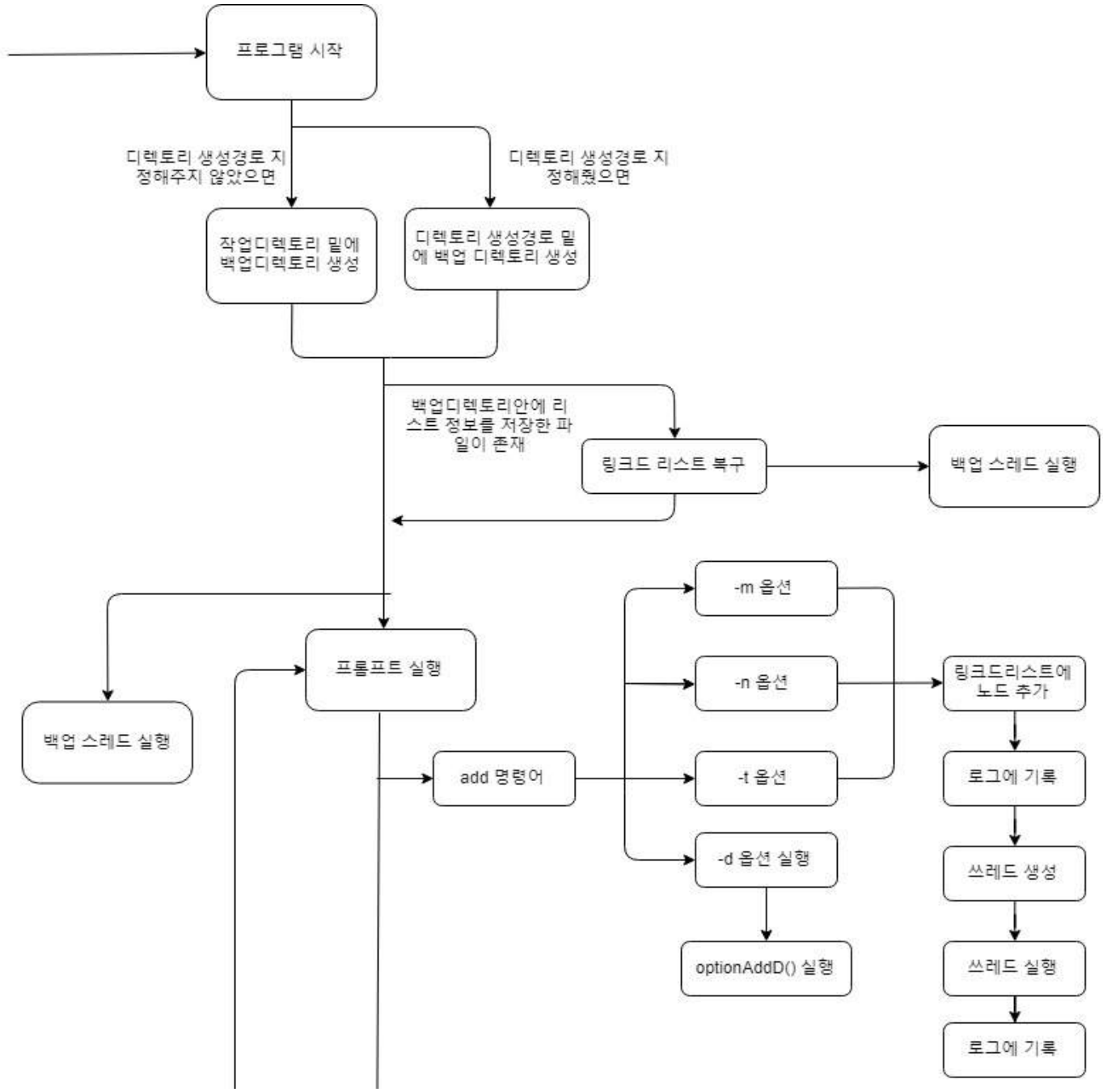
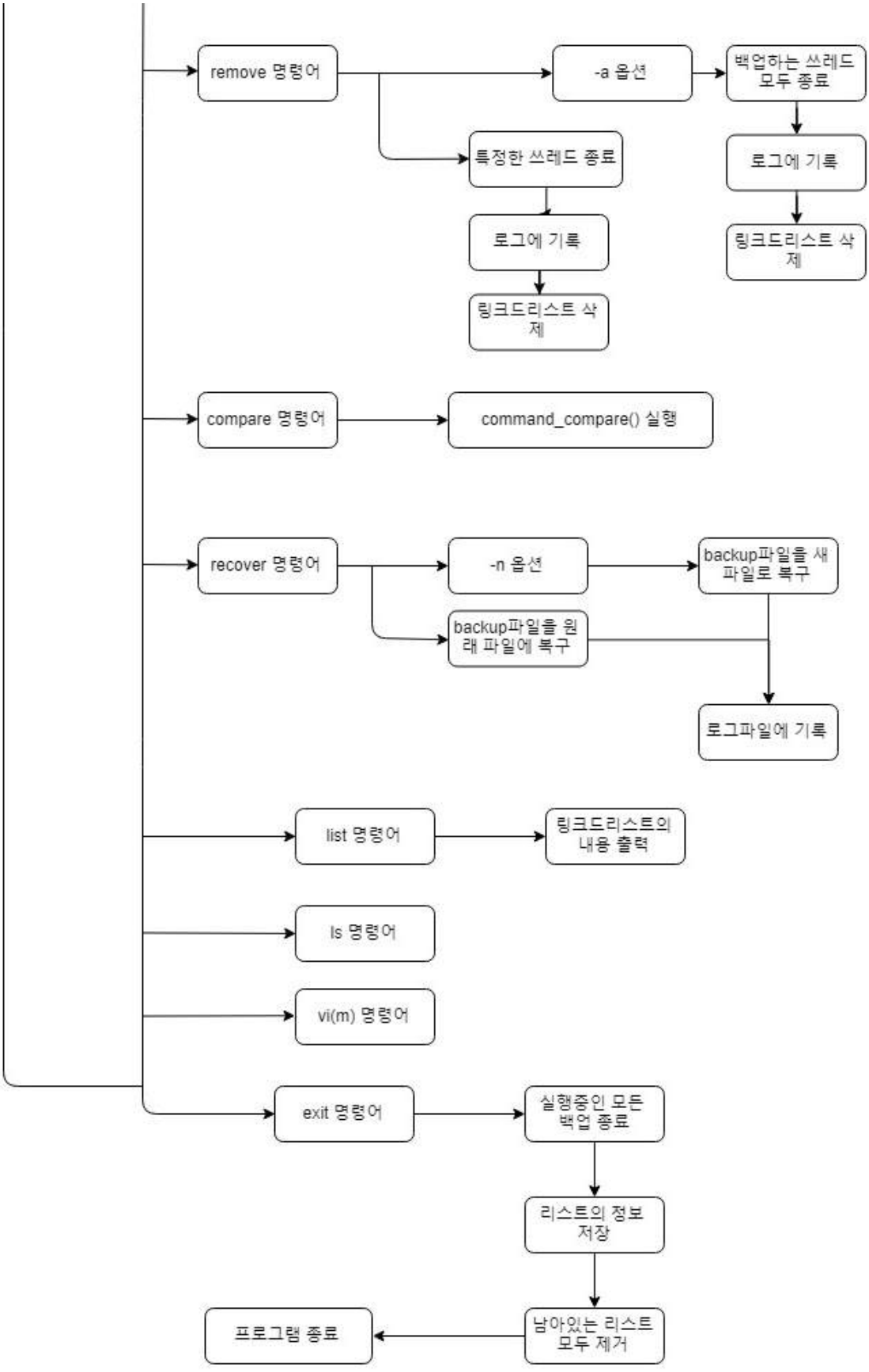
it융합학과 20160433 김민정

1. 과제 개요

ssu_backup은 main 쓰레드에서는 프롬프트를 계속 실행하면서 링크드리스트에 추가된 파일 하나마다 새로운 쓰레드를 만들어 백업하는 프로그램이다. 프롬프트상에서는 add, remove, recover, compare, list등의 명령어를 실행할 수 있으며 이러한 명령어들과 옵션들을 활용하여 사용자가 백업을 원하는 파일이나 디렉토리를, 추가, 삭제, 복구할 수 있게 하는 프로그램이다. 또한 log파일을 만들어 사용자가 백업을 원하는 파일을 추가하고, 삭제하고, 복구하고, 백업파일을 생성하는 과정을 기록으로 남긴다.

2. 설계




remove 명령어
-a 옵션
백업하는 쓰레드 모두 종료
로그에 기록
링크드리스트 삭제
특정한 쓰레드 종료
로그에 기록
링크드리스트 삭제
compare 명령어
command_compare() 실행
recover 명령어
-n 옵션
backup파일을 새 파일로 복구
backup파일을 원래 파일에 복구
로그파일에 기록
list 명령어
링크드리스트의 내용 출력
ls 명령어
vi(m) 명령어
exit 명령어
실행중인 모든 백업 종료
리스트의 정보 저장
남아있는 리스트 모두 제거
프로그램 종료

3. 구현
void Init(LinkedList *list);
싱글링크드리스트를 초기화하는 함수이다.

int getSize(LinkedList *list);
싱글링크드리스트의 노드개수를 반환하는 함수이다.

Node* create(char *file, int period, char *opt,char *filename,int mtime,int list_m_flag,int list_t_flag,int addTime,int list_n_flag, int number);
새로운 노드를 생성하는 함수이다.

void addPos(LinkedList *list, int pos, char *file, int period, char *opt, char *filename,int mtime, int list_m_flag,int list_t_flag,int addTime, int list_n_flag, int number);
특정 위치의 노드 뒤에 새로운 노드를 추가하는 함수이다.

void InsertNode(LinkedList *list, char *file, int period, char *opt,char *filename,int mtime,int list_m_flag,int list_t_flag,int addTime, int list_n_flag, int number);
싱글링크드리스트의 가장 뒤에 노드를 추가하는 함수이다.

void delNode(LinkedList *list, char*file);
싱글링크드리스트에서 원하는 항목을 찾아 삭제하는 함수이다. file이름을 비교해서 같은 file이름인 노드를 찾아서 삭제한다.

void delAllNode(LinkedList *list);
싱글링크드리스트에 있는 모든 노드들을 삭제하는 함수이다.

bool isEmpty(LinkedList *list);
싱글링크드리스트가 빈 리스트인지 검사하는 함수이다.

bool isInList(LinkedList *list, char*file);
싱글링크드리스트 안에 file과 같은 이름을 가진 노드가 존재하는지 여부를 판단하는 함수이다.

void print(LinkedList *list);
싱글링크드리스트의 모든 리스트내용을 출력하는 함수이다.

Node* searchNode(LinkedList *list, char*file);
리스트에서 원하는 항목 1개를 찾아서 반환하는 함수이다.

void command_add(char* command);
명령어 add를 실행시키는 함수이다.

void command_remove(char* command);
명령어 remove를 실행시키는 함수이다.

void command_compare(char* command);
명령어 compare를 실행시키는 함수이다. mtime과 파일크기가 같은 경우 두 파일은 동일한 파일로 취급한다.

void command_recover(char* command);
명령어 recover를 실행시키는 함수이다.

void command_list();
명령어 list를 실행시키는 함수이다.

void printLog(char *name, int index);
명령어 add, remove, recover와 백업파일을 생성시킬 때의 기록을 로그파일에 남기는 함수이다.

void *backupFile(void *arg);
쓰레드에서 실행할 함수이다. 리스트에 추가한 파일을 백업시키는 함수이다.

void optionAddN(char *filename,int number);
명령어 add의 -n옵션을 수행하는 함수이다.

void optionAddD(char *filename,int period,char* option);
명령어 add의 -d옵션을 수행하는 함수이다.

void recoverFileRead(char* fname);
백업한 파일로 변경 후 변경된 파일의 내용을 출력하는 함수이다.

void makeTable();
리스트 내용을 csv파일에 저장해주는 함수

void readTable();
csv파일을 읽어와 그 정보로 싱글링크드리스트를 복원하는 함수.

