp(pos, "")){
        double integer, fraction;

        fraction = modf(max_byte, &integer);

        // 실수 입력한 경우 예러

        if(fraction != 0){

            fprintf(stderr, "min size error\n");

            return;
        }
    }

    // KB = 1024byte

    else if(!strcmp(pos, "kb") || !strcmp(pos, "KB")){
        max_byte *= 1024;
    }

    // MB = 1024KB

    else if(!strcmp(pos, "mb") || !strcmp(pos, "MB")){

```

```

        max_byte *= (1024 * 1024);

    }

// GB = 1024MB

else if(!strcmp(pos, "gb") || !strcmp(pos, "GB")){
    max_byte *= (1024 * 1024 * 1024);

}

else{
    fprintf(stderr, "max size error\n");
    return;
}

// 최대 크기보다 큰 경우 패스

if(filesize > max_byte) continue;

}

// 파일 읽기 권한 없으면 패스

if(!st.st_mode & S_IRWXU) continue;

// sha1값 구하기

FILE *fp = fopen(pathname, "r");

if (fp == NULL) continue; // fopen 에러시 패스

unsigned char filehash[SHA_DIGEST_LENGTH * 2 + 1]; // 해시값 저장할 문자열

strcpy(filehash, get_sha1(fp)); // 해시값 구해서 저장

fclose(fp);

// 동적할당 후 시간 포맷 구하기

```

```
char* mstr = (char*)malloc(sizeof(char) * BUF_SIZE);
char* astr = (char*)malloc(sizeof(char) * BUF_SIZE);
strcpy(mstr, get_time(st.st_mtime, mstr));
strcpy(astr, get_time(st.st_atime, mstr));

// 파일에 저장
if(dt != NULL){

    char size2str[BUF_SIZE];
    sprintf(size2str, "%lld", (long long)filesize);
    fputs("*", dt); // 체크 여부
    fputs("|", dt);
    fputs(size2str, dt); // 파일 크기
    fputs("|", dt);
    fputs(pathname, dt); // 파일 절대경로
    fputs("|", dt);
    fputs(mstr, dt); // mtime
    fputs("|", dt);
    fputs(astr, dt); // atime
    fputs("|", dt);
    fputs(filehash, dt);
    fputs("|", dt);
    fputs("\n", dt);
}

free(mstr);
free(astr);
}
```

```

// 디렉토리일 경우

else if(fileOrDir == 2){

    // 루트에서부터 탐색시, proc, run, sys 제외

    if(!strcmp(find_path, "/")){
        if((!strcmp(filelist[i]->d_name, "proc") || !strcmp(filelist[i]->d_name, "run"))
        || !strcmp(filelist[i]->d_name, "sys"))

            continue;

    }

    push_queue(q, pathname); // 찾은 디렉토리경로 큐 추가

}

}

for(int i = 0; i < cnt; i++){

    free(filelist[i]);

}

free(filelist);

if(!from_main) return; // 처음 메인에서 실행한게 아니라면 리턴 (재귀 종료)

// 큐 빌때까지 bfs탐색(bfs이므로 절대경로, 아스키 순서로 정렬되어있음)

while (!isEmpty_queue(q)){

    ssu_find_sha1(splitOper, pop_queue(q), start, list, q, dt, false);

}

fclose(dt); // w모드 종료

dt = fopen("writeReadData.txt", "r+t"); // r+모드 실행 (체크 표시 남겨야 하므로)

if(dt == NULL) fprintf(stderr, "fopen read error\n");

// 중복파일 리스트 추가

```

```

file2list(dt, list);

fclose(dt);

// 데이터 파일 삭제
if(unlink("writeReadData.txt") == -1)
    fprintf(stderr, "writeReadData delete error\n");

struct timeval end;
gettimeofday(&end, NULL); // 종료 시간 측정

int list_size = get_listLen(list);

if(list_size == 0){
    if(realpath(find_path, dir_path) == NULL){
        fprintf(stderr, "realpath error : %s\n", strerror(errno));
        return;
    }
    fprintf(stdout, "No duplicates in %s\n", dir_path);
    return;
}

