1. 실행 시 인자로 주어진 파일의 타입이 나오는 프로그램을 작성하시오. 단, 아래 조건과 실행 결과를 바탕으로 프로그램을 작성 할 것.

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
2. 프로그램 실행 시 입력한 인자는 각각 정규파일, 디렉토리, 케릭터 특수, 블록 특수, FIFO, 심볼릭 링크, 소켓 파일로 가정
3. 파일 타입은 print_fiole_type 함수를 통해 출력할 것

1.c

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

void print_file_type(struct stat *statbuf) {
    char *str;

    if (S_ISREG(statbuf -> st_mode)) // (statbuf->st_mode & S_IFMT) == S_IFREG 가능
        str = "regular";
    else if (S_ISDIR(statbuf -> st_mode)) // (statbuf->st_mode & S_IFMT) == S_IFDIR 가능
        str = "directory";
    else if (S_ISCHR(statbuf -> st_mode)) // (statbuf->st_mode & S_IFMT) == S_IFCHR 가능
        str = "character special";
    else if (S_ISBLK(statbuf -> st_mode)) // (statbuf->st_mode & S_IFMT) == S_IFBLK 가능
        str = "block special";
    else if (S_ISFIFO(statbuf -> st_mode)) // (statbuf->st_mode & S_IFMT) == S_IFIFO 가능
        str = "FIFO";
    else if (S_ISLNK(statbuf -> st_mode)) // (statbuf->st_mode & S_IFMT) == S_IFBLK 가능
        str = "symbolic link";
    else if (S_ISSOCK(statbuf -> st_mode)) // (statbuf->st_mode & S_IFMT) == S_IFSOCK 가능
        str = "socket";
    else
        str = "unknown mode";

    printf("%s\n", str);
}

int main(int argc, char *argv[])
{
    struct stat statbuf;
    int i;
    FILE *fp

    if((fp=fopen(argv[0], "r")) < 0){
        fprintf(stderr, "fopen error for %s\n", argv[0]);
        exit(1);
    }
    fgets(NULL, 0, fp);
    fclose(fp);
```

```
    if((fp=fopen(argv[0], "r")) < 0){
        fprintf(stderr, "fopen error for %s\n", argv[0]);
        exit(1);
    }
    fgets(NULL, 0, fp);
    fclose(fp);

    for(i = 1; i < argc; i++){
        if (lstat(argv[i], &statbuf) < 0  ) {
            fprintf(stderr, "error\n");
            exit(1);
        }
        print_file_type(&statbuf);
    }

    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out regular directory character block fifo symbolic socket
regular
directory
character special
block special
FIFO
symbolic link
socket
```

2. 다음 주어진 ssu_answer.txt에 있는 학생들의 답을 채점하여 그 결과를 ssu_res.txt에 저장하는 프로그램을 작성하시오. 단, 아래 조건과 실행 결과를 바탕으로 프로그램을 작성 할 것.

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
2. fopen(), fgets(), fclose()를 각 두 번씩 사용할 것
3. 두 개의 fopen()함수의 에러 처리를 위해서 fprintf()를 각각 한 번 사용할 것
4. 표준 출력을 위하여 printf()를, 또한 파일 출력을 위하여 fprintf()를 각각 한 번 사용할 것

<ssu_answer.txt>

```
Jung DJ
1234123412
Lee SS
1233423413
Hong GD
2233412424
```

2.c

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

#define BUFFER_SIZE 256
#define STUDENT_NUM 3
#define Q_SIZE 10
```

```
typedef struct _student {
	char name[10];
	int score ;
	char res[BUFFER_SIZE];
} Student;

char answer[BUFFER_SIZE] = "1233423413"; //test's answer

int main(void)
{
	char *ssu_answer = "ssu_answer.txt";
	char *ssu_res = "ssu_res.txt";
	char tmp_score[BUFFER_SIZE];
	FILE *fp;
	int i, j= 0;
	Student list[STUDENT_NUM];

	 if ((fp = fopen(ssu_answer,"rb")) == NULL) {
		fprintf(stderr, "fopen error for %s\n",ssu_answer);
		exit(1);
	}

	for(j = 0 ; j < STUDENT_NUM ; j++) {
		list[j].score = 0;
		if (fgets(list[j].name, BUFFER_SIZE, fp) == NULL) {
			fprintf(stderr, "fgets error for %s\n", ssu_answer);
			exit(1);
		}
		 i = 0;

		while((list[j].name[i] !='\n'))
			i++;

		list[j].name[i] = '\0';

		if(fgets(list[j].res, BUFFER_SIZE, fp) == NULL){
			fprintf(stderr, "fgets error for %s\n", ssu_answer);
			exit(1);
		}
		i = 0;

		while((list[j].res[i]  !='\n')) i++;

		list[j].res[i] = '\0';
		i = 0 ;

		for( ; i < Q_SIZE ; i++) {
			if (list[j].res[i] == answer[i]) {
				list[j].score += 10;
				list[j].res[i] = 'O';
```

```
            }
            else
                list[j].res[i] = 'X';
        }
        printf("Student name : %s , score : %d , res : %s \n", list[j].name, list[j].score, list[j].res);
    }

    fclose(fp);
    if ((fp = fopen(ssu_res, "wb")) == NULL) {
        fprintf(stderr, "fopen error for %s \n",ssu_res);
        exit(1);
    }

    for (i = 0 ; i < STUDENT_NUM ; i++) {
            fprintf(fp, "%s |%d| %s\n", list[i].name, list[i].score, list[i].res);
    }

    fclose(fp);
    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
Student name : Jung DJ , score : 70 , res : OOOXXOOOOX
Student name : Lee SS , score : 100 , res : OOOOOOOOOO
Student name : Hong GD , score : 50 , res : XOOOOXXOXX
root@localhost:/home/oslab# cat ssu_res.txt
Jung DJ |70| OOOXXOOOOX
Lee SS |80| OOOOOOOOOO
Hong GD |50| XOOOOXXOXX
```

3. 다음 주어진 ssu_test.txt 파일을 한 줄씩 읽고 그 줄을 출력하며 전체 줄 수를 출력하는 프로그램을 작성하시오. 단, 아래 조건과 실행 결과를 바탕으로 프로그램을 작성 할 것.