4. 테스트 및 결과

1) make파일을 이용하여 ssu_backup을 컴파일한다.

```
minjeong@minjeong-VirtualBox:~/lsp3$ ls
makefile  singlelinkedlist.h  ssu_backup.c
minjeong@minjeong-VirtualBox:~/lsp3$ make
gcc -o ssu_backup ssu_backup.c -lpthread
minjeong@minjeong-VirtualBox:~/lsp3$ ls
makefile  singlelinkedlist.h  ssu_backup  ssu_backup.c
```

2) ssu_backup실행

①백업디렉토리 경로를 설정하지 않고 ssu_backup을 실행했을 때

```
minjeong@minjeong-VirtualBox:~/lsp3$ ls
makefile  singlelinkedlist.h  ssu_backup  ssu_backup.c
minjeong@minjeong-VirtualBox:~/lsp3$ ./ssu_backup
20160433>add singlelinkedlist.h 5
20160433>ls backupDir
log.txt  singlelinkedlist.h_190530205004
20160433>exit
minjeong@minjeong-VirtualBox:~/lsp3$ ls
backupDir  makefile  singlelinkedlist.h  ssu_backup  ssu_backup.c
minjeong@minjeong-VirtualBox:~/lsp3$ ls backupDir
log.txt  singlelinkedlist.h_190530205004
```

②백업디렉토리 경로를 설정하여 ssu_backup을 실행했을 때

```
minjeong@minjeong-VirtualBox:~/lsp3$ ls
backupDir  makefile  singlelinkedlist.h  ssu_backup  ssu_backup.c  testBackup
minjeong@minjeong-VirtualBox:~/lsp3$ ls testBackup
minjeong@minjeong-VirtualBox:~/lsp3$ ./ssu_backup ./testBackup
20160433>add makefile 5
20160433>ls testBackup
backupDir
20160433>ls testBackup/backupDir
log.txt  makefile_190530205910  makefile_190530205915  makefile_190530205920
20160433>exit
minjeong@minjeong-VirtualBox:~/lsp3$ ls testBackup/backupDir
log.txt  makefile_190530205910  makefile_190530205915  makefile_190530205920  makefile_190530205925
```

③ 인자가 2개 이상인 경우

```
minjeong@minjeong-VirtualBox:~/lsp3$ ./ssu_backup testBackup backupDir
usage: ./ssu_backup <directory>
```

④ 인자로 입력받은 디렉토리를 찾을 수 없는 경우

```
minjeong@minjeong-VirtualBox:~/lsp3$ ./ssu_backup aaa
usage : directory is not exist
```

⑤ 인자로 입력받은 디렉토리가 디렉토리 파일이 아닐 경우

```
minjeong@minjeong-VirtualBox:~/lsp3$ ./ssu_backup a.c
usage : No directory
```

⑥ 인자로 입력받은 디렉토리의 접근권한이 없는 경우

```
minjeong@minjeong-VirtualBox:~/lsp3$ ls -al
total 172
drwxrwxr-x  5 minjeong minjeong  4096  5월 30 21:54 .
drwxr-xr-x 20 minjeong minjeong  4096  5월 30 20:56 ..
-rw-rw-r--  1 minjeong minjeong   156  5월 29 16:26 aaa.c
-rw-rw-r--  1 minjeong minjeong   286  5월 20 18:42 a.c
drwxrwxr-x  2 minjeong minjeong  4096  5월 30 21:26 backupDir
-rw-rw-r--  1 minjeong minjeong   157  5월 29 15:32 한글테스트.c
-rwxrwx---  1 minjeong minjeong    70  5월 29 02:34 makefile
d---------  2 minjeong minjeong  4096  5월 30 21:54 permission
-rw-rw-r--  1 minjeong minjeong  3907  5월 30 20:36 singlelinkedlist.h
-rwxrwxr-x  1 minjeong minjeong 36272  5월 30 21:53 ssu_backup
-rw-rw-r--  1 minjeong minjeong 31000  5월 30 21:53 ssu_backup.c
-rw-r--r--  1 minjeong minjeong 53248  5월 30 21:54 .ssu_backup.c.swp
-rw-rw-r--  1 minjeong minjeong   440  5월 30 21:07 t1.c
-rwxrwx---  1 minjeong minjeong  5567  5월 20 18:30 test2.c
drwxrwxr-x  3 minjeong minjeong  4096  5월 30 20:58 testBackup
minjeong@minjeong-VirtualBox:~/lsp3$ ./ssu_backup permission
usage : directory에 접근권한이 없습니다.
```

⑦ 프롬프트가 출력된 상태에서 엔터만 입력할 경우

```
minjeong@minjeong-VirtualBox:~/lsp3$ ./ssu_backup
20160433>
20160433>
```

⑧ 프롬프트가 출력된 상태에서 add, remove, compare, recover, list, ls, vi(m), exit 이외의 명령어 입력

```
20160433>command
존재하지 않는 명령어입니다.
20160433>
```

3) add명령어

① 옵션 없을 때

```
minjeong@minjeong-VirtualBox:~/lsp3$ ./ssu_backup
20160433>add makefile 5
20160433>list
/home/minjeong/lsp3/makefile       5
20160433>exit
minjeong@minjeong-VirtualBox:~/lsp3$ cat backupDir/log.txt
[190530 204959] /home/minjeong/lsp3/singlelinkedlist.h added
[190530 205004] /home/minjeong/lsp3/backupDir/singlelinkedlist.h_190530205004 generateds
[190530 211322] /home/minjeong/lsp3/makefile added
[190530 211327] /home/minjeong/lsp3/backupDir/makefile_190530211327 generated
```

② FILENAME 입력 없을 때

```
20160433>add
usage : add <FILENAME> [PERIOD] [OPTION]
20160433>
```

③ 백업해야할 파일이 존재하지 않을 때

```
20160433>ls
aaa.c  backupDir     makefile            ssu_backup    t1.c     testBackup
a.c    한글테스트.c  singlelinkedlist.h  ssu_backup.c  test2.c
20160433>add no.c
usage : file is not exist
```

④ 백업해야할 파일이 일반파일이 아닐 때

```
20160433>add backupDir 5
path is only Regular file
20160433>
```

⑤ 백업해야할 파일이 이미 백업리스트에 존재할 때

```
20160433>add t1.c 5
20160433>list
/home/minjeong/lsp3/t1.c        5
20160433>add t1.c 8
file is existed in backuplist
20160433>exit
```

⑥ period 입력없을 때

```
20160433>add t1.c
usage: <FILENAME> [period] [option]
20160433>
```

⑦ period를 실수형으로 입력했을 때 또는 $5 \leq period \leq 10$를 만족하지 않는 값을 입력했을 때

```
20160433>add aaa.c 7.7
period는 정수형이어야합니다
20160433>add aaa.c 3
5 <= period <= 10
20160433>add aaa.c 12
5 <= period <= 10
20160433>
```

⑧ m옵션

```
minjeong@minjeong-VirtualBox:~/lsp3$ stat inBackup.c
  File: 'inBackup.c'
  Size: 64              Blocks: 8          IO Block: 4096   regular file
Device: 801h/2049d      Inode: 142919      Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/minjeong)   Gid: ( 1000/minjeong)
Access: 2019-05-30 23:34:27.307633180 +0900
Modify: 2019-05-30 23:34:27.311633180 +0900
Change: 2019-05-30 23:34:27.323633180 +0900
 Birth: -
```

```
20160433>add inBackup.c 5 -m
20160433>ls backupDir
log.txt  makefile_190530211327  singlelinkedlist.h_190530205004  t1.c_190530212555  t1.c_190530212600
20160433>list
/home/minjeong/lsp3/inBackup.c       5   -m
20160433>ls backupDir
inBackup.c_190530221125  log.txt               singlelinkedlist.h_190530205004  t1.c_190530212600
inBackup.c_190530233430  makefile_190530211327  t1.c_190530212555
```

⑨ n옵션

```
20160433>add a.c 8 -n 5
20160433>ls backupDir
inBackup.c_190530221125  log.txt               singlelinkedlist.h_190530205004  t1.c_190530212600
inBackup.c_190530233430  makefile_190530211327  t1.c_190530212555
20160433>list
/home/minjeong/lsp3/inBackup.c       5   -m
/home/minjeong/lsp3/a.c       8   -n
20160433>ls backupDir
a.c_190530233602  a.c_190530233626         log.txt                          t1.c_190530212555
a.c_190530233610  inBackup.c_190530221125  makefile_190530211327             t1.c_190530212600
a.c_190530233618  inBackup.c_190530233430  singlelinkedlist.h_190530205004
20160433>ls backupDir
a.c_190530233618  a.c_190530233642         inBackup.c_190530233430  singlelinkedlist.h_190530205004
a.c_190530233626  a.c_190530233650         log.txt                  t1.c_190530212555
a.c_190530233634  inBackup.c_190530221125  makefile_190530211327     t1.c_190530212600
```

⑩ n옵션 수행할 때 NUMBER 입력하지 않을 경우

```
20160433>add a.c 8 -n
usage : -n NUMBER
```

⑪ t옵션

```
20160433>ls backupDir
aaa.c_190531004305  aaa.c_190531004317  inBackup.c_190531004239  log.txt               makefile_190530205915
aaa.c_190531004311  aaa.c_190531004323  inBackup.c_190531004246  makefile_190530205910  makefile_190531001645
20160433>list
20160433>add makefile 5 -t 60
20160433>list
/home/minjeong/lsp3/makefile       5   -t
20160433>ls backupDir
aaa.c_190531004305  aaa.c_190531004317  inBackup.c_190531004239  log.txt               makefile_190531004636
aaa.c_190531004311  aaa.c_190531004323  inBackup.c_190531004246  makefile_190531004631
20160433>ls backupDir
aaa.c_190531004305       inBackup.c_190531004246  makefile_190531004746  makefile_190531004811
aaa.c_190531004311       log.txt                  makefile_190531004751  makefile_190531004816
aaa.c_190531004317       makefile_190531004731    makefile_190531004756  makefile_190531004821
aaa.c_190531004323       makefile_190531004736    makefile_190531004801  makefile_190531004826
inBackup.c_190531004239  makefile_190531004741    makefile_190531004806
20160433>ls backupDir
aaa.c_190531004305       inBackup.c_190531004246  makefile_190531004806  makefile_190531004831
aaa.c_190531004311       log.txt                  makefile_190531004811  makefile_190531004836
aaa.c_190531004317       makefile_190531004751    makefile_190531004816  makefile_190531004841
aaa.c_190531004323       makefile_190531004756    makefile_190531004821  makefile_190531004846
inBackup.c_190531004239  makefile_190531004801    makefile_190531004826
20160433>
```

⑫ t옵션 수행할 때 TIME 입력하지 않을 경우

```
20160433>add ssu_backup.c 8 -t
usage : -t TIME
```

⑬ d옵션

```
20160433>add testDir 8 -d
20160433>list
/home/minjeong/Desktop/#P3/code/testDir/aa.c       8   -d
/home/minjeong/Desktop/#P3/code/testDir/abcTest/makefile_190530205910_190531001640       8   -d
/home/minjeong/Desktop/#P3/code/testDir/abcTest/makefile_190530205910_190531001645       8   -d
/home/minjeong/Desktop/#P3/code/testDir/abcTest/makefile_190531001645       8   -d
/home/minjeong/Desktop/#P3/code/testDir/abcTest/recoverTest.c_190531001823       8   -d
/home/minjeong/Desktop/#P3/code/testDir/recoverTest.c       8   -d
20160433>
```

⑭ d옵션 수행할 때 FILENAME이 디렉토리가 아닐 경우

```
20160433>add a.c 5 -d
path is only Directory file
20160433>
20160433>list
20160433>
```

⑮ d옵션 수행할 때 FILENAME 디렉토리 안의 파일이 이미 백업리스트 안에 존재할 때

```
20160433>add testDir 8 -d
20160433>list
/home/minjeong/Desktop/#P3/code/testDir/recoverTest.c       5   -n
/home/minjeong/Desktop/#P3/code/testDir/aa.c       8   -d
/home/minjeong/Desktop/#P3/code/testDir/abcTest/makefile_190530205910_190531001640       8   -d
/home/minjeong/Desktop/#P3/code/testDir/abcTest/makefile_190530205910_190531001645       8   -d
/home/minjeong/Desktop/#P3/code/testDir/abcTest/makefile_190531001645       8   -d
/home/minjeong/Desktop/#P3/code/testDir/abcTest/recoverTest.c_190531001823       8   -d
20160433>remove -a
20160433>list
20160433>add testDir 7 -d
20160433>list
/home/minjeong/Desktop/#P3/code/testDir/aa.c       7   -d
/home/minjeong/Desktop/#P3/code/testDir/abcTest/makefile_190530205910_190531001640       7   -d
/home/minjeong/Desktop/#P3/code/testDir/abcTest/makefile_190530205910_190531001645       7   -d
/home/minjeong/Desktop/#P3/code/testDir/abcTest/makefile_190531001645       7   -d
/home/minjeong/Desktop/#P3/code/testDir/abcTest/recoverTest.c_190531001823       7   -d
/home/minjeong/Desktop/#P3/code/testDir/recoverTest.c       7   -d
20160433>
```