// 파일크기대로 정렬 (bfs이므로 파일크기 같을 경우 절대경로 짧은 순 -> 임의(아스키 코드 순))
sort_list(list, list_size);

print_list(list); // 리스트 출력
get_searchtime(start, end); // 탐색 시간 출력
option(list); // 옵션 실행

```

}

// 중복파일 리스트에 추가 (체크한 파일 : **, 체크 x : *)

void file2list(FILE * dt, Node *list){

char *line;

char *cmpline;

while (!feof(dt)){

char buf[BUF_SIZE * FILEDATA_SIZE]; // 한 라인 읽기

line = fgets(buf, BUF_SIZE * FILEDATA_SIZE, dt);

if(line == NULL) break; // 파일 끝인경우 종료

char *splitFile[FILEDATA_SIZE] = {NULL, }; // 파일 크기, 파일 경로, hash 분리

char *ptr = strtok(buf, "|"); // | 기준으로 문자열 자르기

int idx = 0;

while (ptr != NULL){

if(idx < FILEDATA_SIZE) splitFile[idx] = ptr;

idx++;

ptr = strtok(NULL, "|");

}

if(!strcmp(splitFile[0], "**")) continue; // 이미 중복 체크 됐다면, 패스

int now_ftell = ftell(dt); // 돌아갈 위치 저장

bool is_first = true; // 기준 파일 추가해줘야 하는지

// 중복 파일 있는지 체크

while (!feof(dt)){

int cmp_ftell = ftell(dt); // 체크 표시 위해 위치 저장

char cmp_buf[BUF_SIZE * FILEDATA_SIZE]; // 한 라인 읽기

```

cmpline = fgets(cmp_buf, BUF_SIZE * FILEDATA_SIZE, dt);

if(cmpline == NULL) break; // 파일 끝인경우 종료

char *cmp_split[FILEDATA_SIZE] = {NULL, }; // 파일 크기, 파일 경로, hash 분리

char *cmp_ptr = strtok(cmp_buf, "|"); // | 기준으로 문자열 자르기

int cmp_idx = 0;

while (cmp_ptr != NULL){

    if(cmp_idx < FILEDATA_SIZE) cmp_split[cmp_idx] = cmp_ptr;

    cmp_idx++;

    cmp_ptr = strtok(NULL, "|");

}

if(!strcmp(cmp_split[0], "**")) continue; // 이미 중복 체크 됐다면, 패스

// 파일 크기 다르면 패스

if(strcmp(splitFile[1], cmp_split[1])){

    continue;

// 해시값 같으면 리스트에 추가(처음 찾은 경우 기준 라인부터 추가)

    if(!strcmp(splitFile[5], cmp_split[5])){

        if(is_first){

            long long filesize = atol(splitFile[1]);

            append_list(list, filesize, splitFile[2], splitFile[3], splitFile[4], splitFile[5]); // 리스트

에 추가

        is_first = false;

    }

    // 리스트에 추가

    long long filesize = atol(cmp_split[1]);

    append_list(list, filesize, cmp_split[2], cmp_split[3], cmp_split[4], cmp_split[5]); // 리스트에
}

```

추가

```
fseek(dt, cmp_ftell, SEEK_SET); // 체크 위치로 이동  
fputs("**|", dt); // **으로 체크 표시  
fseek(dt, cmp_ftell, SEEK_SET); // 체크 위치로 이동  
}  
}  
fseek(dt, now_ftell, SEEK_SET); // 다시 위치로 이동  
}
```

}

void option(Node *list){

```
while(1){  
    if(!get_listLen(list)) break; // 리스트 없으면 탈출
```

```
fprintf(stdout, ">> ");
```

```
char oper[BUF_SIZE];  
fgets(oper, BUF_SIZE, stdin); // 명령어 입력  
oper[strlen(oper)-1] = '\0'; // 공백 제거
```

```
// 시작 공백 제거
```

```
while(oper[0] == ' '){  
    memmove(oper, oper + 1, strlen(oper));  
}
```

```
char *splitOper[OPTION_LEN] = {NULL, }; // 명령어 split
```

```
char *ptr = strtok(oper, " "); // 공백 기준으로 문자열 자르기

int idx = 0;
bool goNext = true;

while (ptr != NULL){

    if(idx < OPTION_LEN) splitOper[idx] = ptr;

    else{

        fprintf(stderr, "option 입력 초과\n");

        goNext = false; // 다시 프롬프트 출력

        break;

    }

    idx++;

    ptr = strtok(NULL, " ");

}

// INDEX 입력 없을 경우

if(splitOper[0] == NULL){

    fprintf(stderr, "index 입력이 없음\n");

}

else if(!strcmp(splitOper[0], "exit")){

    fprintf(stdout, ">> Back to Prompt\n");

    break;