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
2. open(), read(), lseek(), close()를 각 한 번씩 사용할 것
3. open()함수의 에러 처리를 위해 fprintf()를 사용할 것
4. 출력을 위해 printf()를 두 번 사용할 것
5. 입력 텍스트 파일이 변경되어도 정상적으로 작동되어야 함

<ssu_test.txt>

```
Linux System Programming!
Unix System Programming!
Linux Mania
Unix Mania
```

3.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
```

```c
#include <sys/types.h>

#define BUFFER_SIZE 1024
#define WORD_MAX 100

int main(void)
{
    int fd;
    int length = 0, offset = 0, count = 0;
    char *fname = "ssu_test.txt";
    char buf[WORD_MAX][BUFFER_SIZE];
    int i;

    if ((fd = open(fname, O_RDONLY)) < 0) {
        fprintf(stderr, "open error for %s .\n", fname);
        exit(1);
    }

    while ((length = read(fd, buf[count], BUFFER_SIZE)) > 0 ) {
        buf[count][length] = '\0';
        for ( i = 0 ; i < BUFFER_SIZE ; i++) {
            if (buf[count][i] == '\n') {
                if (i == 0)
                    break;
                offset = offset + i + 1;
                lseek(fd, offset, SEEK_SET);
                count++;
            }
        }
    }

    close(fd);
    for (i = 0 ; i < count ; i++)
        printf("%s\n", buf[i]);
    printf("line number : %d \n", count);
    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
Linux System Programming!
UNIX System Programming!
Linux Mania
Unix Mania

UNIX System Programming!
Linux Mania
Unix Mania

Linux Mania
Unix Mania
```

```
Unix Mania

line number : 4
```

4. 구조체 내용을 파일에 저장한 후, 해당 파일을 읽어 출력을 하는 프로그램을 작성하시오. 단, 아래 조건과 실행 결과를 바탕으로 프로그램을 작성 할 것.

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
2. ftest.txt. 파일을 fopen()을 통해 쓰기 모드로 한 번 호출하고 에러 처리를 위하여 fprintf()를 사용할 것
3. 구조체의 내용을 파일에 저장하기 위하여 fwrite()를 한 번만 사용할 것
4. ftest.txt 파일을 fopen()을 통해 읽기 모드로 한 번 호출하고 에러 처리를 위하여 fprintf()를 사용할 것
5. fread(), fclose()를 각각 두 번 사용하고 rewind()를 한 번 사용할 것

4.c

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

typedef struct _person {
    char name[10];
    int age;
    double height;
} Person;

int main(void)
{
    FILE *fp;
    int i, res;
    Person ary[3] = {{"Hong GD", 500, 175.4},
        {"Lee SS", 350,180.0},
        {"King SJ", 500, 178.6}};
    Person tmp;

    if((fp = fopen("ftest.txt", "wb")) == 0) {
        fprintf(stderr, "fopen error!\n");
        exit(1);
    }
    fwrite(&ary,sizeof(ary),1,fp);

    fclose(fp);

    if ((fp = fopen("ftest.txt", "rb")) == 0) {
        fprintf(stderr , "fopen error!\n");
        exit(1);
    }

    printf("[ First print]\n");

    while (!feof(fp)) {
        if ((res = fread(&tmp, sizeof(Person), 1, fp)) != 1)
```

```
            break;
        printf("%s %d %.2lf\n", tmp.name, tmp.age, tmp.height);
    }

    rewind(fp);
    printf("[ Second print]\n");

    while (!feof(fp)) {
        if ((res = fread(&tmp, sizeof(Person), 1, fp)) != 1)
            break;
        printf("%s %d %.2lf\n", tmp.name, tmp.age, tmp.height);
    }

    fclose(fp);
    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.ou
[ First print]
Hong GD 500 175.40
Lee SS 350 180.00
King SJ 500 178.60
[ Second print]
Hong GD 500 175.40
Lee SS 350 180.00
King SJ 500 178.60
root@localhost:/home/oslab#
```

5. read()와 write()를 사용하여 파일을 복사하는 프로그램을 작성하시오, 단, 아래 조건과 실행 결과를 바탕으로 프로그램을 작성 할 것.

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
2. open(), close()를 각각 두 번씩 사용하고 read()와 write()를 각각 한 번씩 사용할 것
3. main()의 인자의 에러 처리를 위하여(프로그램 사용법) fprintf()를 사용할 것
4. open()함수의 에러 처리를 위해 fprintf()를 사용할 것
5. 생성되는 파일의 mode는 0644이며 파일 허가(권한) 매크로를 사용하여 작성할 것
6. BUFFER_SIZE보다 큰 파일도 복사가 되어야 함
7. 지정된 이름을 갖는 파일이 있어도 복사가 되어야 함

5.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>

#define BUFFER_SIZE 128
#define MODE (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)
```

```
int main(int argc, char *argv[])
{
    char buf[BUFFER_SIZE];
    int fd1, fd2;
    ssize_t size;

    if(argc != 3){
        fprintf(stderr, "Usagee : a.out filename1 filename2\n");
        exit(1);
    }

    if((fd1 = open(argv[1], O_RDONLY)) < 0){
        fprintf(stderr, "open error for %s\n", argv[1]);
        exit(1);
    }

    if((fd2 = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, MODE)) < 0){
        fprintf(stderr, "open error for %s\n", argv[1]);
        exit(1);
    }

    while((size = read(fd1, buf, sizeof(buf))) > 0){
        write(fd2, buf, size);
    }

    close(fd1);
    close(fd2);
}
```

실행결과

```
root@localhost:/home/oslab# cat ssu_file1
Linux System Programming!
root@localhost:/home/oslab# ./a.out ssu_file1 ssu_file2
root@localhost:/home/oslab# ls
9  9.c  ssu_file1  ssu_file2
root@localhost:/home/oslab# cat ssu_file2
Linux System Programming!
root@localhost:/home/oslab#
```

6. 아래 프로그램은 파일 디스크립터를 이용하여 "ssu_test.txt"를 읽고 읽은 내용을 표준출력 및 파일 포인터를 사용하여 "ssu_test_new.txt"에 쓴다. 아래 조건과 실행 결과를 바탕으로 프로그램을 작성하시오.

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
2. open() <-> fopen(), close() <-> fclose()로 변경할 것
3. read() <-> fread(), write() <-> fwrite()로 변경할 것
4. lseek() <-> fseek()으로 변경할 것
5. 파일을 읽고 쓰는 순서는 변경 후에도 동일해야 함
6. 코드가 변경되어도 실행결과는 동일해야 함
7. ssu_test_new.txt는 없을 경우 생성되어야 하고, 이미 존재할 경우 기존 내용은 제거되어야 함
8. ssu_test_new.txt의 권한은 0644로 할 것

<ssu_test.txt>