⑯ m,n,t,d,옵션 모두 사용했을 때

```
20160433>add testDir 10 -n 5 -t 100 -d -m
20160433>list
/home/minjeong/Desktop/#P3/code/testDir/aa.c      10   -n -t -m -d
/home/minjeong/Desktop/#P3/code/testDir/abcTest/makefile_190530205910_190531001640      10   -n -t -m -d
/home/minjeong/Desktop/#P3/code/testDir/abcTest/makefile_190530205910_190531001645      10   -n -t -m -d
/home/minjeong/Desktop/#P3/code/testDir/abcTest/makefile_190531001645      10   -n -t -m -d
/home/minjeong/Desktop/#P3/code/testDir/abcTest/recoverTest.c_190531001823      10   -n -t -m -d
/home/minjeong/Desktop/#P3/code/testDir/recoverTest.c      10   -n -t -m -d
20160433>
```

⑰ 백업해야할 파일의 절대경로에 한글이 포함됐을 때

```
20160433>ls
aaa.c                      a.c        한글테스트.c  singlelinkedlist.h  ssu_backup.c  test2.c
abcdefghijklmnopqrstuvwxyz  backupDir  makefile      ssu_backup          t1.c          testBackup
20160433>add 한글테스트.c
can't use Hangul for ◆
20160433>
```

⑱ 백업해야할 파일이름이 255byte를 넘었을 때

```
20160433>ls
aaa.c                      a.c        한글테스트.c  singlelinkedlist.h  ssu_backup.c  test2.c
abcdefghijklmnopqrstuvwxyz  backupDir  makefile      ssu_backup          t1.c          testBackup
20160433>add /home/minjeong/lsp3/abcdefghijklmnopqrstuvwxyz/a1a2a3a4a5a6a7a8a9a10/abcdefghijklmnopqrstuvwxyz/b1b
2b3b4b5b6b7b8b9b10/applebananabananabanana/aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa50/abcdefghijklmnop
qrstuvwxyz/yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy50/linkedlist.h 5
filename is over 255byte
20160433>
```

4) remove명령어

① FILENAME 입력 없을 때

```
20160433>remove
usage : remove <FILENAME> [OPTION]
20160433>
```

② 백업을 중단할 파일이 백업리스트에 존재하지 않을 때

```
20160433>list
/home/minjeong/lsp3/aaa.c       6   -t
20160433>remove a.c
filename is not existed in backuplist
20160433>
```

③ 옵션이 없을 때

```
20160433>list
/home/minjeong/lsp3/a.c       8   -m -n
/home/minjeong/lsp3/aaa.c       6   -t
20160433>remove a.c
20160433>list
/home/minjeong/lsp3/aaa.c       6   -t
20160433>
```

④ a옵션

```
20160433>list
/home/minjeong/lsp3/inBackup.c       5   -m
/home/minjeong/lsp3/a.c       8   -n
/home/minjeong/lsp3/aaa.c      10   -t
20160433>remove -a
20160433>list
20160433>
```

⑤ a옵션 입력할 때 FILENAME을 입력한 경우

```
20160433>remove a.c -a
usage: remove -a
20160433>
```

5) compare

① FILENAME이 존재하지 않은 파일일 경우

```
20160433>ls
aaa.c  backupDir     inBackup.c  singlelinkedlist.h  ssu_backup.c  test2.c
a.c    한글테스트.c  makefile    ssu_backup          t1.c          testBackup
20160433>compare abc.c a.c
usage : <FILENAME1> is not exist
20160433>compare a.c abc.c
usage : <FILENAME2> is not exist
20160433>
```

② FILENAME이 일반파일이 아닐 경우

```
20160433>compare a.c backupDir
<FILENAME2> is only Regular file
20160433>compare backupDir a.c
<FILENAME1> is only Regular file
20160433>
```

③ 입력인자가 2개가 아닐 때

```
20160433>compare
usage : compare <FILENAME1> <FILENAME2>
20160433>compare a.c
usage : compare <FILENAME1> <FILENAME2>
20160433>compare a.c makefile t1.c
usage : compare <FILENAME1> <FILENAME2>
20160433>
```

④ 동일한 두 파일을 입력해서 compare명령어 사용할 때

```
20160433>compare inBackup.c inBackup.c
inBackup.c와 inBackup.c는 동일한 파일입니다.
20160433>compare inBackup.c /home/minjeong/lsp3/inBackup.c
inBackup.c와 inBackup.c는 동일한 파일입니다.
```

```
20160433>compare /home/minjeong/lsp3/inBackup.c /home/minjeong/lsp3/inBackup.c
inBackup.c와 inBackup.c는 동일한 파일입니다.
```

⑤ 동일하지 않은 두 파일 입력해서 compare명령어 사용할 때

```
20160433>compare a.c makefile
a.c 1558345344 286
makefile 1559064850 70
```

```
20160433>compare /home/minjeong/lsp3/inBackup.c /home/minjeong/lsp3/a.c
inBackup.c 1559226867 64
a.c 1559235349 286
```

6) recover

① 변경할 파일이 존재하지 않을 때

```
20160433>ls
aaa.c  backupDir     inBackup.c  singlelinkedlist.h  ssu_backup.c  test2.c
a.c    한글테스트.c  makefile    ssu_backup          t1.c          testBackup
20160433>recover abc.c
usage : <FILENAME> is not exist
20160433>
```

② 변경할 파일에 대한 백업파일이 존재하지 않을 때

```
20160433>ls
aaa.c  backupDir     inBackup.c  singlelinkedlist.h  ssu_backup.c  test2.c
a.c    한글테스트.c  makefile    ssu_backup          t1.c          testBackup
20160433>ls backupDir
aaa.c_190531004305       inBackup.c_190531004246  makefile_190531004841  makefile_190531004906
aaa.c_190531004311       log.txt                  makefile_190531004846  makefile_190531004911
aaa.c_190531004317       makefile_190531004826    makefile_190531004851  makefile_190531004916
aaa.c_190531004323       makefile_190531004831    makefile_190531004856  makefile_190531004921
inBackup.c_190531004239  makefile_190531004836    makefile_190531004901
20160433>recover t1.c
복구할 수 있는 백업파일이 존재하지 않습니다.
20160433>
```

③ 변경할 파일이 현재 백업리스트에 존재하지 않을 경우

```
20160433>ls backupDir
aaa.c_190531010843  a.c_190531010718  log.txt            t1.c_190531010557  test2.c_190531010508
a.c_190531010713    a.c_190531010723  t1.c_190531010549  t1.c_190531010605  test2.c_190531010518
20160433>list
/home/minjeong/lsp3/aaa.c       5
20160433>recover a.c
0.   exit
1.   190531010713       286bytes
2.   190531010718       286bytes
3.   190531010723       286bytes
Choose file to recover : 1
Recovery success

1    #include <stdio.h>
2    #include <stdlib.h>
3    #include "linkedlist.h"
4
5    int main(void){
6       List list;
7       Initialisze(&list);
8       Member x,y,z;
9       x={"aaa",3,'c'};
10      InsertFront(&list, &x);
11      y={"bbb",5,'a'};
12      InsertFront(&list, &y);
13      z={"ccc",8,'m'};
14      InsertFront(&list, &z);
15
16      Print(&list);
17
18      exit(0);
19   }

20160433>
```

④ 변경할 파일이 현재 백업리스트에 존재할 경우

```
20160433>list
/home/minjeong/lsp3/a.c       5   -n
/home/minjeong/lsp3/t1.c       8
20160433>ls backupDir
a.c_190531010543  a.c_190531010553  t1.c_190531010549  test2.c_190531010508
a.c_190531010548  log.txt           t1.c_190531010557  test2.c_190531010518
20160433>recover t1.c
0.   exit
1.   190531010549       440bytes
2.   190531010557       440bytes
3.   190531010605       440bytes
Choose file to recover : 2
Recovery success

1    #include "singlelinkedlist.h"
2    #include <ctype.h>
3    #include <unistd.h>
4    LinkedList list;
5
6    int main(void){
7       Init(&list);
8       printf("현재 노드 개수 : %d\n",getSize(&list));
9
10      InsertNode(&list, "abc.txt",3," ");
11      InsertNode(&list, "rrrr.txt", 4, "-m");
12      print(&list);
13
14      printf("현재 노드 개수 : %d\n",getSize(&list));
15
16      printf("현재 노드 개수 : %d\n",getSize(&list));
17
18      char aa[100];
19      getcwd(aa,200);
20
21      print(&list);
22   exit(0);
23   }
24

20160433>list
/home/minjeong/lsp3/a.c       5   -n
20160433>
```

⑤ n옵션

```
20160433>ls
aaa.c  backupDir     inBackup.c  singlelinkedlist.h  ssu_backup.c  test2.c
a.c    한글테스트.c  makefile    ssu_backup          t1.c          testBackup
20160433>ls backupDir
aaa.c_190531010843  aaa.c_190531010858  a.c_190531010718  t1.c_190531010549  test2.c_190531010508
aaa.c_190531010848  aaa.c_190531010903  a.c_190531010723  t1.c_190531010557  test2.c_190531010518
aaa.c_190531010853  a.c_190531010713    log.txt           t1.c_190531010605
20160433>recover aaa.c -n abc.c
0.   exit
1.   190531010843       156bytes
2.   190531010848       156bytes
3.   190531010853       156bytes
4.   190531010858       156bytes
5.   190531010903       156bytes
Choose file to recover : 5
Recovery success

1    #include <stdio.h>
2
3    int main(void){
4       char a[5]="\n";
5       printf("%d\n",a[0]);
6       if(a[0]=='\n')
7               printf("888\n");
8       if(a[0]==10)
9               printf("&&&\n");
10      return 0;
11   }

20160433>ls
aaa.c  a.c        한글테스트.c  makefile            ssu_backup    t1.c     testBackup
abc.c  backupDir  inBackup.c    singlelinkedlist.h  ssu_backup.c  test2.c
20160433>
```

⑥ n옵션 수행시 NEWFILE 입력 없을 때

```
20160433>recover a.c -n
usage: recover <FILENAME> [-n <NEWFILE>]
20160433>
```

⑦ n옵션 수행시 NEWFILE이 이미 존재할 때

```
20160433>ls
aaa.c  a.c        한글테스트.c  makefile            ssu_backup    t1.c     testBackup
abc.c  backupDir  inBackup.c    singlelinkedlist.h  ssu_backup.c  test2.c
20160433>recover a.c -n abc.c
usage : <NEWFILE> is existed
20160433>
```

⑧ 리스트에서 파일을 선택하지 않는 경우

```
20160433>recover a.c
0.   exit
1.   190531010713       286bytes
2.   190531010718       286bytes
3.   190531010723       286bytes
Choose file to recover : 0
20160433>
```

7) list 명령어

```
20160433>add test.c 9 -n 4 -t 70
20160433>list
/home/minjeong/Desktop/#P3/code/aa.c       6   -t
/home/minjeong/Desktop/#P3/code/abc.txt      10   -n
/home/minjeong/Desktop/#P3/code/testDir/aa.c       5   -m -d
/home/minjeong/Desktop/#P3/code/testDir/abcTest/makefile_190530205910_190531001640       5   -m -d
/home/minjeong/Desktop/#P3/code/testDir/abcTest/makefile_190530205910_190531001645       5   -m -d
/home/minjeong/Desktop/#P3/code/testDir/abcTest/makefile_190531001645       5   -m -d
/home/minjeong/Desktop/#P3/code/testDir/abcTest/recoverTest.c_190531001823       5   -m -d
/home/minjeong/Desktop/#P3/code/testDir/recoverTest.c       5   -m -d
/home/minjeong/Desktop/#P3/code/inBackup.c       8   -t -m -n
/home/minjeong/Desktop/#P3/code/test.c       9   -n -t
20160433>
```

8) ls 명령어

```
minjeong@minjeong-VirtualBox:~/lsp3$ ./ssu_backup
20160433>ls
aaa.c  backupDir     makefile            ssu_backup    t1.c     testBackup
a.c    한글테스트.c  singlelinkedlist.h  ssu_backup.c  test2.c
20160433>ls testBackup
backupDir
20160433>exit
```

9) vi(m) 명령어

```
20160433>vim
20160433>vi
20160433>vi inBackup.c
20160433>ls
aaa.c  backupDir     inBackup.c  singlelinkedlist.h  ssu_backup.c  test2.c
a.c    한글테스트.c  makefile    ssu_backup          t1.c          testBackup
```