}

else if(splitOper[1] == NULL){

    fprintf(stderr, "옵션 입력 x\n");

}

else if(goNext && !strcmp(splitOper[1], "d")){ // d옵션
```

```

        if(splitOper[2] == NULL)

            fprintf(stderr, "LIST_IDX 입력 x\n");

        else option_d(splitOper, list);

    }

else if(goNext && !strcmp(splitOper[1], "i")){ // i옵션

    option_i(atoi(splitOper[0]), list);

}

else if(goNext && !strcmp(splitOper[1], "f")){ // f옵션

    option_f(atoi(splitOper[0]), list);

}

else if(goNext && !strcmp(splitOper[1], "t")){ // t옵션

    option_t(atoi(splitOper[0]), list);

}

else if(goNext && !strcmp(splitOper[1], "a")){ // 추가기능 : a옵션

    option_a(atoi(splitOper[0]), list);

}

else{

    fprintf(stderr, "올바른 옵션을 입력해주세요\n");

}

}

}

// d옵션

void option_d(char *splitOper[OPTION_LEN], Node *list){

    Node *cur = list->next;

```

```

char *set_idx = malloc(sizeof(char) * BUF_SIZE);
strcpy(set_idx, splitOper[0]);

// 세트 같을때까지 탐색
while (cur->set_num != atoi(set_idx)){
    if(cur->next == NULL){

        printf(stderr, "세트 범위 벗어남\n");
        free(set_idx);

        return;
    }

    cur = cur->next;
}

char *list_idx = malloc(sizeof(char) * BUF_SIZE);
strcpy(list_idx, splitOper[2]);

// 인덱스 같을때까지 탐색
while (cur->idx_num != atoi(list_idx)){
    // 만약 같은 세트가 아닐경우 인덱스 범위 벗어남
    if(cur->next == NULL || cur->set_num != atoi(set_idx)){

        printf(stderr, "인덱스 범위 벗어남\n");
        free(set_idx);

        free(list_idx);

        return;
    }

    cur = cur->next;
}

```

```
}
```

```
free(set_idx);
```

```
free(list_idx);
```

```
int set_num = cur->set_num; // 삭제할 세트 번호
```

```
int idx_num = cur->idx_num; // 삭제할 인덱스 번호
```

```
// 파일 삭제
```

```
if(unlink(cur->path) == -1)
```

```
    fprintf(stderr, "%s delete error", cur->path);
```

```
else{
```

```
    fprintf(stdout, "\"%s\" has been deleted in #%d\n", cur->path, set_num);
```

```
    del_node(list, set_num, idx_num); // 해당 노드 연결 리스트에서 삭제
```

```
    del_onlySet(list, set_num); // 하나만 남은 경우 제거
```

```
    print_list(list); // 프린트, 넘버링(인덱스 재배치)
```

```
    if(get_listLen(list)) fprintf(stdout, "\n"); // 학번 프롬프트 출력 시 \n x
```

```
}
```

```
}
```

```
// i옵션
```

```
void option_i(int set_idx, Node *list){
```

```
    Node *cur = list->next;
```

```
    Node *pre = list; // 삭제 시 cur 위치 복구해줄 포인터
```

```
// 세트 같을때까지 탐색
```

```

while (cur->set_num != set_idx){

    if(cur->next == NULL){

        fprintf(stderr, "세트 범위 벗어남\n");

        return;

    }

    pre = cur;

    cur = cur->next;

}

// 세트 내에서 탐색

while (cur->set_num == set_idx){

    char *oper = malloc(sizeof(char) * BUF_SIZE);

    fprintf(stdout, "Delete %s? [y/n] ", cur->path);

    fgets(oper, BUF_SIZE, stdin); // yes or no

    oper[strlen(oper)-1] = '\0'; // 공백 제거


    // 시작 공백 제거

    while(oper[0] == ' '){

        memmove(oper, oper + 1, strlen(oper));

    }

    if(!strcasecmp(oper, "Y") || !strcasecmp(oper, "y")){

        //파일 삭제

        if(unlink(cur->path) == -1){

            fprintf(stderr, "%s delete error", cur->path);

        }

    }

}

```

```

        return;

    }

    else{
        del_node(list, cur->set_num, cur->idx_num); // 해당 노드 연결 리스트에서 삭제
        cur = pre->next; // 삭제했으므로 cur 위치 복구
    }