```
Linux System Programming!
Unix System Programming!
```

6.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

#define BUFFER_SIZE 1024

int main(void)
{
        char buf[BUFFER_SIZE];
        char *fname = "ssu_test.txt";
        char *new_fname = "ssu_test_new.txt";
        int fd;
        FILE *fp;

        fd = open(fname, O_RDONLY);
        fp = fopen(new_fname, "w");
        if(fd < 0 || fp == NULL){
                fprintf(stderr, "open error for %s\n", fname);
                exit(1);
        }

        read(fd, buf, 25);
        buf[25] = 0;
        printf("first printf : %s\n", buf);
        lseek(fd, 1, SEEK_CUR);
        read(fd, buf+25+1, 24);
        buf[25+1+24] = 0;
        printf("second printf : %s\n", buf+25+1);
        close(fd);
        fwrite(buf, 25, 1, fp);
        fwrite(buf+25, 24, 1, fp);
        fclose(fp);
        exit(0);

        //위 코드를 아래 코드로 변경시 정답
        fp = fopen(fname, "r");
        fd = open(new_fname, O_WRONLY|O_CREAT|O_TRUNC, 0644);
        if(fd < 0 || fp == NULL){
                fprintf(stderr, "open error for %s\n", fname);
                exit(1);
        }
        fread(buf, 25, 1, fp);
        buf[25] = 0;
        printf("first printf : %s\n", buf);
        fseek(fp, 1, SEEK_CUR);
```

```
        fread(buf+25+1, 24, 1, fp);
        buf[25+1+24] = 0;
        printf("second printf : %s\n", buf+25+1);
        fclose(fp);
        write(fd, buf, 25);
        write(fd, buf+25, 24);
        close(fd);
        exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
first printf : Linux System Programming!
second printf : Unix System Programming!
```

ssu_test_new.txt

```
Linux System Programming!Unix System Programming!
```

7. system()으로 grep를 사용하여 타겟 파일에서 키워드를 검색하는 프로그램을 작성하시오. 단, 아래 조건과 실행 결과를 바탕으로 프로그램을 작성 할 것.

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
2. 타겟 파일에 디렉토리가 올 수 있는 것을 제외하고 grep의 옵션 및 사용법을 따를 것
3. 타겟 파일이 디렉토리가 아닐 경우 system()을 사용하여 grep를 실행할 것
4. 타겟 파일이 디렉토리일 경우 모든 하위파일에서 키워드를 검색할 것
5. ssu_do_grep()함수에서 파일의 정보를 얻기 위해 lstat()을 한 번 사용하고 에러 처리를 위해 fprintf()를 사용할 것
6. ssu_do_grep()함수에서 opendir()을 한 번만 사용하고 에러 처리를 위해 fprintf()를 사용할 것
7. ssu_make_grep()함수에서 string.h에 있는 함수를 사용하여 grep_cmd를 만들 것
8. 컴파일 시 -D 옵션으로 pathmax 값을 입력받지 못할 경우 pathmax의 값을 pathconf를 사용하여 구할 것. 단, 에러가 발생할 경우 MAX_PATH_GUESSED를 사용할 것
9. pathname은 malloc()을 사용하여 메모리 공간을 할당받을 것

<ssu_osdir/ssu_dir1/ssu_file1>

```
Hi,
it's the Keyword
Bye.
```

<ssu_osdir/ssu_dir2/ssu_file2>

```
Hi,
it's not the keyword
Bye.
```

7.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <dirent.h>
#include <limits.h>
#include <string.h>
#include <sys/stat.h>

#ifdef  PATH_MAX
static int pathmax = PATH_MAX;
#else
```

```
static int pathmax = 0;
#endif
#define MAX_PATH_GUESSED 1024

#ifndef LINE_MAX
#define LINE_MAX 2048
#endif

char *pathname;
char command[LINE_MAX], grep_cmd[LINE_MAX];

int ssu_do_grep(void) {
    struct dirent *dirp;
    struct stat statbuf;
    char *ptr;
    DIR *dp;

    if (lstat(pathname, &statbuf) < 0) {
        fprintf(stderr, "lstat error for %s\n", pathname);
        return 0;
    }

    if (S_ISDIR(statbuf.st_mode) == 0) {
        sprintf(command, "%s %s", grep_cmd, pathname);
        printf("%s : \n", pathname);
        system(command);
        return 0;
    }

    ptr = pathname + strlen(pathname);
    *ptr++ = '/';
    *ptr = '\0';

    if ((dp = opendir(pathname)) == NULL) {
        fprintf(stderr, "opendir error for %s\n", pathname);
        return 0;
    }

    while ((dirp = readdir(dp)) != NULL)
        if (strcmp(dirp->d_name, ".") && strcmp(dirp->d_name, "..")) {
            strcpy(ptr, dirp->d_name);

            if (ssu_do_grep() < 0)
                break;
        }

    ptr[-1] = 0;
    closedir(dp);
    return 0;
}
```

```
void ssu_make_grep(int argc, char *argv[]) {
    int i;
    strcpy(grep_cmd, " grep");

    for (i = 1;i < argc-1;i++) {
        strcat(grep_cmd, " ");
        strcat(grep_cmd, argv[i]);
    }
}

int main(int argc, char *argv[])
{
    if (argc < 2) {
        fprintf(stderr, "usage: %s <-CVbchilnsvwx> <-num> <-A num> <-B num> <-f file> \n"
                "              <-e> expr <directory>\n", argv[0]);
        exit(1);
    }

    if (pathmax == 0) {
        if ((pathmax = pathconf("/", _PC_PATH_MAX)) < 0)
            pathmax = MAX_PATH_GUESSED;
        else
            pathmax++;
    }

    if ((pathname = (char *) malloc(pathmax+1)) == NULL) {
        fprintf(stderr, "malloc error\n");
        exit(1);
    }