10) exit 명령어

```
minjeong@minjeong-VirtualBox:~/lsp3$ ./ssu_backup
20160433>exit
minjeong@minjeong-VirtualBox:~/lsp3$
```

11) 로그파일

```
minjeong@minjeong-VirtualBox:~/lsp3$ cat ./backupDir/log.txt
[190531 015447] /home/minjeong/lsp3/inBackup.c added
[190531 015452] /home/minjeong/lsp3/backupDir/inBackup.c_190531015452 generated
[190531 015457] /home/minjeong/lsp3/backupDir/inBackup.c_190531015457 generated
[190531 015500] /home/minjeong/lsp3/a.c added
[190531 015502] /home/minjeong/lsp3/backupDir/inBackup.c_190531015502 generated
[190531 015507] /home/minjeong/lsp3/backupDir/inBackup.c_190531015507 generated
[190531 015510] /home/minjeong/lsp3/backupDir/a.c_190531015510 generated
[190531 015512] /home/minjeong/lsp3/backupDir/inBackup.c_190531015512 generated
[190531 015517] /home/minjeong/lsp3/backupDir/inBackup.c_190531015517 generated
[190531 015520] /home/minjeong/lsp3/backupDir/a.c_190531015520 generated
[190531 015522] /home/minjeong/lsp3/backupDir/inBackup.c_190531015522 generated
[190531 015523] /home/minjeong/lsp3/a.c deleted
[190531 015527] /home/minjeong/lsp3/backupDir/inBackup.c_190531015527 generated
[190531 015530] /home/minjeong/lsp3/inBackup.c deleted
[190531 015549] /home/minjeong/lsp3/a.c recovered
```

12) ssu_backup 프로그램을 다시 실행할 경우 이전에 실행했던 백업 파일 리스트 다시 불러오기

```
20160433>list
/home/minjeong/lsp3/inBackup.c        6    -n
/home/minjeong/lsp3/abc.c       8
20160433>exit
minjeong@minjeong-VirtualBox:~/lsp3$ ./ssu_backup
20160433>list
/home/minjeong/lsp3/inBackup.c        6    -n
/home/minjeong/lsp3/abc.c       8
20160433>exit
minjeong@minjeong-VirtualBox:~/lsp3$
```

5. 소스코드

<siglelinkedlist.h>

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <pthread.h>

typedef struct Node{
        char file[256];
        int period;
        char opt[100];
        char temp_d[7];
        char temp_t[7];
        char filename[100];
        int mtime_record;
        int list_m_flag;
        int list_t_flag;
        int addTime;
        int list_n_flag;
        int number;
        pthread_t tid;
        struct Node *next;
}Node;

typedef struct{
        Node *head; //리스트의 첫 항목을 가리키는 포인터 변수
        int size; //노드의 개수
} LinkedList;

//싱글링크드리스트를 초기화한다.
void Init(LinkedList *list){
        list->head = NULL;
        list->size = 0;
}

//싱글링크드리스트의 노드개수를 반환한다.
int getSize(LinkedList *list){
        return list->size;
}

//새로운 노드를 생성한다
Node* create(char *file, int period, char *opt,char *filename,int mtime,int list_m_flag,int list_t_flag,int
addTime,int list_n_flag, int number){
```

```
	Node *newNode = (Node*)malloc(sizeof(Node));
	if(newNode == NULL)
	{
		printf("Error: creating a new node.\n");
		exit(1);
	}
	strcpy(newNode->file,file);
	newNode->period = period;
	strcpy(newNode->opt , opt);
	strcpy(newNode->filename, filename);
	newNode->mtime_record = mtime;
	newNode->list_m_flag = list_m_flag;
	newNode->list_t_flag=list_t_flag;
	newNode->addTime = addTime;
	newNode->list_n_flag = list_n_flag;
	newNode->number = number;
	return newNode;
}

//특정 위치의 노드 뒤에 새로운 노드를 추가하는 함수이다.
void addPos(LinkedList *list, int pos, char *file, int period, char *opt, char *filename,int mtime, int list_m_flag,int list_t_flag,int addTime, int list_n_flag, int number){
	if(pos > (list->size)+1 || pos < 1){
		printf("Error: Position is out of range!\n");
		exit(1);
	}
	else{
		Node* newNode = create(file, period, opt, filename,mtime,list_m_flag,list_t_flag,addTime, list_n_flag, number);

		if(pos == 1){
			newNode->next = list->head;
			list->head = newNode;
		}
		else{
			Node *temp = list -> head;
			for(int i=1;i<pos-1;i++)
				temp = temp->next;

			newNode->next = temp->next;
			temp->next = newNode;
		}
		(list->size)++;
	}
```

```
}

//리스트의 가장 뒤에 노드를 추가한다.
void InsertNode(LinkedList *list, char *file, int period, char *opt,char *filename,int mtime,int list_m_flag,int list_t_flag,int addTime, int list_n_flag, int number){

	addPos(list, (list->size)+1,file, period, opt,filename,mtime,list_m_flag,list_t_flag,addTime,list_n_flag, number);
}

//리스트에서 원하는 항목 1개를 찾아서 삭제한다.
void delNode(LinkedList *list, char*file){
	Node* cursor = list->head;
	Node* prev = NULL;
	if(!strcmp(cursor->file, file)){
		list->head = cursor->next;
		free(cursor);
		(list->size)--;
	}
	else{
		while(cursor != NULL){
			if(!strcmp(cursor->file, file)) break;
			prev = cursor;
			cursor = cursor->next;
		}
		if(cursor != NULL){
			prev->next = cursor->next;
			free(cursor);
			(list->size)--;
		}
	}
}

//싱글링크드리스트의 모든 노드들의 메모리를 해제한다.
void delAllNode(LinkedList *list){
	Node *temp;
	Node *cursor = list->head;
	list->head = NULL;

	while(cursor != NULL){
		temp = cursor->next;
		free(cursor);
		cursor = temp;
	}
```

```
	list->size = 0;
}

//싱글링크드리스트가 빈 리스트인지 검사하는 함수이다.
bool isEmpty(LinkedList *list){
	return (list->head == NULL);
}

//싱글링크드리스트 안에 file과 같은 이름을 가진 노드가 존재하는지 여부를 판단하는 함수이다.
bool isInList(LinkedList *list, char*file){
	Node *cursor = list->head;
	while(cursor != NULL){
		if(!strcmp(cursor->file, file)) return true;
		cursor = cursor->next;
	}
	return false;
}

//싱글링크드리스트의 모든 리스트내용을 출력하는 함수이다.
void print(LinkedList *list){
	Node *current = list->head;
	while(current != NULL){
		printf("%s %7d  %-15s\n",current->file,current->period,current->opt);
		current = current->next;
	}
}