}

else if(!strcasecmp(oper, "N") && !strcasecmp(oper, "n")){
    pre = cur;
    cur = cur->next;
}

else{
    fprintf(stderr, "y, Y, n, N 중 하나 입력\n");
    return;
}

free(oper);

if(cur == NULL) break; // 마지막인 경우 종료
}

fprintf(stdout, "\n");

del_onlySet(list, set_idx); // 하나만 남은 경우 제거
print_list(list); // 프린트

if(get_listLen(list)) fprintf(stdout, "\n"); // 학번 프롬프트 출력 시 \n x

}

// f옵션

```

```

void option_f(int set_idx, Node *list){

    Node *cur = list->next;

    Node *pre = list; // 삭제 시 cur 위치 복구해줄 포인터

    // 세트 같을때까지 탐색

    while (cur->set_num != set_idx){

        if(cur->next == NULL){

            fprintf(stderr, "세트 범위 벗어남\n");

            return;

        }

        pre = cur;

        cur = cur->next;

    }

    Node *recent = get_recent(set_idx, cur); // 가장 최근 시간 노드 구하기

    // 세트 내에서 탐색

    while (cur->set_num == set_idx){

        // 가장 최근 수정 노드 아니라면

        if(cur != recent){

            //파일 삭제

            if(unlink(cur->path) == -1){

                fprintf(stderr, "%s delete error", cur->path);

                return;

            }

            else{


```

```

        del_node(list, cur->set_num, cur->idx_num); // 해당 노드 연결 리스트에서 삭제
        cur = pre->next; // 삭제했으므로 cur 위치 복구
    }

}

else{ // 최근 수정 노드인 경우
    pre = cur;
    cur = cur->next;
}

if(cur == NULL) break; // 마지막인 경우 종료
}

```

fprintf(stdout, "Left file in #%-d : %s (%-15s)\n\n", recent->set_num, recent->path, recent->mtime);

del_onlySet(list, set_idx); // 하나만 남은 경우 제거

print_list(list); // 프린트

if(get_listLen(list)) fprintf(stdout, "\n"); // 학번 프롬프트 출력 시 \n x

}

// t옵션

```

void option_t(int set_idx, Node *list){
    // 휴지통 경로 생성 (이미 존재한 경우는 에러x)
    if(mkdir("trash", 0776) == -1 && errno != EEXIST){
        fprintf(stderr, "directory create error: %s\n", strerror(errno));
        exit(1);
    }
}

```

Node *cur = list->next;

Node *pre = list; // 삭제 시 cur 위치 복구해줄 포인터

```

// 세트 같을때까지 탐색

while (cur->set_num != set_idx){

    if(cur->next == NULL){

        fprintf(stderr, "세트 범위 벗어남\n");

        return;

    }

    pre = cur;

    cur = cur->next;

}

```

```
Node *recent = get_recent(set_idx, cur); // 가장 최근 시간 노드 구하기
```

```

// 세트 내에서 탐색

while (cur->set_num == set_idx){

    // 가장 최근 수정 노드 아니라면

    if(cur != recent){

        // 파일 이동을 위해 먼저 복사하기

        char *str = malloc(sizeof(char) * PATH_SIZE);

        sprintf(str, "trash%ss", strrchr(cur->path, '/') ); // trash 파일 경로 만들어주기

        if(link(cur->path, str) == -1){

            if(errno != EEXIST){

                fprintf(stderr, "link error\n");

                exit(1);

            }

            // 같은 이름의 파일이 존재할 경우 -> cp1, cp2, ... 이름붙여 휴지통으로 이동

```

```

else{

    int cpnum = 1;

    while(1){

        char str2[PATH_SIZE];

        if(strrchr(cur->path, '.') == NULL) // 확장자 없는 파일인경우

            sprintf(str2, "trash/cp%d", cpnum);

        else

            sprintf(str2, "trash/cp%d%s", cpnum, strrchr(cur->path, '.));

        cpnum++;

        if(!link(cur->path, str2) == -1 && errno == EEXIST) break; // 중복파일
    }
}

```

있으면 다음 숫자 이름붙임

```

}

}

}

free(str);

// 그 후 기존 파일 삭제

if(unlink(cur->path) == -1){

    fprintf(stderr, "%s delete error", cur->path);

    return;

}

else{

    del_node(list, cur->set_num, cur->idx_num); // 해당 노드 연결 리스트에서 삭제

    cur = pre->next; // 삭제했으므로 cur 위치 복구

}

}

```

```

else{ // 최근 수정 노드인 경우

    pre = cur;

    cur = cur->next;