    strcpy(pathname, argv[argc-1]);
    ssu_make_grep(argc, argv);
    ssu_do_grep();
    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# vi ssu_osdir/ssu_dir1/ssu_file1
root@localhost:/home/oslab# vi ssu_osdir/ssu_dir2/ssu_file2
root@localhost:/home/oslab# ./a.ou
usage: ./a.out <-CVbchilnsvwx> <-num> <-A num> <-B num> <-f file>
              <-e> expr <directory>
root@localhost:/home/oslab# ./a.out -n Keyword /home/oslab/ssu_osdir
/home/oslab/ssu_osdir/ssu_dir2/ssu_file2 :
/home/oslab/ssu_osdir/ssu_dir1/ssu_file1 :
2:it's the Keyword
```

8. link()와 unlink()를 사용하여 파일을 이동하거나 이름을 변경하는 프로그램을 작성하시오. 단, 아래 조건과 실행 결과를 바탕으로 프로그램을 작성 할 것.

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
8. main()의 인자의 에러 처리를 위하여(프로그램 사용법) fprintf()를 사용할 것
2. link(), unlink()를 각각 한 번씩 사용할 것
3. link(), unlink()의 에러 처리를 위해 fprintf()를 사용할 것
4. 프로그램의 실행은 "./a.out [arg1] [arg2]" 와 같이 반드시 인자 두개를 받아야 하며, arg1에 해당하는 파일을 arg2에 해당하는 경로로 이동하거나 이름을 변경해야 함
5. 실행결과와 동일한 포맷으로 gettimeofday() 함수를 사용하여 프로그램 실행 시간을 출력해야 함. 단, 출력된 결과 값은 달라질 수 있음.
6. 측정된 수행시간은 단 한번의 printf() 함수를 사용할 것

8.c

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

#define SEC_TO_MICRO 1000000

int main(int argc, char *argv[])
{
    struct timeval begin_t, end_t;

    gettimeofday(&begin_t, NULL);

    if(argc != 3){
        fprintf(stderr, "argc != 3\n");
        exit(1);
    }

    if(link(argv[1], argv[2]) == -1){
        fprintf(stderr, "link() error\n");
        exit(1);
    }

    if(unlink(argv[1]) < 0){
        fprintf(stderr, "link() error\n");
        exit(1);
    }

    gettimeofday(&end_t, NULL);

    end_t.tv_sec -= begin_t.tv_sec;
    if(end_t.tv_usec < begin_t.tv_usec){
        end_t.tv_sec--;
        end_t.tv_usec += SEC_TO_MICROSEC;
    }
    end_t.tv_usec -= begin_t.tv_usec;

    printf("Runtime : %ld:%ld(sec:microsec)\n", end_t.tv_sec, end_t.tv_usec);
```

```
    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ls
1.txt 10.c a.out subdir
root@localhost:/home/oslab# ./a.out 1.txt 2.txt
Runtime : 0:15(sec:microsec)
root@localhost:/home/oslab# ls
10.c 2.txt a.out subdir
root@localhost:/home/oslab# ./a.out 2.txt ./subdir/2.txt
Runtime : 0:21(sec:microsec)
root@localhost:/home/oslab# ls
10.c a.out subdir
root@localhost:/home/oslab# cd subdir
root@localhost:/home/oslab/subdir# ls
2.txt
```

9. 다음은 컴파일 과정에서 컴파일이 필요한 파일만 찾아 컴파일을 시켜주는 프로그램이다. 단, 아래 조건과 실행 결과를 바탕으로 다음 프로그램을 작성하시오.

< 조 건 >

1. 프로그램 실행 시 입력으로 주어지는 인자와 Makefile에는 오류가 없음
2. Makefile에는 매크로, include, 긴 글 처리가 없으며 target-dependency 사이에 순환은 없음
3. target, dependency, command는 각각 MAX 값을 넘을 수 없음
4. command가 실행될 때만 해당 command가 출력되고 그 외의 메시지는 출력되지 않음 => 실행되는 target이 없을 경우 출력되는 것이 없음
5. Makefile은 아래의 형태만 가능
   가. target:dependency1 dependency2
   나. (tab)command
   다. #(주석)
   라. (빈 라인)
6. make 및 Makefile은 아래의 규칙을 따름
   가. target의 수정 시간이 dependency의 시간보다 최신일 경우 command는 실행되지 않음
   나. dependency가 없는 경우 해당 target은 실행됨
   다. target과 같은 이름을 같는 파일이 없을 경우 command가 실행 됨
   라. target의 dependency가 다른 target일 경우 먼저 실행됨
   마. ':' 의 앞뒤에는 공백이 올 수 없음
   바. dependency는 스페이스바로만 구분됨

Makefile

```
test.out:test1.o test2.o
#test.out:
	gcc -o test.out test1.o test2.o
test1.o:test1.c
	gcc -c test1.c
test2.o:test2.c
	gcc -c test2.c
```

9.c

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

```
#include <unistd.h>
#include <string.h>
#include <sys/stat.h>

#define MAX_TARGET 5
#define MAX_DEPENDENCY 4
#define MAX_COMMAND 5
#define INT_MAX 0x7fffffff

typedef struct {
	char name[BUFSIZ];
	char dependency[MAX_DEPENDENCY][BUFSIZ];
	char command[MAX_COMMAND][BUFSIZ];
	int dep_num;
	int cmd_num;
} target;

target target_arr[MAX_TARGET];
int target_num = -1;

int run_command(char *);

int main(int argc, char **argv){
	FILE *fp;
	char buf[BUFSIZ];
	char *ptr;

	if((fp = fopen("Makefile", "r")) == NULL){
		fprintf(stderr, "fopen error for Makefile\n");
		exit(1);
	}

	while(fgets(buf, sizeof(buf), fp) != NULL){
		if(buf[0] == '#' || buf[0] == '\n')
			continue;
		if(strchr(buf, ':') != NULL){//target
			target_num++;
			strcpy(target_arr[target_num].name, strtok(buf, ":"));

			if((ptr = strtok(NULL, "\n")) != NULL)
				strcpy(buf, ptr);
			else
				strcpy(buf, "");

			if((ptr = strtok(buf, " ")) != NULL){
				strcpy(target_arr[target_num].dependency[target_arr[target_num].dep_num], ptr);
				target_arr[target_num].dep_num++;
			}

			while((ptr = strtok(NULL, " ")) != NULL){
```

```
                strcpy(target_arr[target_num].dependency[target_arr[target_num].dep_num], ptr);
                target_arr[target_num].dep_num++;
            }
        }else{ //command
            strcpy(target_arr[target_num].command[target_arr[target_num].cmd_num], buf);
            target_arr[target_num].cmd_num++;
        }
    }

    int i;
    if(argc == 1)
        run_command(target_arr[0].name);
    else
        for(i = 1; i < argc; i++)
    run_command(argv[i]);
    exit(0);
}

/**
* target 이름을 입력받아 파일 수정시간은 반환, 파일이 없을 경우 상황에 따라 0 또는 INT_MAX 반환
* 재귀적을 실행되며 의존성 검사와 command를 실행
*/
int run_command(char *target_name){
    struct stat statbuf;
    target cur_target;
    int chk=0;
    int i;

    for(i = 0, chk=1;  i<= target_num; i++){
        if(strcmp(target_name, target_arr[i].name) == 0){
            chk = 0;
            cur_target = target_arr[i];
            break;
        }
    }

    if(chk){//dependency is not in target
        if(access(target_name, F_OK) != 0)
            return INT_MAX;

        stat(target_name, &statbuf);
        return statbuf.st_ctime;
    }

    //target
    if(access(cur_target.name, F_OK) != 0)
        statbuf.st_ctime = 0;
    else
        stat(cur_target.name, &statbuf);
```

```
        if(cur_target.dep_num == 0){
            for(i = 0; i < cur_target.cmd_num; i++){
                printf("%s", cur_target.command[i]+1);
                system(cur_target.command[i]);
            }
        return INT_MAX;
    }
    else{
        for(i = 0, chk=0; i < cur_target.dep_num; i++){
            int a;

            if(statbuf.st_ctime < (a=run_command(cur_target.dependency[i])))
                chk = 1;
        }

        if(chk)
            for(i = 0; i < cur_target.cmd_num; i++){
                printf("%s", cur_target.command[i]+1);
                system(cur_target.command[i]);
            }
    }

    //target
    if(access(cur_target.name, F_OK) != 0)
        statbuf.st_ctime = INT_MAX;
    else
        stat(cur_target.name, &statbuf);
    return statbuf.st_ctime;
}
```