//리스트에서 원하는 항목 1개를 찾아서 반환한다.
Node* searchNode(LinkedList *list, char*file){
	Node* cursor = list->head;
	if(!strcmp(cursor->file, file)){
		return cursor;
	}
	else{
		while(cursor != NULL){
			if(!strcmp(cursor->file, file)) break;
			cursor = cursor->next;
		}
		if(cursor != NULL){
			return cursor;
		}
	}
	return NULL;
}
```

<ssu_backup.c>

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <dirent.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <pthread.h>
#include <ctype.h>
#include "singlelinkedlist.h“

void command_add(char* command); //명령어 add를 실행시키는 함수
void command_remove(char* command); //명령어 remove를 실행시키는 함수
void command_compare(char* command); //명령어 compare를 실행시키는 함수
void command_recover(char* command); //명령어 recover를 실행시키는 함수
void command_list(); //명령어 list를 실행시키는 함수
void printLog(char *name, int index); // 로그입력함수
void *backupFile(void *arg);//쓰레드에서 백업시키는 함수
void optionAddN(char *filename,int number); //add -n옵션을 수행하는 함수
void optionAddD(char *filename,int period,char* option);//add -d옵션을 수행하는 함수
void recoverFileRead(char* fname);//백업한 파일로 변경 후 변경된 파일의 내용을 출력
void makeTable();//리스트 내용을 저장해주는 함수
void readTable();//리스트 내용을 읽어오는 함수

char dirname[PATH_MAX];//백업파일을 저장할 경로
int checkD=0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER; //뮤텍스 변수를 정적으로 선언
LinkedList list;

#define MODE R_OK|X_OK|W_OK
#define MAX_SIZE 1024

int main(int argc, char* argv[]){
        struct timeval begin_t, end_t;
        char command[MAX_SIZE];
        char b_directory[MAX_SIZE];
        char pathname[MAX_SIZE];
        Node *temp;
        char csvfile[MAX_SIZE];
        char *ddummy;
        char tempname[MAX_SIZE];
        char pname[MAX_SIZE];
        Init(&list);
```

```
//인자가 2개 이상이면 usage 출력후 프로그램 종료
if(argc>2){
	fprintf(stderr, "usage: ./ssu_backup <directory>\n");
	exit(1);
}

if(argc==1){
	//current working 디렉토리 밑에 백업 디렉토리를 생성한다.
	getcwd(pathname,MAX_SIZE);
	if(access("backupDir",F_OK)){
		mkdir("backupDir",0777);
	}
	sprintf(dirname,"%s/backupDir",pathname);
}

else{
	//절대경로, 상대경로 상관없이 경로인식
	if(argv[1][0]=='/')
		sprintf(pathname,"%s",argv[1]);
	else if(argv[1][0]=='.'){
		strcpy(tempname,argv[1]);
		ddummy=tempname+2;
		strcpy(tempname,ddummy);
		getcwd(pname,MAX_SIZE);
		sprintf(pathname,"%s/%s",pname,tempname);
	}
	else{
		getcwd(pname,MAX_SIZE);
		sprintf(pathname,"%s/%s",pname,argv[1]);
	}

	//인자로 입력받은 디렉토리를 찾을 수 없으면 usage 출력 후 프로그램 종료한다.
	if(access(pathname, F_OK)){
		fprintf(stderr, "usage : directory is not exist\n");
		exit(1);
	}

	//인자로 입력받은 디렉토리가 디렉토리파일이 아니라면 usage 출력후 프로그램 종료한다.
	struct stat st;
	stat(pathname, &st);
	if(!S_ISDIR(st.st_mode)){
		fprintf(stderr, "usage : No directory\n");
		exit(1);
```

```
	}

	//인자로 입력받은 디렉토리의 접근권한이 없는 경우 usage 출력 후 프로그램 종료한다.
	if(access(pathname,MODE)){
		fprintf(stderr, "usage : directory에 접근권한이 없습니다.\n");
		exit(1);
	}
	//지정한 경로에 백업 디렉토리를 생성한다.
	sprintf(dirname,"%s/backupDir",pathname);
	if(access(dirname,F_OK)){
		mkdir(dirname,0777);
	}
}

//백업디렉토리 안에 리스트 정보를 저장한 파일이 있는지 확인한다
//있다면 파일을 읽어와 링크드리스트를 복구한다.
sprintf(csvfile,"%s/listLog.csv",dirname);
if(!access(csvfile,F_OK)){//csv파일 있으면
	readTable();
}
//프롬프트 출력
while(1){
	printf("20160433>");
	strcpy(command,"");
	scanf("%[^\n]s",command);
	getc(stdin);
	if(command[0]==0)
		continue;
	if(!strcmp(command,"exit")){
		//현재 실행중인 모든 백업을 중지하고 프로그램을 종료한다.
		temp=list.head;
		for(int i=0;i<list.size;i++){
			if(i!=0){
				temp=temp->next;
			}
			pthread_cancel(temp->tid);
		}

		pthread_mutex_destroy(&mutex);
		//exit 직전 남아있는 리스트의 정보를 따로 저장한다.
		makeTable();
		delAllNode(&list);
		break;
	}
```

```
if(strstr(command," ")==NULL){
        if(!strcmp(command,"list")){
                command_list();
        }
        else if(!strcmp(command,"ls")){
                system(command);
        }
        else if(!strcmp(command,"vi")){
                system(command);
        }
        else if(!strcmp(command,"vim")){
                system(command);
        }
        else if(!strcmp(command, "add")){
                fprintf(stderr, "usage : add <FILENAME> [PERIOD] [OPTION]\n");
                continue;
        }
        else if(!strcmp(command, "remove")){
                fprintf(stderr, "usage : remove <FILENAME> [OPTION]\n");
                continue;
        }
        else if(!strcmp(command, "compare")){
                fprintf(stderr, "usage : compare <FILENAME1> <FILENAME2>\n");
                continue;
        }
        else if(!strcmp(command, "recover")){
                fprintf(stderr, "usage : recover <FILENAME> [OPTION]\n");
                continue;
        }
        else{
                fprintf(stderr, "존재하지 않는 명령어입니다.\n");
                continue;
        }
}
else{
        char dummy_command1[1024];
        char* dummy_command2;
        strcpy(dummy_command1,command);
        dummy_command2 = strtok(dummy_command1," ");
        if(!strcmp(dummy_command1,"add")){
                command_add(command);
        }
        else if(!strcmp(dummy_command1,"remove")){
                command_remove(command);
```

```
                        }
                        else if(!strcmp(dummy_command1,"compare")){
                                command_compare(command);
                        }
                        else if(!strcmp(dummy_command1,"recover")){
                                command_recover(command);
                        }
                        else if(!strcmp(dummy_command1,"vi")){
                                system(command);
                        }
                        else if(!strcmp(dummy_command1,"vim")){
                                system(command);
                        }
                        else if(!strcmp(dummy_command1,"ls")){
                                system(command);
                        }
                        else{
                                fprintf(stderr, "존재하지 않는 명령어입니다.\n");
                                continue;
                        }
                }
        }
        exit(0);
}

//명령어 add를 실행시키는 함수이다.
void command_add(char* command){
        pthread_t tid;
        int add_m_flag=0;
        int add_n_flag=0;
        int add_t_flag=0;
        int add_d_flag=0;
        char dummy_command1[1024];
        char filename[MAX_SIZE];
        char pathname[MAX_SIZE];
        int period;
        char option[100]=" ";
        char optionPrint[100]=" ";
        int number;
        int addTime;
        double temp;
        double temp2;
        char tempName[100];
        struct stat s;
```

```
char *dum2;
char dum[300];
char *token;
char *ptr[2];
char *token_data[4];
char per[5];
char data[5]="";
char tsec[5];
int k=0;
int check_f=0;
int check_f2=0;
int check_count=0;
char *ddummy;
strcpy(dummy_command1,command);
token =strtok_r(dummy_command1," ",&ptr[0]);
token_data[0] =strtok_r(ptr[0]," ",&ptr[1]);
token_data[1]=strtok_r(ptr[1]," ",&ptr[0]);
token_data[2]=strtok_r(ptr[0]," ",&ptr[1]);
if(token_data[2] != NULL){
        sprintf(token_data[2],"%s %s",token_data[2],ptr[1]);
}
strcpy(tempName, token_data[0]);

//절대경로로 변환
if(tempName[0]=='.'){
        ddummy=token_data[0]+2;
        strcpy(token_data[0],ddummy);
        strcpy(dum,token_data[0]);
        dum2 = strtok(dum,"/");
        while(dum2!=NULL){
                if((strstr(dum2,"."))!=NULL){
                        check_f=1;
                        check_f2=1;
                        break;
                }
                dum2=strtok(NULL,"/");
        }

        if(!check_f){
                strcpy(dum,token_data[0]);
                dum2 = strtok(dum,"/");
                while(dum2!=NULL){
                        if((strstr(dum2,"_"))!=NULL){
                                check_f=0;
```

```
                                check_f2=1;
                                break;
                        }
                        dum2=strtok(NULL,"/");
                }
        }

        if(!check_f2){
                strcpy(dum,token_data[0]);
                dum2 = strtok(dum,"/");
                while(dum2!=NULL){
                        check_count++;
                        dum2=strtok(NULL,"/");
                }
                strcpy(dum,token_data[0]);
                dum2 = strtok(dum,"/");
                check_count--;
                while(dum2!=NULL){
                        if(check_count==0){
                                check_f2=0;
                                break;
                        }
                        check_count--;
                        dum2=strtok(NULL,"/");
                }
        }

        strcpy(tempName,dum2);
        getcwd(pathname,MAX_SIZE);
        sprintf(filename,"%s/%s",pathname,token_data[0]);
}

else if(tempName[0]!='/'){
        getcwd(pathname,MAX_SIZE);
        sprintf(filename,"%s/%s",pathname,tempName);
}

else{
        strcpy(dum,token_data[0]);
        dum2 = strtok(dum,"/");
        while(dum2!=NULL){
                if((strstr(dum2,"."))!=NULL){
                        check_f=1;
                        check_f2=1;
```

```
                        break;
                }
                dum2=strtok(NULL,"/");
        }

        if(!check_f){
                strcpy(dum,token_data[0]);
                dum2 = strtok(dum,"/");
                while(dum2!=NULL){
                        if((strstr(dum2,"_"))!=NULL){
                                check_f=0;
                                check_f2=1;
                                break;
                        }
                        dum2=strtok(NULL,"/");
                }
        }

        if(!check_f2){
                strcpy(dum,token_data[0]);
                dum2 = strtok(dum,"/");
                while(dum2!=NULL){
                        check_count++;
                        dum2=strtok(NULL,"/");
                }
                strcpy(dum,token_data[0]);
                dum2 = strtok(dum,"/");
                check_count--;
                while(dum2!=NULL){
                        if(check_count==0){
                                check_f2=0;
                                break;
                        }
                        check_count--;
                        dum2=strtok(NULL,"/");
                }
        }

        strcpy(tempName,dum2);
        strcpy(filename,token_data[0]);
}

//한글이 포함되면 에러처리 후 프롬프트로 제어가 넘어간다.
for(int i=0;i<strlen(filename);i++){
```

```
        if(filename[i]>=33 && filename[i]<=122){
                continue;
        }
        else{
                fprintf(stderr, "can't use Hangul for %c\n",filename[i]);
                return;
        }
}

//filename이 255byte를 넘을 시 에러처리 후 프롬프트로 제어가 넘어간다
if(strlen(filename)>255){
        fprintf(stderr, "filename is over 255byte\n");
        return;
}

//파일이 존재하지 않을 시 에러처리 후 프롬프트로 제어가 넘어간다.
if(access(filename, F_OK)){
        fprintf(stderr, "usage : file is not exist\n");
        return;
}

add_m_flag=0;
add_t_flag=0;
add_n_flag = 0;

//이제 period가 있는지를 확인한다
if(token_data[1]==NULL){ //FileNAME만 존재
        fprintf(stderr, "usage: <FILENAME> [period] [option]\n");
        return;
}

else{
        if(!isdigit(token_data[1][0])){ //option만 존재!
                fprintf(stderr, "usage: <FILENAME> [period] [option]\n");
                return;
        }
        else{ //period 존재!
                strcpy(per,token_data[1]);
                temp = atof(per);
                if((temp - (int)temp)!=0){
                        fprintf(stderr, "period는 정수형이어야합니다\n");
                        return;
                }
                else{
```

```
period = atoi(per);
if(period<5 || period>10){
        fprintf(stderr, "5 <= period <= 10\n");
        return;
}
if(token_data[2] == NULL){//옵션 없음!
        strcpy(option , " ");
        strcpy(optionPrint," ");
}
else{
        strcpy(option," ");
        strcpy(optionPrint," ");
        if(token_data[2][0]!='-'){
                fprintf(stderr,"usage: <FILENAME> [period] [option]\n");
                return;
        }
        for(int i=0;i<strlen(token_data[2]);i++){
                if(token_data[2][i]=='-'){
                        i++;
                        if(token_data[2][i]=='m'){
                                strcat(option,"-m ");
                                strcat(optionPrint,"-m ");
                                add_m_flag=1;
                        }
                        else if(token_data[2][i]=='n'){
                                strcat(option, "-n ");
                                strcat(optionPrint, "-n ");
                                add_n_flag=1;
                                if(token_data[2][i+1]=='\n'){//NUMBER입력 없음
                                        fprintf(stderr, "usage : -n NUMBER\n");
                                        return;
                                }
                                if(token_data[2][i+1]!=' '){//NUMBER입력 없음
                                        fprintf(stderr, "usage : -n NUMBER\n");
                                        return;
                                }
                                if(!isdigit(token_data[2][i+2])){
                                        fprintf(stderr, "usage : -n NUMBER\n");
                                        return;
                                }
                                i=i+2;
                                for(k=i+1;k<strlen(token_data[2]);k++){
                                        if(token_data[2][k]==' '){
                                                break;
```

```
                }
            }
            strcpy(data,&token_data[2][i]);
            data[k-i]='\0';
            i=k-1;
            temp2 = atof(data);
            if((temp2 - (int)temp2)!=0){
                fprintf(stderr, "NUMBER는 정수형을 입력해
야합니다.\n");

                return;