}

if(cur == NULL) break; // 마지막인 경우 종료

}

fprintf(stdout, "All files in #%"PRIu64" have moved to Trash except %"PRIu64" (%-15s)\n\n", recent->set_num, recent->path,
recent->mtime);

del_onlySet(list, set_idx); // 하나만 남은 경우 제거

print_list(list); // 프린트

if(get_listLen(list)) fprintf(stdout, "\n"); // 학번 프롬프트 출력 시 \n x

}

// 추가기능 : a옵션

// [LIST_IDX] a : 세트의 해당번째 인덱스 모두 삭제

// 해당 인덱스 없을 경우 삭제 x

void option_a(int list_idx, Node *list){

    Node *cur = list->next;

    Node *pre = list; // 삭제 시 cur 위치 복구해줄 포인터

    // 세트 같을때까지 탐색

    while (cur != NULL){

        if(cur->idx_num == list_idx){ // 인덱스 같으면 삭제

            //파일 삭제

            if(unlink(cur->path) == -1){

                fprintf(stderr, "%s delete error", cur->path);


```

```

        return;

    }

    else{
        del_node(list, cur->set_num, cur->idx_num); // 노드 삭제

        cur = pre->next;
    }

}

else{
    pre = cur;

    cur = cur->next;
}

}

del_onlyList(list); // 하나 남은 경우 삭제
print_list(list); // 프린트

if(get_listLen(list)) fprintf(stdout, "%n"); // 학번 프롬프트 출력 시 %n x
}

// scandir 필터(. 과 .. 제거)

int scandirFilter(const struct dirent *info){

    if(strcmp(info->d_name, ".") == 0 || strcmp(info->d_name, "..") == 0){

        return 0; // .이나 .. 이면 filter
    }

    else return 1;
}

// 파일, 디렉토리 판별(파일 : 1, 디렉토리 : 2 리턴)

```

```
int check_fileOrDir(char *path){

    struct stat st;

    int fileOrDir = 0;

    // 파일 정보 얻기

    if(lstat(path, &st) == -1){

        fprintf(stderr, "stat error -> checkfile\n");

        return -1;

    }

    // 파일 형식

    switch (st.st_mode & S_IFMT){

        case S_IFREG:

            fileOrDir = 1;

            break;

        case S_IFDIR:

            fileOrDir = 2;

            break;

        case S_IFIFO:

            fileOrDir = 0;

            break;

        case S_IFLNK:

            fileOrDir = 0;

            break;

    }

    return fileOrDir;

}
```

```
// sha1 값 조회

char *get_sha1(FILE *fp){

    SHA_CTX c;

    static unsigned char md[SHA_DIGEST_LENGTH];

    static unsigned char buf[BUF_MAX];

    int fd = fileno(fp);

    SHA1_Init(&c);

    int i = read(fd, buf, BUF_MAX);

    SHA1_Update(&c, buf, (unsigned long)i);

    SHA1_Final(md,&c);

    static char sha1string[SHA_DIGEST_LENGTH * 2 + 1];

    for(int i = 0; i < SHA_DIGEST_LENGTH; ++i)

        sprintf(&sha1string[i*2], "%02x", (unsigned int)md[i]);

    return sha1string;

}
```

```
// 시간 정보 포맷에 맞게 변환

char* get_time(time_t stime, char * str){

    struct tm *tm;

    tm = localtime(&stime);

    strftime(str, BUF_SIZE, "%Y-%m-%d %H:%M:%S", tm);
```

```

return str;

}

// 탐색 소요 시간 계산

void get_searchtime(struct timeval start, struct timeval end){

    end.tv_sec -= start.tv_sec; // 초 부분 계산

    if(end.tv_usec < start.tv_usec){ // ms 연산 결과가マイ너스인 경우 고려

        end.tv_sec--;
        end.tv_usec += 1000000;

    }

    end.tv_usec -= start.tv_usec;

    fprintf(stdout, "Searching time: %ld:%06ld(sec:usec)\n\n", end.tv_sec, end.tv_usec);

}

const char *size2comma(long long n){

    static char comma_str[COMMA_SIZE];

    char str[COMMA_SIZE];

    int idx, len, cidx = 0, mod;

    sprintf(str, "%lld", n);

    len = strlen(str);

    mod = len % 3;

    for(idx = 0; idx < len; idx++){

        if(idx != 0 && (idx) % 3 == mod){


```

```

        comma_str[cidx++] = ',';