실행결과

```
root@localhost:/home/oslab# ls
Makefile   a.out   test1.c   test2.c
root@localhost:/home/oslab# ./a.out
gcc -c test1.c
gcc -c test2.c
gcc -o test.out test1.o test2.o
root@localhost:/home/oslab# ls
Makefile   a.out   test.out   test1.c   test1.o   test2.c   test2.o
root@localhost:/home/oslab# ./a.out
root@localhost:/home/oslab#
```

10. 다음은 in.txt에서 특정 문자열을 찾아 해당 문자열을 삭제하고 삭제된 문자열자리에 hole을 생성해 out.txt로 출력하는 프로그램이다. 아래 조건과 실행 결과를 바탕으로 다음 프로그램을 작성하시오.

< 조 건 >

1. 다음 프로그램은 특정 문자열을 “bcd” 로 함
2. 다음의 순서에 따라 구현할 것
   가. in.txt 파일을 오픈, 파일의 사이즈를 size변수에 저장
   나. in.txt 파일의 내용을 읽어 buf 변수에 저장 (in.txt 파일의 크기는 100을 넘지 않음)
   다. in.txt 파일을 닫음
   라. out.txt 파일을 오픈
   마. buf 변수에 저장된 문자열을 out.txt에 출력. 단, out.txt에 대한 출력은 제공된 ssu_write() 함수만 사용할 것 (이외의 출력함수는 허용하지 않음)
   바. 단, pattern과 일치하는 문자열은 원래 문자열을 출력하지 않고 pattern의 길이만큼의 hole을 생성
   사. out.txt 파일을 닫음
3. ssu_write() 함수는 수정을 금지함
4. 프로그램이 정상적으로 수행되었을 시 in.txt파일과 out.txt파일의 크기는 일치할 것

in.txt 예시

aaaaabcdbbbbbcdcccbcddbcde

10.c

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
#include<string.h>

int ssu_write(int fd, char *buf);

int main()
{
        char buf[128];
        char pattern[4] = “bcd“;
        char *pos1=buf, *pos2=buf;
        char *fname_in = “in.txt“;
        char *fname_out = “out.txt“;
        int size;
        int fd1, fd2; //fd1 is input file, fd2 is output file
        int i=0;
        if((fd1 = open(fname_in, O_RDWR)) < 0){
                fprintf(stderr, “in.txt open() error\n“);
                exit(1);
        }

        if((size = lseek(fd1, 0, SEEK_END)) < 0){
                fprintf(stderr, “lseek() error\n“);
                exit(1);
        }

        if(lseek(fd1, 0, SEEK_SET) < 0){
                fprintf(stderr, “lseek() error\n“);
                exit(1);
        }

        if((fd2 = open(fname_out, O_RDWR ¦ O_CREAT, 0664)) < 0){
```

```
                fprintf(stderr, "out.txt open() error\n");
                exit(1);
        }

        if(read(fd1, buf, size) == -1){
                fprintf(stderr, "read() error\n");
                exit(1);
        }

        buf[size] = '\0';
        close(fd1);

        while((pos2 = strstr(pos1, pattern)) != NULL){
                *pos2 = '\0';
                ssu_write(fd2, pos1);
                lseek(fd2, strlen(pattern), SEEK_CUR);

                pos1 = pos2 + strlen(pattern);
        }
        ssu_write(fd2, pos1);

        close(fd1);
        close(fd2);
        return 0;
}
int ssu_write(int fd, char *buf)
{
        return write(fd, buf, strlen(buf));
}
```

실행결과 - vi out.txt 예시