            }
            else{
                number = atoi(data);
                if(number<1 || number>100){
                    fprintf(stderr, "1 <= NUMBER <=
100\n");

                    return;
                }
                strcat(option, data);
                strcat(option," ");
            }
        }
        else if(token_data[2][i]=='t'){
            strcat(option,"-t ");
            strcat(optionPrint,"-t ");
            add_t_flag=1;
            if(token_data[2][i+1]=='\n'){//TIME 입력 없음
                fprintf(stderr, "usage : -t TIME\n");
                return;
            }
            if(token_data[2][i+1]!=' '){//TIME입력 없음
                fprintf(stderr, "usage: -t TIME\n");
                return;
            }
            if(!isdigit(token_data[2][i+2])){
                fprintf(stderr, "usage : -t TIME\n");
                return;
            }
            i=i+2;
            for(k=i+1;k<strlen(token_data[2]);k++){
                if(token_data[2][k]==' '){
                    break;
                }
            }
```

```
                                        strcpy(tsec,&token_data[2][i]);
                                        tsec[k-i]='\0';
                                        i=k-1;
                                        temp2 = atof(tsec);
                                        if((temp2 - (int)temp2)!=0){
                                                fprintf(stderr, "TIME은 정수형을 입력해야
합니다.\n");

                                                return;
                                        }
                                        else{
                                                addTime = atoi(tsec);
                                                if(addTime<60 || addTime>1200){
                                                        fprintf(stderr, "60 <= TIME <=
1200\n");

                                                        return;
                                                }
                                                strcat(option, tsec);
                                                strcat(option," ");
                                        }
                                }
                                else if(token_data[2][i]=='d'){
                                        add_d_flag=1;
                                }
                                else
                                        printf("존재하지 않는 옵션입니다.\n");
                                }
                        }
                }
        }
        }

        if(checkD){
                strcat(optionPrint,"-d ");
        }

        if(stat(filename,&s)<0){//stat 에러나면
                fprintf(stderr, "stat error\n");
                exit(1);
        }

        //백업해야할 파일이 백업리스트에 존재하는지 검사
        //존재한다면 에러처리 후 프롬프트로 제어가 넘어감
        if(isInList(&list,filename)){
```

```
                fprintf(stderr, "file is existed in backuplist\n");
                return;
        }

        if(!add_d_flag){
                //백업해야할 파일은 일반 파일만 가능하며 일반 파일이 아니라면 에러처리 후 프롬프트로 제어가 넘어감
                if(!S_ISREG(s.st_mode)){
                        fprintf(stderr, "path is only Regular file\n");
                        return;
                }
                InsertNode(&list,filename,period,optionPrint,tempName,s.st_mtime,add_m_flag,add_t_flag,addTime,
add_n_flag, number);
                printLog(filename,2);
                Node *tempNode;
                tempNode = (Node*)malloc(sizeof(Node*));
                tempNode= searchNode(&list,filename);
                //백업하는 쓰레드 생성하는 코드 추가
                if(pthread_create(&tid,NULL,backupFile,(void*)tempNode)!=0){
                        fprintf(stderr, "pthread_create error\n");
                        exit(1);
                }
        }

        if(add_d_flag){ //d옵션일 때는 filename이 디렉토리여야 한다.
                if(!S_ISDIR(s.st_mode)){
                        fprintf(stderr, "path is only Directory file\n");
                        return;
                }
                add_d_flag=0;
                checkD=1;
                optionAddD(filename,period,option);
                checkD=0;
        }
}

//명령어 remove를 실행하는 함수이다.
void command_remove(char* command){
        char dummy_command1[1024];
        char *dummy_command2;
        char *dummy_command3;
        char filename[MAX_SIZE];
        char pathname[MAX_SIZE];
        char tempName[100];
        Node *temp;
```

```
char dum[400];
char *dum2;
int check_f=0;
int check_f2=0;
int check_count=0;
char* ddummy;
strcpy(dummy_command1,command);
dummy_command2 = strtok(dummy_command1," ");
dummy_command3 = strtok(NULL," ");
if(dummy_command3[0]=='-'){
        if(!strcmp(dummy_command3,"-a")){
                //백업하는 쓰레드 모두 종료하는 코드 추가
                temp=list.head;
                for(int i=0;i<list.size;i++){
                        if(i!=0){
                                temp=temp->next;
                        }
                        pthread_cancel(temp->tid);
                        printLog(temp->file,3);
                }
                delAllNode(&list);
        }
        else{
                fprintf(stderr, "존재하지 않는 옵션입니다.\n");
                return;
        }
}

else{
        dummy_command2 = strtok(NULL," ");
        if(dummy_command2 !=NULL){
                if(!strcmp(dummy_command2,"-a")){
                        fprintf(stderr, "usage: remove -a\n");
                        return;
                }
                else{
                        fprintf(stderr, "usage : remove [FILENAME]\n");
                        return;
                }
        }
        else{
                strcpy(tempName, dummy_command3);
                //절대경로로 변환
                if(tempName[0]=='.'){
```

```
ddummy=dummy_command3+2;
strcpy(dummy_command3,ddummy);
strcpy(dum,dummy_command3);
dum2 = strtok(dum,"/");
while(dum2!=NULL){
        if((strstr(dum2,"."))!=NULL){
                check_f=1;
                check_f2=1;
                break;
        }
        dum2=strtok(NULL,"/");
}
if(!check_f){
        strcpy(dum,dummy_command3);
        dum2 = strtok(dum,"/");
        while(dum2!=NULL){
                if((strstr(dum2,"_"))!=NULL){
                        check_f=0;
                        check_f2=1;
                        break;
                }
                dum2=strtok(NULL,"/");
        }
}
if(!check_f2){
        strcpy(dum,dummy_command3);
        dum2 = strtok(dum,"/");
        while(dum2!=NULL){
                check_count++;
                dum2=strtok(NULL,"/");
        }
        strcpy(dum,dummy_command3);
        dum2 = strtok(dum,"/");
        check_count--;
        while(dum2!=NULL){
                if(check_count==0){
                        check_f2=0;
                        break;
                }
                check_count--;
                dum2=strtok(NULL,"/");
        }
}
strcpy(tempName,dum2);
```

```
		getcwd(pathname,MAX_SIZE);
		sprintf(filename,"%s/%s",pathname,dummy_command3);
}
else if(tempName[0]!='/'){
		getcwd(pathname,MAX_SIZE);
		sprintf(filename,"%s/%s",pathname,tempName);
}
else{
		strcpy(dum,dummy_command3);
		dum2 = strtok(dum,"/");
		while(dum2!=NULL){
			if((strstr(dum2,"."))!=NULL){
				check_f=1;
				check_f2=1;
				break;
			}
			dum2=strtok(NULL,"/");
		}
		if(!check_f){
			strcpy(dum,dummy_command3);
			dum2 = strtok(dum,"/");
			while(dum2!=NULL){
				if((strstr(dum2,"_"))!=NULL){
					check_f=0;
					check_f2=1;
					break;
				}
				dum2=strtok(NULL,"/");
			}
		}
		if(!check_f2){
			strcpy(dum,dummy_command3);
			dum2 = strtok(dum,"/");
			while(dum2!=NULL){
				check_count++;
				dum2=strtok(NULL,"/");
			}
			strcpy(dum,dummy_command3);
			dum2 = strtok(dum,"/");
			check_count--;
			while(dum2!=NULL){
				if(check_count==0){
					check_f2=0;
					break;
```

```
                                        }
                                        check_count--;
                                        dum2=strtok(NULL,"/");
                                }
                        }
                        strcpy(tempName,dum2);
                        strcpy(filename,dummy_command3);
                }

                //한글이 포함되면 에러처리 후 프롬프트로 제어가 넘어간다.
                for(int i=0;i<strlen(filename);i++){
                        if(filename[i]>=33 && filename[i]<=122){
                                continue;
                        }
                        else{
                                fprintf(stderr, "can't use Hangul\n");
                                return;
                        }
                }

                //filename이 255byte를 넘을 시 에러처리 후 프롬프트로 제어가 넘어감
                if(strlen(filename)>255){
                        fprintf(stderr, "filename is over 255byte\n");
                        return;
                }

                //제거하고싶은 파일이 백업리스트에 존재하는지 검사
                //존재하지않는다면 에러처리 후 프롬프트로 제어가 넘어감
                if(!isInList(&list,filename)){
                        fprintf(stderr, "filename is not existed in backuplist\n");
                        return;
                }

                else{
                        //백업하는 쓰레드 종료하는 코드 추가
                        pthread_cancel(searchNode(&list,filename)->tid);
                        printLog(filename,3);
                        delNode(&list,filename);
                }
        }
    }
}

//명령어 compare를 실행하는 함수이다.
```

```
void command_compare(char* command){
	char dummy_command1[1024];
	char *dummy_command2;
	char *dummy_command3;
	char *dummy_command4;
	char filename[MAX_SIZE];
	char pathname[MAX_SIZE];
	char tempName[100];
	struct stat s;
	char filename2[256];
	char pathname2[200];
	char tempName2[100];
	struct stat s2;
	char dum[400];
	char *dum2;
	int check_f=0;
	int check_f2=0;
	int check_count=0;
	int check_ff=0;
	int check_ff2=0;
	int check_count_f=0;
	char *ddummy;
	strcpy(dummy_command1,command);
	dummy_command2 = strtok(dummy_command1," ");
	dummy_command3 = strtok(NULL," ");
	dummy_command2 = strtok(NULL," ");
	if(dummy_command2 == NULL){ //입력인자가 2개가 아닐 경우 에러처리 후 프롬프트로 제어가 넘어감
		fprintf(stderr, "usage : compare <FILENAME1> <FILENAME2>\n");
		return;
	}
	dummy_command4 = strtok(NULL, " ");
	if(dummy_command4 != NULL){
		fprintf(stderr, "usage : compare <FILENAME1> <FILENAME2>\n");
		return;
	}

	strcpy(tempName, dummy_command3);
	strcpy(tempName2, dummy_command2);

	//절대경로로 변환
	if(tempName[0]=='.'){
		ddummy=dummy_command3+2;
		strcpy(dummy_command3,ddummy);
		strcpy(dum,dummy_command3);
```

```
	dum2 = strtok(dum,"/");
	while(dum2!=NULL){
		if((strstr(dum2,"."))!=NULL){
			check_f=1;
			check_f2=1;
			break;
		}
		dum2=strtok(NULL,"/");
	}
	if(!check_f){
		strcpy(dum,dummy_command3);
		dum2 = strtok(dum,"/");
		while(dum2!=NULL){
			if((strstr(dum2,"_"))!=NULL){
				check_f=0;
				check_f2=1;
				break;
			}
			dum2=strtok(NULL,"/");
		}
	}
	if(!check_f2){
		strcpy(dum,dummy_command3);
		dum2 = strtok(dum,"/");
		while(dum2!=NULL){
			check_count++;
			dum2=strtok(NULL,"/");
		}
		strcpy(dum,dummy_command3);
		dum2 = strtok(dum,"/");
		check_count--;
		while(dum2!=NULL){
			if(check_count==0){
				check_f2=0;
				break;
			}
			check_count--;
			dum2=strtok(NULL,"/");
		}
	}
	strcpy(tempName,dum2);
	getcwd(pathname,MAX_SIZE);
	sprintf(filename,"%s/%s",pathname,dummy_command3);
}
```

```
else  if(tempName[0]!='/'){
        getcwd(pathname,MAX_SIZE);
        sprintf(filename,"%s/%s",pathname,tempName);
}

else{
        strcpy(dum,dummy_command3);
        dum2 = strtok(dum,"/");
        while(dum2!=NULL){
                if((strstr(dum2,"."))!=NULL){
                        check_f=1;
                        check_f2=1;
                        break;
                }
                dum2=strtok(NULL,"/");
        }
        if(!check_f){
                strcpy(dum,dummy_command3);
                dum2 = strtok(dum,"/");
                while(dum2!=NULL){
                        if((strstr(dum2,"_"))!=NULL){
                                check_f=0;
                                check_f2=1;
                                break;
                        }
                        dum2=strtok(NULL,"/");
                }
        }
        if(!check_f2){
                strcpy(dum,dummy_command3);
                dum2 = strtok(dum,"/");
                while(dum2!=NULL){
                        check_count++;
                        dum2=strtok(NULL,"/");
                }
                strcpy(dum,dummy_command3);
                dum2 = strtok(dum,"/");
                check_count--;
                while(dum2!=NULL){
                        if(check_count==0){
                                check_f2=0;
                                break;
                        }
```

```
                        check_count--;
                        dum2=strtok(NULL,"/");
                }
        }
        strcpy(tempName,dum2);
        strcpy(filename,dummy_command3);
}

//한글이 포함되면 에러처리 후 프롬프트로 제어가 넘어간다.
for(int i=0;i<strlen(filename);i++){
        if(filename[i]>=33 && filename[i]<=122){
                continue;
        }
        else{
                fprintf(stderr, "<FILENAME1> can't use Hangul\n");
                return;
        }
}

//filename1이 255byte를 넘을 시 에러처리 후 프롬프트로 제어가 넘어간다
if(strlen(filename)>255){
        fprintf(stderr, "<FILENAME1> is over 255byte\n");
        return;
}

//파일이 존재하지 않을 시 에러처리 후 프롬프트로 제어가 넘어간다
if(access(filename, F_OK)){
        fprintf(stderr, "usage : <FILENAME1> is not exist\n");
        return;
}

if(stat(filename,&s)<0){//stat 에러나면
        fprintf(stderr, "stat error\n");
        exit(1);
}

//FILENAME1이 일반 파일이 아니라면 에러처리 후 프롬프트로 제어가 넘어간다
if(!S_ISREG(s.st_mode)){
        fprintf(stderr, "<FILENAME1> is only Regular file\n");