    }

    comma_str[cidx++] = str[idx];

}

comma_str[cidx] = 0x00;

return comma_str;

}

// 리스트 끝에 추가

void append_list(Node *list, long long filesize, char *path, char *mtime, char *atime, unsigned char hash[SHA_DIGEST_LENGTH]){

    // 리스트 빌 경우(처음인경우 포함)

    if(list -> next == NULL){

        Node *newNode = malloc(sizeof(Node));

        newNode->next = list->next;

        newNode->filesize = filesize;

        strcpy(newNode->path, path);

        strcpy(newNode->mtime, mtime);

        strcpy(newNode->atime, atime);

        strcpy(newNode->hash, hash);

        list->next = newNode;

    }

    else{

        Node *cur = list;

```

```
while (cur->next != NULL)

    cur = cur->next;

Node *newNode = malloc(sizeof(Node));

newNode->next = cur->next;

newNode->filesize = filesize;

strcpy(newNode->path, path);

strcpy(newNode->mtime, mtime);

strcpy(newNode->atime, atime);

strcpy(newNode->hash, hash);

cur->next = newNode;

}
```

// 리스트 크기 구하기

```
int get_listLen(Node *list){

    int cnt = -1; // head 제외

    Node *cur = list;

    while(cur != NULL){

        cnt++;

        cur = cur->next;

    }
```

```

return cnt;

}

// 리스트 전체 데이터 출력 and 라벨링

void print_list(Node *list){

    Node *cur = list->next;

    int cnt = 0;

    int small_cnt = 1;

    unsigned char pre_hash[BUF_SIZE] = "";

    while (cur != NULL){

        // 해시값이 다르면 다른 리스트출력

        if(strcmp(pre_hash, cur->hash)){

            cnt++;

            small_cnt = 1;

            if(cnt != 1) fprintf(stdout, "%n"); // 2번째 파일부터는 한칸 씩 더 띄워줌

            fprintf(stdout, "---- Identical files # %d (%s bytes - ", cnt, size2comma(cur->filesize));

            fprintf(stdout, "%s", cur->hash);

            fprintf(stdout, ") ----%n");

            strcpy(pre_hash, cur->hash);

        }

        cur->set_num = cnt; // 현재 세트 번호 저장

        cur->idx_num = small_cnt; // 세트 내 인덱스 번호 저장
    }
}

```

```
    fprintf(stdout, "[%d] %s (mtime : %-15s) (atime : %-15s)\n", small_cnt, cur->path, cur->mtime, cur->atime);

    small_cnt++;

    cur = cur->next;
}
```

```
}
```

// 메모리 해제

```
void delete_list(Node *list){
```

```
    Node *cur = list;
```

```
    Node *next;
```

```
    while (cur != NULL){
```

```
        next = cur->next;
```

```
        free(cur);
```

```
        cur = next;
    }
```

```
}
```

// 큐 초기화

```
void init_queue(queue *q){
```

```
    q->front = q->rear = NULL;
```

```
    q->cnt = 0;
}
```

// 특정 파일 삭제

```
void del_node(Node *list, int set_num, int idx_num){
```

```

Node *cur = list->next; // head 다음

if(cur == NULL){

    fprintf(stderr, "node is NULL\n");

    return; // 빈 리스트일 경우 리턴

}

Node *pre = list;

while(cur != NULL){

    // 삭제할 인덱스 도달하면

    if(cur->set_num == set_num && cur->idx_num == idx_num){

        // 해당 인덱스 삭제

        if(cur->next != NULL){ // 중간 인덱스 삭제할 경우

            pre->next = cur->next; // 이전 노드의 다음 -> 삭제할 노드의 다음

            free(cur);

            return;

        }

        else{ // 마지막 인덱스 삭제할 경우

            pre->next = NULL;

            cur->next = NULL;

            free(cur);

            return;

        }

    }

    else{

        pre = cur;

```

```
        cur = cur->next;
    }
}

// 삭제 작업후 세트에 인덱스 하나만 남았으면 삭제

void del_onlySet(Node *list, int set_num){

    Node *cur = list->next; // head 다음

    if(cur == NULL) return; // 빈 리스트일 경우 리턴

    Node *pre = list; // 삭제 시 cur 위치 복구해줄 포인터

    // 세트 같을때까지 탐색

    while(cur->set_num != set_num){

        // 세트가 완전히 삭제된경우

        if(cur->next == NULL) return;

        pre = cur;

        cur = cur->next;