```
aaaaa^@^@^@bbbb^@^@^@ccc^@^@^@dd^@^@^@e
```

실행결과

```
root@localhost:/home/oslab# ./a.out
root@localhost:/home/oslab# od -c in.txt
0000000   a   a   a   a   a   b   c   d   b   b   b   b   b   c   d   c
0000020   c   c   b   c   d   d   d   b   c   d   e  \n
0000034
root@localhost:/home/oslab# od -c out.txt
0000000   a   a   a   a   a  \0  \0  \0   b   b   b   b  \0  \0  \0   c
0000020   c   c  \0  \0  \0   d   d  \0  \0  \0   e  \n
0000034
```

11. 다음 프로그램은 하위 디렉터리를 재귀적으로 생성하는 프로그램 코드이다. 아래 조건과 실행 결과를 바탕으로 다음 프로그램을 작성하시오.

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
2. 명령어 뒤에는 인자로 숫자N(1~9)을 받는다고 가정
3. 처음 생성되는 디렉터리의 이름은 '0'(숫자)이며 하위 디렉터리는 '0 / N / N-1 / N-2 / .../ 0' 식으로 생성
4. 하위 디렉터리는 N에는 'N-1' 디렉터리 와 실행결과 <예시> 와 같이 권한이 0600인 '<N-1>ssu_test.txt'를 복사
5. 'ssu_test.txt' 파일의 복사는 선택적으로 하는 옵션으로는 -e 옵션과 -o 옵션이 있음
6. -e 옵션과 -o 옵션 뒤에는 숫자N을 인자로 받으며 기본 동작과 동일하게 N개의 하위 디렉터리를 만드나, -e 옵션이 있다면 짝수 디렉터리에 '<N-1>ssu_test.txt' 파일을 복사하며 해당 모드 권한을 원본파일의 권한과 동일하게 MODE 매크로와 chmod를 사용하여 변경
7. 반대로 -o옵션은 홀수 디렉터리에 '<N-1>ssu_test.txt' 파일을 복사

11.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <dirent.h>
#include <fcntl.h>
#include <string.h>
#include <sys/stat.h>

#define DIRECTORY_SIZE MAXNAMLEN      //디렉토리 이름 길이 매크로
#define isdigit(x) (x>='0'&&x<='9')?1:0 //숫자 판단 매크로
#define MODE S_IRUSR|S_IWUSR|S_IRGRP | S_IWGRP|S_IROTH //권한 매크로

int create_dir(int depth, char* cur_dir); //디렉토리 생성 함수
void writefile(char *in_f, char *out_f); // 파일을 복사하는 함수
void change_mod(const char *file_path); //모드를 변경하는 함수

char *fname = "ssu_test.txt"; //생성하고 복사할 파일의 기본 이름
int o_flag=0, e_flag=0; //e 옵션과 o옵션을 나타낼 플래그
int main(int argc, char *argv[]){
    int opt; //옵션인자를 받을 변수
    int depth = 0; //하위 디렉터리의 갯수를 받을 변수
    char cur_dir_name[DIRECTORY_SIZE]= {"\0",}; //현재 디렉토리 이름
    int fd;

    while((opt = getopt(argc, argv, "e:o:")) != -1)
    {
        switch(opt)
        {
            case 'e':
                for( int i = 0; i < strlen(optarg); i++)
                {
                    if(!isdigit(optarg[i]))
                    {
                        fprintf(stderr, "put the depth N number.\n");
                        exit(1);
                    }
```

```
                depth = optarg[i] - '0';
            }
            e_flag = 1;
            break;

        case 'o':
            for ( int i = 0; i < strlen(optarg); i++)
            {
                if(!isdigit(optarg[i]))
                {
                    fprintf(stderr, "put the depth N number.\n");
                    exit(1);
                }
                depth = optarg[i] - '0';
            }
            o_flag = 1;
            break;

        case '?' :
            break;
    }
}

if( argc < 3)
{
    printf("%s\n" , argv[argc-1]);
    for(int i = 0 ; i < strlen(argv[argc-1]) ; i++)
        if(isdigit(argv[argc-1][i]))
        {
            depth = argv[argc-1][i] - '0';
            break;
        }
}
else
    fprintf( stderr, "too many argv\n");

if ((fd = creat(fname, MODE)) < 0) {
    fprintf(stderr, "creat error for %s \n", fname);
    exit(1);
}
else
    close(fd);

if ( (fd = creat(fname, 0666)) < 0 )
{
    fprintf(stderr, "mkdir error\n");
    exit(1);
}

if (mkdir("0", 0755)) //기본 디렉터리 생성 , 권한 755
```

```
    {
        fprintf(stderr, "mkdir error\n");
        exit(1);
    }
    strcpy(cur_dir_name,"0"); //현재 디렉터리 이름 저장
    create_dir(depth,cur_dir_name);
    exit(0);
}

int create_dir(int depth, char* cur_dir)
{
    struct stat dir_st;
    int i = 0 ;
    char tmp_filename[MAXNAMLEN] = {'\0',};
    while (cur_dir[i] != '\0') i++;
    cur_dir[i] = '/'; i++;
    cur_dir[i] = depth+'0'; //깊이 저장

    if ( stat(cur_dir, &dir_st) < 0){
        if (mkdir(cur_dir, 0777) < 0){
            fprintf(stderr, " fail to make directory.\n");
            return -1;
        }
    }

    strcat(tmp_filename, cur_dir);
    if(o_flag && (depth%2 == 0) )
    {
        writefile(fname, strcat(tmp_filename, fname));
        change_mod(tmp_filename);
    }
    else if (e_flag && (depth%2 == 1) )
    {
        writefile(fname, strcat(tmp_filename, fname));
        change_mod(tmp_filename);
    }
    else if (!o_flag && !e_flag)
    {
        writefile(fname , strcat(tmp_filename , fname));
    }
    if ( depth == 0)
        return 0;
    return create_dir(depth-1, cur_dir);
}

void writefile(char *in_f, char *out_f)
{
    int in_fd, out_fd;
    int read_count;
    char buf[MAXNAMLEN];
```

```
    in_fd = open(in_f, O_RDONLY);
    out_fd = open(out_f, O_WRONLY|O_CREAT, S_IRUSR|S_IWUSR);

    while((read_count = read(in_fd, buf, sizeof(buf))) >0)
        write(out_fd, buf, read_count);
}

void change_mod(const char *file_path)
{
    if (chmod(file_path, MODE) <0) // file 을 MODE 로 모드전환
        fprintf(stderr, "chmod error %s\n", file_path);
}
```

실행결과 <예시> o 옵션

```
root@localhost:/home/oslab# ./a.out -o5
root@localhost:/home/oslab# tree -fa 0
tree -fa 0
0 └── 0/5
    ├── 0/5/4
    │   └── 0/5/4/3
    │       ├── 0/5/4/3/2
    │       │   └── 0/5/4/3/2/1
    │       │       ├── 0/5/4/3/2/1/0
    │       │       └── 0/5/4/3/2/1/0ssu_test.txt
    │       └── 0/5/4/3/2ssu_test.txt
    └── 0/5/4ssu_test.txt
6 directories, 3 files

root@localhost:/home/oslab# ls -al
drwxrwxr-x  3 ssuos ssuos  4096  4월 26 16:44 .
drwxr-xr-x 27 ssuos ssuos  4096  4월 26 16:42 ..
drwxr-xr-x  3 ssuos ssuos  4096  4월 26 16:44 0
-rwxrwxr-x  1 ssuos ssuos 12256  4월 26 16:32 a.out
-rw-rw-r--  1 ssuos ssuos     0  4월 26 16:44 ssu_test.txt

root@localhost:/home/oslab# ls -al
drwxrwxr-x 3 ssuos ssuos 4096  4월 26 16:44 .
drwxr-xr-x 3 ssuos ssuos 4096  4월 26 16:44 ..
drwxrwxr-x 3 ssuos ssuos 4096  4월 26 16:44 4
-rw-rw-r-- 1 ssuos ssuos    0  4월 26 16:44 4ssu_test.txt  <-권한이 원본과 같음을 확인
```