        return;
}

//절대경로로 변환한다.
if(tempName[0]=='.'){
```

```
ddummy=dummy_command2+2;
strcpy(dummy_command2,ddummy);
strcpy(dum,dummy_command2);
dum2 = strtok(dum,"/");
while(dum2!=NULL){
        if((strstr(dum2,"."))!=NULL){
                check_f=1;
                check_f2=1;
                break;
        }
        dum2=strtok(NULL,"/");
}
if(!check_f){
        strcpy(dum,dummy_command2);
        dum2 = strtok(dum,"/");
        while(dum2!=NULL){
                if((strstr(dum2,"_"))!=NULL){
                        check_f=0;
                        check_f2=1;
                        break;
                }
                dum2=strtok(NULL,"/");
        }
}
if(!check_f2){
        strcpy(dum,dummy_command2);
        dum2 = strtok(dum,"/");
        while(dum2!=NULL){
                check_count++;
                dum2=strtok(NULL,"/");
        }
        strcpy(dum,dummy_command2);
        dum2 = strtok(dum,"/");
        check_count--;
        while(dum2!=NULL){
                if(check_count==0){
                        check_f2=0;
                        break;
                }
                check_count--;
                dum2=strtok(NULL,"/");
        }
}
strcpy(tempName,dum2);
```

```
		getcwd(pathname,MAX_SIZE);
		sprintf(filename,"%s/%s",pathname,dummy_command2);
}
else if(tempName2[0]!='/'){
		getcwd(pathname2,200);
		sprintf(filename2,"%s/%s",pathname2,tempName2);
}
else{
		strcpy(dum,dummy_command2);
		dum2 = strtok(dum,"/");
		while(dum2!=NULL){
			if((strstr(dum2,"."))!=NULL){
				check_f=1;
				check_f2=1;
				break;
			}
			dum2=strtok(NULL,"/");
		}
		if(!check_f){
			strcpy(dum,dummy_command2);
			dum2 = strtok(dum,"/");
			while(dum2!=NULL){
				if((strstr(dum2,"_"))!=NULL){
					check_f=0;
					check_f2=1;
					break;
				}
				dum2=strtok(NULL,"/");
			}
		}
		if(!check_f2){
			strcpy(dum,dummy_command2);
			dum2 = strtok(dum,"/");
			while(dum2!=NULL){
				check_count++;
				dum2=strtok(NULL,"/");
			}
			strcpy(dum,dummy_command2);
			dum2 = strtok(dum,"/");
			check_count--;
			while(dum2!=NULL){
				if(check_count==0){
					check_f2=0;
					break;
```

```
                        }
                        check_count--;
                        dum2=strtok(NULL,"/");
                }
        }
        strcpy(tempName2,dum2);
        strcpy(filename2,dummy_command2);
}

//한글이 포함되면 에러처리 후 프롬프트로 제어가 넘어간다.
for(int i=0;i<strlen(filename2);i++){
        if(filename2[i]>=33 && filename2[i]<=122){
                continue;
        }
        else{
                fprintf(stderr, "<FILENAME2> can't use Hangul\n");
                return;
        }
}


//filename2가 255byte를 넘을 시 에러처리 후 프롬프트로 제어가 넘어간다.
if(strlen(filename2)>255){
        fprintf(stderr, "<FILENAME2> is over 255byte\n");
        return;
}

//파일이 존재하지 않을 시 에러처리 후 프롬프트로 제어가 넘어간다.
if(access(filename2, F_OK)){
        fprintf(stderr, "usage : <FILENAME2> is not exist\n");
        return;
}

if(stat(filename2,&s2)<0){//stat 에러나면
        fprintf(stderr, "stat error\n");
        exit(1);
}

//FILENAME이 일반 파일이 아니라면 에러처리 후 프롬프트로 제어가 넘어간다
if(!S_ISREG(s2.st_mode)){
        fprintf(stderr, "<FILENAME2> is only Regular file\n");
        return;
}
//FILENAME1과 FILENAME2의 mtime과 파일크기를 비교한다.
```

```
        //두 파일이 동일할경우 표준출력으로 동일하다는 문구 출력한다.
        if((s.st_size==s2.st_size) && (s.st_mtime==s2.st_mtime)){
                printf("%s와 %s는 동일한 파일입니다.\n",tempName, tempName2);
        }
        //두파일이 동일하지 않을 때 표준출력으로 각 파일의 mtime과 파일 크기를 출력한다.
        else{
                printf("%s %ld %ld\n",tempName, s.st_mtime, s.st_size);
                printf("%s %ld %ld\n",tempName2, s2.st_mtime, s2.st_size);
        }
}

//명령어 recover를 실행하는 함수이다.
void command_recover(char* command){
        char dummy_command1[1024];
        char *dummy_command2;
        char *dummy_command3;
        char *dummy_command4;
        char filename[MAX_SIZE];
        char pathname[MAX_SIZE];
        char tempName[100];
        char pathname2[MAX_SIZE];
        char filename2[MAX_SIZE];
        char tempName2[100];
        char tempName3[100];
        char newfile[300];
        char fpath[1024];
        Node *temp;
        char dum[400];
        char dum1[400];
        char *dum2;
        char *dum3;
        int count=0; //디렉토리 내부의 파일 개수
        int num=0; // filename과 동일한 백업파일의 개수
        struct dirent **dentry;
        char *ptr;//백업시간을 가리키는 포인터
        char path[200][1024]; //디렉토리 내부의 파일명을 저장
        int file[1024]; //디렉토리 내부의 파일과 filename이 같을 때의 인덱스를 저장하는 변수
        char buf[200][200]; //백업시간을 저장
        struct stat s[200];
        int choose;
        char choose2[5];
        char system_command[1024];
        int check_f=0;
        int check_f2=0;
```

```
int check_count=0;
char *ddummy;

strcpy(dummy_command1,command);

dummy_command2 = strtok(dummy_command1," ");
dummy_command3 = strtok(NULL," ");//FILENAME
dummy_command2 = strtok(NULL," "); //option
if(dummy_command2 !=NULL){//option이 있다.
        if(dummy_command2[0]=='-'){
                if(dummy_command2[1]=='n'){
                        dummy_command4 =strtok(NULL," ");
                        if(dummy_command4==NULL){
                                fprintf(stderr, "usage: recover <FILENAME> [-n <NEWFILE>]\n");
                                return;
                        }
                        else{
                                //상대경로와 절대경로 모두 입력 가능해야한다.
                                strcpy(tempName2, dummy_command4);
                                //절대경로로 변환
                                if(tempName[0]=='.'){
                                        ddummy=dummy_command4+2;
                                        strcpy(dummy_command3,ddummy);
                                        strcpy(dum1,dummy_command4);
                                        dum3 = strtok(dum1,"/");
                                        while(dum3!=NULL){
                                                if((strstr(dum3,"."))!=NULL){
                                                        check_f=1;
                                                        check_f2=1;
                                                        break;
                                                }
                                                dum3=strtok(NULL,"/");
                                        }
                                        if(!check_f){
                                                strcpy(dum1,dummy_command4);
                                                dum3 = strtok(dum1,"/");
                                                while(dum2!=NULL){
                                                        if((strstr(dum3,"_"))!=NULL){
                                                                check_f=0;
                                                                check_f2=1;
                                                                break;
                                                        }
                                                        dum3=strtok(NULL,"/");
                                                }
```

```
		}
		if(!check_f2){
			strcpy(dum1,dummy_command4);
			dum3 = strtok(dum1,"/");
			while(dum2!=NULL){
				check_count++;
				dum2=strtok(NULL,"/");
			}
			strcpy(dum1,dummy_command4);
			dum3 = strtok(dum1,"/");
			check_count--;
			while(dum3!=NULL){
				if(check_count==0){
					check_f2=0;
					break;
				}
				check_count--;
				dum3=strtok(NULL,"/");
			}
		}
		strcpy(tempName2,dum3);
		getcwd(pathname2,MAX_SIZE);
		sprintf(filename2,"%s/%s",pathname2,dummy_command4);
	}
	else if(tempName2[0]!='/'){
		getcwd(pathname2,MAX_SIZE);
		sprintf(filename2,"%s/%s",pathname2,tempName2);
	}
	else{
		strcpy(dum1,dummy_command4);
		dum3 = strtok(dum1,"/");
		while(dum3!=NULL){
			if((strstr(dum3,"."))!=NULL){
				check_f=1;
				check_f2=1;
				break;
			}
			dum3=strtok(NULL,"/");
		}
		if(!check_f){
			strcpy(dum1,dummy_command4);
			dum3 = strtok(dum1,"/");
			while(dum3!=NULL){
				if((strstr(dum3,"_"))!=NULL){
```

```
                                        check_f=0;
                                        check_f2=1;
                                        break;
                                }
                                dum3=strtok(NULL,"/");
                        }
                }
                if(!check_f2){
                        strcpy(dum1,dummy_command4);
                        dum3 = strtok(dum1,"/");
                        while(dum3!=NULL){
                                check_count++;
                                dum3=strtok(NULL,"/");
                        }
                        strcpy(dum1,dummy_command4);
                        dum3 = strtok(dum1,"/");
                        check_count--;
                        while(dum3!=NULL){
                                if(check_count==0){
                                        check_f2=0;
                                        break;
                                }
                                check_count--;
                                dum3=strtok(NULL,"/");
                        }
                }
                strcpy(tempName2,dum3);
                strcpy(filename2,dummy_command4);
        }
        //NEWFILE이 이미 존재한다면 에러처리 후 프롬프트로 제어가 넘어간다.
        if(!access(filename2, F_OK)){
                fprintf(stderr, "usage : <NEWFILE> is existed\n");
                return;
        }
        //한글이 포함되면 에러처리 후 프롬프트로 제어가 넘어간다.
        for(int i=0;i<strlen(filename2);i++){
                if(filename2[i]>=33 && filename2[i]<=122){
                        continue;
                }
                else{
                        fprintf(stderr, "can't use Hangul\n");
                        return;
                }
        }
```

```
                                        //filename이 255byte를 넘을 시 에러처리 후 프롬프트로 제어가 넘어간다.
                                        if(strlen(filename2)>255){
                                                fprintf(stderr, "<NEWFILE> is over 255byte\n");
                                                return;
                                        }
                                }
                        }
                        else{
                                fprintf(stderr, "존재하지않는 옵션입니다\n");
                                return;
                        }
                }
                else{
                        fprintf(stderr, "usage: recover <FILENAME> [-n <NEWFILE>]\n");
                        return;
                }
        }
        else{
                dummy_command2=NULL;
        }
        strcpy(tempName, dummy_command3);

        //절대경로로 변환
        if(tempName[0]=='.'){
                ddummy=dummy_command3+2;
                strcpy(dummy_command3,ddummy);
                strcpy(dum,dummy_command3);
                dum2 = strtok(dum,"/");
                while(dum2!=NULL){
                        if((strstr(dum2,"."))!=NULL){
                                check_f=1;
                                check_f2=1;
                                break;
                        }
                        dum2=strtok(NULL,"/");
                }
                if(!check_f){
                        strcpy(dum,dummy_command3);
                        dum2 = strtok(dum,"/");
                        while(dum2!=NULL){
                                if((strstr(dum2,"_"))!=NULL){
                                        check_f=0;
                                        check_f2=1;
                                        break;
```

```
                        }
                        dum2=strtok(NULL,"/");
                }
        }
        if(!check_f2){
                strcpy(dum,dummy_command3);
                dum2 = strtok(dum,"/");
                while(dum2!=NULL){
                        check_count++;
                        dum2=strtok(NULL,"/");
                }
                strcpy(dum,dummy_command3);
                dum2 = strtok(dum,"/");
                check_count--;
                while(dum2!=NULL){
                        if(check_count==0){
                                check_f2=0;
                                break;
                        }
                        check_count--;
                        dum2=strtok(NULL,"/");
                }
        }
        strcpy(tempName,dum2);
        getcwd(pathname,MAX_SIZE);
        sprintf(filename,"%s/%s",pathname,dummy_command3);
}
else if(tempName[0]!='/'){
        getcwd(pathname,MAX_SIZE);
        sprintf(filename,"%s/%s",pathname,tempName);
}
else{
        strcpy(dum,dummy_command3);
        dum2 = strtok(dum,"/");
        while(dum2!=NULL){
                if((strstr(dum2,"."))!=NULL){
                        check_f=1;
                        check_f2=1;
                        break;
                }
                dum2=strtok(NULL,"/");
        }
        if(!check_f){
                strcpy(dum,dummy_command3);
```

```
                dum2 = strtok(dum,"/");
                while(dum2!=NULL){
                        if((strstr(dum2,"_"))!=NULL){
                                check_f=0;
                                check_f2=1;
                                break;
                        }
                        dum2=strtok(NULL,"/");
                }
        }
        if(!check_f2){
                strcpy(dum,dummy_command3);
                dum2 = strtok(dum,"/");
                while(dum2!=NULL){
                        check_count++;
                        dum2=strtok(NULL,"/");
                }
                strcpy(dum,dummy_command3);
                dum2 = strtok(dum,"/");
                check_count--;
                while(dum2!=NULL){