    }

    // 중복 없고 마지막 파일인 경우 삭제

    if(cur->next == NULL){

        pre->next = NULL;

        cur->next = NULL;

        free(cur);

        cur = NULL;

    }

}
```

```
    }

else if(strcmp(cur->hash, cur->next->hash)){ // 중복 없는 경우 삭제
    pre->next = cur->next; // 이전 노드의 다음 -> 삭제할 노드의 다음
    free(cur);
    cur = pre->next;
}

}
```

// 같은파일 있는지 찾고 없으면 삭제

```
void del_onlyList(Node *list){

    Node *cur = list->next; // head 다음
    if(cur == NULL) return; // 빈 리스트일 경우 리턴

    Node *pre = list;
    int idx = 1; // 현재 cur 인덱스
    int cmp_idx; // 비교할 인덱스

    while(cur != NULL){

        cmp_idx = search_hash(list, idx, cur->hash); // 본인 제외 같은 해시 존재하는지 탐색
        // 같은 해시 값 없으면

        if(cmp_idx == -1){
            // 해당 인덱스 삭제

            if(cur->next != NULL){ // 중간 인덱스 삭제할 경우
                pre->next = cur->next; // 이전 노드의 다음 -> 삭제할 노드의 다음
                free(cur);
                cur = pre->next;
            }
        }
    }
}
```

```

        }

    else{ // 마지막 인덱스 삭제할 경우

        pre->next = NULL;

        cur->next = NULL;

        free(cur);

        cur = NULL;

    }

}

else{

    pre = cur;

    idx++;

    cur = cur->next;

}

}

}

// 해시 일치할경우 인덱스 반환

int search_hash(Node *list, int cmp_idx, unsigned char hash[SHA_DIGEST_LENGTH]){

    Node *cur = list->next; // head 다음

    int idx = 1;

    while(cur != NULL){

        // 본인이 아닌 같은 해시 찾은 경우

        if((idx != cmp_idx) && !strcmp(cur->hash, hash))

            return idx;

    }

}

```

```

cur = cur->next;
idx++;
}

// 못 찾은 경우
return -1;
}

// 리스트 버블정렬
// 파일크기순 정렬 (bfs이므로 파일크기 같을 경우 절대경로 짧은 순 -> 임의(아스키 코드 순))

void sort_list(Node *list, int list_size){

    Node *cur = list->next; // head 다음

    for (int i = 0; i < list_size; i++){
        if(cur->next == NULL) break;

        for (int j = 0; j < list_size - 1 - i; j++){
            if(cur->filesize > cur->next->filesize)

                swap_node(cur, cur->next); // swap

            cur = cur->next;
        }

        cur = list->next;
    }
}

void swap_node(Node *node1, Node *node2){

    int fileSize;

    char path[BUF_SIZE];
    char mtime[BUF_SIZE];
}

```

```
char atime[BUF_SIZE];
unsigned char hash[BUF_SIZE];

fileSize = node1->filesize;
strcpy(path, node1->path);
strcpy(mtime, node1->mtime);
strcpy(atime, node1->atime);
strcpy(hash, node1->hash);

node1->filesize = node2->filesize;
strcpy(node1->path, node2->path);
strcpy(node1->mtime, node2->mtime);
strcpy(node1->atime, node2->atime);
strcpy(node1->hash, node2->hash);

node2->filesize = fileSize;
strcpy(node2->path, path);
strcpy(node2->mtime, mtime);
strcpy(node2->atime, atime);
strcpy(node2->hash, hash);
```

}

// 가장 최근 시간 노드 구하기

```
Node *get_recent(int set_idx, Node *cur){
    Node *recent;
```

```

// 파일 정보 조회

struct stat st;

time_t recent_time = -9999999;

// 세트 내에서 탐색

while (cur->set_num == set_idx){

    // 파일 정보 얻기

    if(lstat(cur->path, &st) == -1){

        fprintf(stderr, "stat error\n");

        exit(1);

    }

    // 가장 최근 시간 노드 구하기

    if(recent_time < st.st_mtime){

        recent_time = st.st_mtime;

        recent = cur;

    }

    cur = cur->next;

    if(cur == NULL) break; // 마지막인 경우 종료

}

return recent;