실행결과 <예시> e옵션

```
root@localhost:/home/oslab# ./a.out -e6
root@localhost:/home/oslab# tree -fa 0
0 └── 0/6
    ├── 0/6/5
    │   └── 0/6/5/4
    │       ├── 0/6/5/4/3
    │       │   └── 0/6/5/4/3/2
    │       │       ├── 0/6/5/4/3/2/1
    │       │       │   └── 0/6/5/4/3/2/1/0
    │       │       └── 0/6/5/4/3/2/1ssu_test.txt
```

```
    |       └── 0/6/5/4/3ssu_test.txt
    └── 0/6/5ssu_test.txt
7 directories, 3 files
```

실행결과 <예시>

```
root@localhost:/home/oslab# ./a.out 5
root@localhost:/home/oslab# tree -fa 0
0 ├── 0/5
|   ├── 0/5/4
|   |   ├── 0/5/4/3
|   |   |   ├── 0/5/4/3/2
|   |   |   |   ├── 0/5/4/3/2/1
|   |   |   |   |   ├── 0/5/4/3/2/1/0
|   |   |   |   |   |   └── 0/5/4/3/2/1/0ssu_test.txt
|   |   |   |   |   └── 0/5/4/3/2/1ssu_test.txt
|   |   |   |   └── 0/5/4/3/2ssu_test.txt
|   |   |   └── 0/5/4/3ssu_test.txt
|   |   └── 0/5/4ssu_test.txt
|   └── 0/5ssu_test.txt
6 directories, 6 files

root@localhost:/home/oslab# ls -al 0/
drwxr-xr-x 3 ssuos ssuos 4096  4월 26 16:46 .
drwxrwxr-x 3 ssuos ssuos 4096  4월 26 16:46 ..
drwxrwxr-x 3 ssuos ssuos 4096  4월 26 16:46 5
-rw------- 1 ssuos ssuos    0  4월 26 16:46 5ssu_test.txt     <-권한이 원본과 다름을 확인
```

12. 다음 주어진 ssu_test.txt 파일을 한 줄씩 읽어 줄의 번호와 함께 읽은 줄을 출력하는 프로그램을 작성하시오.

< 조 건 >

1. 주어진 변수를 그대로 사용할 것, 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. 본 프로그램의 실행파일은 1개의 인자만 받도록 할 것
4. open(), read(), close()를 각 1번 씩 사용하고 write()는 3번사용 할 것
5. open()함수와 write()함수에 대한 에러 처리를 위해 fprintf() 4번만 사용할 것
6. 입력 텍스트 파일이 변경되어도 정상적으로 작동되어야 할 것

ssu_test.txt

```
Linux System Programming!
UNIX System Programming!
Linux Mania
Unix Mania

UNIX System Programming!
Linux Mania
Unix Mania
```

12.c

```
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include<unistd.h>
```

```c
#define BUF_MAX 4

int main(int argc, char *argv[])
{
    int fd;                     // 읽을 파일 디스크립터
    size_t n;                   // read()의 리턴값
    int count=0;                // 읽은 파일의 라인수
    char buf[2], cbuf[5];   // read()에서 읽는 버퍼와 출력할 버퍼

    if(argc != 2) {
        fprintf(stderr,"usage : %s file", argv[0]);
        exit(1);
    }
    fd_r = open(argv[1], O_RDONLY);

    if(fd == -1) {
        fprintf(stderr,"open error for %s ",argv[1]);
        exit(1);
    }

    sprintf(cbuf, "%d  ", count);
    if(write(1, cbuf, BUF_MAX) != BUF_MAX)
        fprintf(stderr,"write error\n");

    while((n = read(fd, buf, 1)) > 0) {
        if(write(1, buf, n) != n)
        fprintf(stderr,"write error\n");
        if(buf[0] == '\n') {
            count++;
            sprintf(cbuf, "%d  ", count);
            if(write(1, cbuf, BUF_MAX) != BUF_MAX)
                fprintf(stderr,"write error");
        }
    }

    if(n == -1)
        fprintf(stderr,"read error");
    close(fd);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out not_exist_file
open error for not_exist_file
root@localhost:/home/oslab# ./a.out ssu_test.txt ssu_line.txt
usage : ./a.out file
root@localhost:/home/oslab# ./a.out ssu_test.txt
0  Linux System Programming!
1  UNIX System Programming!
2  Linux Mania
3  Unix Mania
4
5  UNIX System Programming!
6  Linux Mania
7  Unix Mania
```

13. 다음 주어진 ssu_blank.txt 파일을 읽어, 공백으로 단어를 구분하여, 각 단어를 줄 단위로 ssu_line.txt에 저장하고 단어의 개수를 출력하는 프로그램을 작성하시오.

< 조 건 >

1. 주어진 변수를 그대로 사용할 것, 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. open() 2번, read()와 setbuf()를 각 1번 씩 사용할 것
4. dup2()는 3번만 사용 할 것
5. open(), creat(), read()에 대한 에러 처리를 위해 fprintf() 3번만 사용할 것
6. write()와 fwrite()를 사용하지 않고 dup2()를 사용하여 텍스트 사용할 것
7. 문자의 개수 출력결과가 나와야 할 것
8. 입력 텍스트 파일이 변경되어도 정상적으로 작동되어야 할 것