                        if(check_count==0){
                                check_f2=0;
                                break;
                        }
                        check_count--;
                        dum2=strtok(NULL,"/");
                }
        }
        strcpy(tempName,dum2);
        strcpy(filename,dummy_command3);
}

//한글이 포함되면 에러처리 후 프롬프트로 제어가 넘어간다.
for(int i=0;i<strlen(filename);i++){
        if(filename[i]>=33 && filename[i]<=122){
                continue;
        }
        else{
                fprintf(stderr, "can't use Hangul\n");
                return;
        }
}
```

```
//filename이 255byte를 넘을 시 에러처리 후 프롬프트로 제어가 넘어간다.
if(strlen(filename)>255){
	fprintf(stderr, "filename is over 255byte\n");
	return;
}

//파일이 존재하지 않을 시 에러처리 후 프롬프트로 제어가 넘어간다.
if(access(filename, F_OK)){
	fprintf(stderr, "usage : <FILENAME> is not exist\n");
	return;
}

//변경할 파일이 현재 백업 리스트에 존재한다면 백업 수행 종료 후 복구를 진행한다.
if(isInList(&list,filename)){
	//백업하는 중인 쓰레드를 종료한다.
	pthread_cancel(searchNode(&list,filename)->tid);
	printLog(filename,3);
	delNode(&list,filename);
}

if((count = scandir(dirname,&dentry, NULL,alphasort)) == -1){
	fprintf(stderr, "scandir error\n");
	exit(1);
}

sprintf(tempName3,"/%s_",tempName);
for(int i=0;i<count;i++){
	realpath(dentry[i]->d_name,fpath);
	if(strstr(fpath,tempName3))
		file[num++]=i;
}

//변경할 파일에 대한 백업파일이 존재하지 않으면 에러처리 후 프롬프트로 제어가 넘어간다.
if(num==0){
	fprintf(stderr, "복구할 수 있는 백업파일이 존재하지 않습니다.\n");
	return;
}

for(int i=0;i<num;i++){
	sprintf(path[i],"%s/%s",dirname,dentry[file[i]]->d_name);//path에 backup내부의 상대경로를 저장
	if(stat(path[i],&s[i])<0){
		fprintf(stderr, "stat error\n");
		return;
	}
```

```
                ptr=dentry[file[i]]->d_name+strlen(dentry[file[i]]->d_name)-12;//백업시간을 저장
                strcpy(buf[i],ptr);//buf에 복사
        }
        printf("0.    exit\n");
        for(int i=0;i<num;i++){
                char sp[5];
                sprintf(sp,"%d.",i+1);
                printf("%-5s%s\t%ldbytes\n",sp,buf[i],s[i].st_size);
        }
        printf("Choose file to recover : ");
        fgets(choose2,sizeof(choose2),stdin);
        choose=atoi(choose2);
        if(choose==0)
                return;
        else{
                if(dummy_command2!=NULL){
                        sprintf(system_command,"cp %s %s", path[choose-1],filename2);
                }
                else{
                        sprintf(system_command,"cp %s %s",path[choose-1],filename);
                }
                system(system_command);
                printf("Recovery success\n");
                printLog(filename,4);
                if(dummy_command2!=NULL){
                        recoverFileRead(filename2);
                }
                else{
                        recoverFileRead(filename);
                }
        }
        for(int i=0;i<count;i++)
                free(dentry[i]);
        free(dentry);
        return;
}

//명령어 list를 실행시키는 함수이다.
void command_list(){
        print(&list);
}

//명령어 add, remove, recover와 백업파일을 생성시킬 때의 기록을 로그파일에 남기는 함수이다.
void printLog(char *name, int index){
```

```
        time_t timer = time(NULL);
        struct tm *t = localtime(&timer);
        char temp_d[7];
        char temp_t[7];
        sprintf(temp_d,"%02d%02d%02d",t->tm_year-100,t->tm_mon+1,t->tm_mday);
        sprintf(temp_t,"%02d%02d%02d",t->tm_hour, t->tm_min, t->tm_sec);
        FILE *fd;
        char logName[300];
        char logContext[1024];
        Node *temp;
        char backupName[300];
        sprintf(logName,"%s/log.txt",dirname);

        //file을 open한다.
        if((fd=fopen(logName,"a+"))==NULL){//file open (append)
                fprintf(stderr, "fopen error\n");
                exit(1);
        }
        //file에 기록을 남긴다.
        switch(index){
                case 1:
                        temp = (Node*)malloc(sizeof(Node*));
                        temp = searchNode(&list,name);

sprintf(backupName,"%s/%s_%s%s",dirname,temp->filename,temp->temp_d,temp->temp_t);
                        sprintf(logContext,"[%s %s] %s generated\n",temp->temp_d,temp->temp_t,backupName);
                        fputs(logContext,fd);
                        break;
                case 2:
                        sprintf(logContext,"[%s %s] %s added\n",temp_d,temp_t,name);
                        fputs(logContext,fd);
                        break;
                case 3:
                        sprintf(logContext,"[%s %s] %s deleted\n",temp_d,temp_t,name);
                        fputs(logContext,fd);
                        break;
                case 4:
                        sprintf(logContext,"[%s %s] %s recovered\n",temp_d,temp_t,name);
                        fputs(logContext,fd);
                        break;
        }
        fclose(fd); //file close
}
```

```
//쓰레드에서 실행할 함수이다. 리스트에 추가한 파일을 백업시키는 함수이다.
void *backupFile(void *arg){
	pthread_t tid;
	Node *tempNode = (Node*)arg;
	char backupName[300];
	char commandname[MAX_SIZE];
	struct stat s;
	int count=0; //디렉토리 내부의 파일 개수
	struct dirent **dentry;
	char path[MAX_SIZE]; //디렉토리 내부의 파일명을 저장
	int file[MAX_SIZE]; //디렉토리 내부의 파일과 filename이 같을 때의 인덱스를 저장하는 변수
	char tempName3[100];
	char fpath[MAX_SIZE];
	char *ptr;//백업시간을 가리키는 포인터
	char *tptr;
	char buf[200]; //백업시간을 저장
	char buf_h[20];
	char buf_m[20];
	char buf_s[20];
	char buf_d[20];
	tid=pthread_self();
	tempNode->tid = tid;

	while(1){
		int buf_t=0;
		int tmp_t=0;
		int num=0; // filename과 동일한 백업파일의 개수
		sleep(tempNode->period);

		pthread_mutex_lock(&mutex);

		time_t timer = time(NULL);
		struct tm *t = localtime(&timer);
		char temp_d[7];
		char temp_t[7];

		if(stat(tempNode->file,&s)<0){
			fprintf(stderr, "stat error\n");
			exit(1);
		}
		sprintf(temp_d,"%02d%02d%02d",t->tm_year-100,t->tm_mon+1,t->tm_mday);
		sprintf(temp_t,"%02d%02d%02d",t->tm_hour, t->tm_min, t->tm_sec);
		strcpy(tempNode->temp_d, temp_d);
		strcpy(tempNode->temp_t, temp_t);
```

```
sprintf(backupName,"%s/%s_%s%s",dirname,tempNode->filename,tempNode->temp_d,tempNode->temp_t);
                sprintf(commandname,"cp %s %s",tempNode->file,backupName);
                if(tempNode->list_m_flag){
                        if(tempNode->mtime_record!=s.st_mtime){
                                system(commandname);
                                printLog(tempNode->file,1);
                                tempNode->mtime_record = s.st_mtime;
                        }
                }
                else{
                        system(commandname);
                        printLog(tempNode->file,1);
                }

                //t옵션 있을 때
                if(tempNode->list_t_flag){
                        if((count = scandir(dirname,&dentry, NULL,alphasort)) == -1){
                                fprintf(stderr, "scandir error\n");
                                exit(1);
                        }
                        sprintf(tempName3,"/%s_",tempNode->filename);
                        for(int i=0;i<count;i++){
                                realpath(dentry[i]->d_name,fpath);
                                if(strstr(fpath,tempName3)){
                                        file[num++]=i;
                                }
                        }
                        for(int i=0;i<num;i++){
                                sprintf(path,"%s/%s",dirname,dentry[file[i]]->d_name);//path에 backup내부의 절대
경로를 저장
                                ptr=dentry[file[i]]->d_name+strlen(dentry[file[i]]->d_name)-8;//백업시간을 저장
                                strcpy(buf,ptr);//buf에 복사
                                strcpy(buf_d,buf);
                                tptr=buf+strlen(buf)-6;
                                strcpy(buf_h,tptr);
                                tptr=buf+strlen(buf)-4;
                                strcpy(buf_m,tptr);
                                tptr=buf+strlen(buf)-2;
                                strcpy(buf_s,tptr);
                                buf_d[2]='\0';
                                buf_h[2]='\0';
                                buf_m[2]='\0';
                                buf_t = (atoi(buf_d)*86400)+(atoi(buf_h)*3600)+(atoi(buf_m)*60)+atoi(buf_s);
```

```
                        tptr=temp_d+strlen(temp_d)-2;
                        strcpy(buf_d,tptr);
                        strcpy(buf_h,temp_t);
                        tptr=temp_t+strlen(temp_t)-4;
                        strcpy(buf_m,tptr);
                        tptr=temp_t+strlen(temp_t)-2;
                        strcpy(buf_s,tptr);
                        buf_h[2]='\0';
                        buf_m[2]='\0';
                        tmp_t = (atoi(buf_d)*86400)+(atoi(buf_h)*3600)+(atoi(buf_m)*60)+atoi(buf_s);

                        if((tmp_t - buf_t)>=(tempNode->addTime)){
                                remove(path);
                        }
                }

                for(int i=0;i<count;i++)
                        free(dentry[i]);
                free(dentry);
        }

        //n옵션 있을 때
        if(tempNode->list_n_flag){
                optionAddN(tempNode->filename,tempNode->number);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}

//add -n옵션을 수행하는 함수
void optionAddN(char *filename,int number){
    int count=0; //디렉토리 내부의 파일 개수
    int num=0; // filename과 동일한 백업파일의 개수
    struct dirent **dentry;
    char *ptr;//백업시간을 가리키는 포인터
    char path[MAX_SIZE]; //디렉토리 내부의 파일명을 저장
    int file[MAX_SIZE]; //디렉토리 내부의 파일과 filename이 같을 때의 인덱스를 저장하는 변수
    char buf[20]; //백업시간을 저장
    struct stat s;
    char tempName3[400];
    char fpath[MAX_SIZE];
```

```
	if((count = scandir(dirname,&dentry, NULL,alphasort)) == -1){
		fprintf(stderr, "scandir error\n");
		exit(1);
	}

	sprintf(tempName3,"/%s_",filename);
	for(int i=0;i<count;i++){
		realpath(dentry[i]->d_name,fpath);
		if(strstr(fpath,tempName3))
			file[num++]=i;
	}

	for(int i=0;i<num-number;i++){
		sprintf(path,"%s/%s",dirname,dentry[file[i]]->d_name);//path에 backup내부의 상대경로를 저장
		ptr=dentry[file[i]]->d_name+strlen(dentry[file[i]]->d_name)-12;//백업시간을 저장
		strcpy(buf,ptr);//buf에 복사
		remove(path);
	}

	for(int i=0;i<count;i++)
		free(dentry[i]);
	free(dentry);
	return;
}

//add -d옵션을 수행하는 함수
void optionAddD(char *filename,int period, char*option){
	struct dirent **dentry;
	pthread_t tid;
	int nitems;
	char pathname[MAX_SIZE];
	struct stat fstat;
	char add_command_d[MAX_SIZE];

	nitems=scandir(filename,&dentry,NULL,alphasort);
	for(int i=2;i<nitems;i++){
		sprintf(pathname,"%s/%s",filename,dentry[i]->d_name);
		stat(pathname, &fstat);
		if(S_ISDIR(fstat.st_mode))		{
			optionAddD(pathname,period,option);
		}
		else if(S_ISREG(fstat.st_mode)){
			if(isInList(&list,pathname)){
				continue;
```

```
                        }
                        sprintf(add_command_d,"add %s %d%s",pathname,period,option);
                        command_add(add_command_d);
                }
        }
}

//백업한 파일로 변경 후 변경된 파일의 내용을 출력하는 함수이다.
void recoverFileRead(char* fname){
        char c_str_read[2048];
        FILE *fd;
        int cLine=0;
        printf("\n");
        if((fd=fopen(fname, "r"))==NULL){
                fprintf(stderr, "fopen error for %s\n",fname);
                exit(1);
        }
        else{
                while(1){
                        fgets(c_str_read,2048,fd);
                        if(feof(fd))
                                break;
                        printf("%-4d %s",cLine+1,c_str_read);
                        cLine++;
                }
        }
        printf("\n");
        fclose(fd);
}

//리스트 내용을 csv파일에 저장해준다.
void makeTable() {
        FILE *pFile;
        char pFilePath[MAX_SIZE];

        sprintf(pFilePath, "%s/listLog.csv", dirname);
        Node *current = list.head;
        pFile = fopen(pFilePath, "w");
        while(current != NULL){

fprintf(pFile,"%s,%d,%s,%s,%d,%d,%d,%d,%d,%d\n",current->file,current->period,current->opt,current->filename,c
urrent->mtime_record,current->list_m_flag,current->list_t_flag,current->addTime,current->list_n_flag,current->n
umber);
                current=current->next;
```

```
        }
        fclose(pFile);
}

//csv파일을 읽어와 그 정보로 싱글링크드리스트를 복원한다.
void readTable(){
        FILE *pFile;
        char pFilePath[MAX_SIZE];
        char strA[MAX_SIZE];
        char strB[MAX_SIZE];
        char file[256];
        int period;
        char opt[100];
        char filename[100];
        int mtime_record;
        int list_m_flag;
        int list_t_flag;
        int addTime;
        int list_n_flag;
        int number;

        sprintf(pFilePath, "%s/listLog.csv", dirname);
        pFile = fopen(pFilePath, "r");
        while(fgets(strA,sizeof(strA),pFile)!=NULL){
                strcpy(strB,strtok(strA,","));
                strcpy(file, strB);
                strcpy(strB,strtok(NULL,","));
                period=atoi(strB);
                strcpy(strB,strtok(NULL,","));
                strcpy(opt, strB);
                strcpy(strB,strtok(NULL,","));
                strcpy(filename, strB);
                strcpy(strB,strtok(NULL,","));
                mtime_record=atoi(strB);
                strcpy(strB,strtok(NULL,","));
                list_m_flag=atoi(strB);
                strcpy(strB,strtok(NULL,","));
                list_t_flag=atoi(strB);
                strcpy(strB,strtok(NULL,","));
                addTime=atoi(strB);
                strcpy(strB,strtok(NULL,","));
                list_n_flag=atoi(strB);
                strcpy(strB,strtok(NULL,","));
                number=atoi(strB);
```

```
InsertNode(&list,file,period,opt,filename,mtime_record,list_m_flag,list_t_flag,addTime,list_n_flag,number);
                        if(strtok(NULL,",")=="\n")
                                continue;
        }
        fclose(pFile);
}
```