}

bool isEmpty_queue(queue *q){

    return q->cnt == 0;

}

```

```

// 큐에 데이터 삽입

void push_queue(queue *q, char path[BUF_SIZE]){
    Qnode *newNode = malloc(sizeof(Qnode)); // newNode 생성
    strcpy(newNode->path, path);
    newNode->next = NULL;

    if (isEmpty_queue(q)) // 큐가 비어있을 때
        q->front = newNode;
    else
        q->rear->next = newNode; // 맨 뒤의 다음을 newNode로 설정

    q->rear = newNode; // 맨 뒤를 newNode로 설정
    q->cnt++; // 큐 노드 개수 1 증가
}

// 큐 pop

char *pop_queue(queue *q){
    static char data[BUF_SIZE];
    Qnode *ptr;
    if (isEmpty_queue(q)){
        fprintf(stderr, "Error : Queue is empty!\n");
        return data;
    }
    ptr = q->front;
    strcpy(data, ptr->path);
    q->front = ptr->next; // ptr의 다음 노드를 front로 설정
}

```

```
free(ptr);

q->cnt--;

return data;

}
```

```
<ssu_find-sha1.h>

#ifndef SSU_FIND_SHA1_H
#define SSU_FIND_SHA1_H


#ifndef BUF_MAX
#define BUF_MAX 1024 * 16

#endif

#ifndef BUF_SIZE
#define BUF_SIZE 1024

#endif

#ifndef PATH_SIZE
#define PATH_SIZE 4096

#endif

#ifndef FILEDATA_SIZE
#define FILEDATA_SIZE 7 // 마지막은 \n

#endif

#ifndef COMMA_SIZE
#define COMMA_SIZE 64

#endif

#ifndef OPER_LEN
#define OPER_LEN 5
```

```
#endif
```

```
#ifndef OPTION_LEN
```

```
#define OPTION_LEN 3
```

```
#endif
```

```
#include <openssl/sha.h>
```

```
// 동일 파일 링크드리스트
```

```
typedef struct Nodes{
```

```
    struct Nodes *next; // 다음 주소
```

```
    long long filesize; // 파일 크기(byte)
```

```
    char path[PATH_SIZE]; // 파일 경로
```

```
    char mtime[BUF_SIZE]; // mtime
```

```
    char atime[BUF_SIZE]; // atime
```

```
    unsigned char hash[BUF_SIZE]; // hash value
```

```
    int set_num; // 현재 세트 번호
```

```
    int idx_num; // 세트 내 인덱스 번호
```

```
}Node;
```

```
typedef struct QNode{
```

```
    char path[BUF_SIZE];
```

```
    struct QNode *next;
```

```
}Qnode;
```

```
// BFS 구현 용 큐
```

```

typedef struct Queue{
    Qnode *front;
    Qnode *rear;
    int cnt; // 큐 안의 노드 개수
}queue;

void ssu_find_sha1(char *splitOper[OPER_LEN], char *find_path, struct timeval start, Node *list, queue *q, FILE *dt, bool from_main);

void file2list(FILE * dt, Node *list);

void option(Node *list);

void option_d(char *splitOper[OPTION_LEN], Node *list);

void option_i(int set_idx, Node *list);

void option_f(int set_idx, Node *list);

void option_t(int set_idx, Node *list);

void option_a(int list_idx, Node *list); // 추가기능

int scandirFilter(const struct dirent *info);

int check_fileOrDir(char *path);

char *get_sha1(FILE *fp);

char* get_time(time_t stime, char * str);

void get_searchtime(struct timeval start, struct timeval end);

int get_listLen(Node *list);

const char *size2comma(long long n);

void append_list(Node *list, long long filesize, char *path, char *mtime, char *atime, unsigned char hash[SHA_DIGEST_LENGTH]);

```

```
void print_list(Node *list);

void delete_list(Node *list);

void del_node(Node *list, int set_num, int idx_num);

void del_onlySet(Node *list, int set_num);

void del_onlyList(Node *list);

int search_hash(Node *list, int cmp_idx, unsigned char hash[SHA_DIGEST_LENGTH]);

void sort_list(Node *list, int list_size);

void swap_node(Node *node1, Node *node2);

Node *get_recent(int set_idx, Node *cur);

void init_queue(queue *q);

bool isEmpty_queue(queue *q);

void push_queue(queue *q, char path[BUF_SIZE]);

char *pop_queue(queue *q);

#endif
```