<ssu_blank.txt>

```
berry grape raisin apple watermelon grapefruit pomelo
```

13.c

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>

void main()
{
    char *fname = "ssu_line.txt";
    char *frname= "ssu_blank.txt";
    int fd_w, fd_r;          //쓰기 파일 디스크립터, 읽을 파일 디스크립터
    size_t length            //read()의 리턴값
    int wordcnt = 1;        //읽은 파일의 문자의 개수
    char buf[50];            //read()에서 읽는 버퍼
    int i = 0;                //for(),while() 문에 조건문 사용

    if((fd_w =open(fname,O_WRONLY|O_CREAT)) < 0){
        fprintf(stderr,"creat error for %s \n", fname);
        exit(1);
    }

    if((fd_r= open(frname,O_RDONLY |O_CREAT)) < 0){
        fprintf(stderr,"open error for %s \n",frname);
        exit(1);
    }

    setbuf(stdout,NULL);
    if ((read(fd_r,buf,50)) < 0){
        fprintf(stderr,"read range over");
        exit(1);
    }

    close(fd_r);
    dup2(1,fd_r);
    dup2(fd_w,1);
    close(fd_w);
```

```
    while(1)
    {
        printf("%c",buf[i]);
        if(buf[i]==' '){
            printf("\n");
            wordcnt++;
        }
        if(buf[i]=='\0')
            break;
        i++;
    }

    dup2(fd_r,1);
    printf("wordcount = %d \n",wordcnt);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
wordcount = 7
root@localhost:/home/oslab# cat ssu_line.txt
berry
grape
raisin
apple
watermelon
grapefruit
pomelo
```

14. 다음 프로그램 실행 시 프로그램의 인자로 주어진 파일의 타입이 나오는 프로그램을 작성하시오.

< 조 건 >

1. 주어진 변수를 그대로 사용할 것, 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. 프로그램 실행 시 프로그램의 인자는 정규파일, 디렉토리, 케릭터 특수, 블록 특수, FIFO, 심볼릭 링크 파일만 있다고 가정
4. access()는 1번 사용할 것
5. 파일 타입은 주어진 print_file_type()를 통해 실행결과와 같이 출력할 것

14.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>

void print_file_type(struct stat *statbuf)
{
    char *str;

    if(S_ISREG(statbuf->st_mode))
        str = "regular";
    else if(S_ISDIR(statbuf->st_mode))
        str = "directory";
```

```
        else if(S_ISCHR(statbuf->st_mode))
                str = "character special";
        else if(S_ISBLK(statbuf->st_mode))
                str = "block special";
        else if(S_ISFIFO(statbuf->st_mode))
                str = "FIFO";
        else if(S_ISLNK(statbuf->st_mode))
                str = "symbolic link";
        else if(S_ISSOCK(statbuf->st_mode))
                str = "socket";
        else
                str = "unknown mode";

        printf("%s is %ld bytes\n", str, statbuf->st_size);
}

int main(int argc, char *argv[])
{
        struct stat statbuf;
        int i;

        for(i = 1; i < argc; i++){
                if(access(argv[i], F_OK) < 0){
                        fprintf(stderr, "%s doesn't exit.\n", argv[i]);
                        continue;
                }

                if(lstat(argv[i], &statbuf) < 0){
                        fprintf(stderr, "error\n");
                        exit(1);
                }
                print_file_type(&statbuf);
        }

        exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# echo "oslab" > regular
root@localhost:/home/oslab# mkdir directory
root@localhost:/home/oslab# sudo mknod character c 3 10
root@localhost:/home/oslab# sudo mknod block b 3 10
root@localhost:/home/oslab# sudo mknod fifo p
root@localhost:/home/oslab# ln -s regular symbolic
root@localhost:/home/oslab# ls
a.out  block  character  directory  fifo  regular  symbolic
root@localhost:/home/oslab# ./a.out regular directory character block fifo symbolic
regular is 6 bytes
directory is 4096 bytes
character special is 0 bytes
block special is 0 bytes
FIFO is 0 bytes
symbolic link is 7 bytes
```

15. 파일을 이동하거나 이름을 변경하는 프로그램을 작성하시오.

< 조 건 >

1. 주어진 변수를 그대로 사용할 것, 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. link(), unlink()를 각 1번씩 사용할 것
4. link(), unlink()의 에러 처리를 위해 fprintf()를 2번만 사용할 것
5. 프로그램의 실행은 "./a.out [arg1] [arg2]" 와 같이 반드시 인자 두 개를 받아야 하며, arg1에 해당하는 파일을 arg2에 해당하는 경로로 이동하거나 이름을 변경해야 할 것
6. 변경 전, 후 파일의 inode 번호를 알아내기 위해 stat 구조체의 멤버 변수를 사용할 것

15.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>

int main(int argc, char *argv[])
{
    struct stat statbuf;
    ino_t inode;

    if(argc != 3){
        fprintf(stderr, "argc != 3\n");
        exit(1);
    }

    if((stat(argv[1], &statbuf)) < 0){
        fprintf(stderr, "stat error\n");
        exit(1);
    }

    printf("%s's inode = %lu -> ", argv[1], statbuf.st_ino);

    if(link(argv[1], argv[2]) == -1){
        fprintf(stderr, "link() error\n");
        exit(1);
    }

    if(unlink(argv[1]) < 0){
        fprintf(stderr, "ulink() error\n");
        exit(1);
    }

    if((stat(argv[2], &statbuf)) < 0){
        fprintf(stderr, "stat error\n");
        exit(1);
    }

    printf("%s's inode = %lu\n", argv[2], statbuf.st_ino);

    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ls
a.out   a.txt   subdir
root@localhost:/home/oslab# ./a.out a.txt subdir/a.txt
a.txt's inode = 23204489 -> subdir/a.txt's inode = 2304489
root@localhost:/home/oslab# ls
a.out   subdir
```

16. 다음 주어진 ssu_answer.txt에 있는 답 파일을 채점하여 그 결과를 ssu_res.txt에 저장하는 프로그램을 작성하시오.

< 조 건 >

1. 주어진 변수를 그대로 사용할 것, 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. fopen(), fgets(), fclose()를 각 2번씩 사용할 것
4. fopen()의 에러 처리를 위해 fprintf()를 2번만 사용할 것
5. 결과의 표준 출력을 위해 printf()를 1번, ssu_res.txt 파일의 출력을 위해 fputs()를 1번 사용할 것

<ssu_answer.txt>

```
Jung DJ
1234123412
Lee SS
1233423413
Hong GD
2233412424
```

16.c

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

#define BUFFER_SIZE 256
#define STUDENT_NUM 3
#define Q_SIZE 10

typedef struct _student {
    char name[10];
    int score ;
    char res[BUFFER_SIZE];
} Student;

char answer[BUFFER_SIZE] = "1233423413"; //test's answer

int main(void)
{
    char *ssu_answer = "ssu_answer.txt"; // 학생들의 답안이 있는 텍스트 파일
    char *ssu_res = "ssu_res.txt"; // 학생들의 답안 채점 결과가 있는 텍스트 파일
    char tmp_score[BUFFER_SIZE];
    FILE *fp;
    int i, j= 0;
    Student list[STUDENT_NUM];

    if ((fp = fopen(ssu_answer,"rb")) == NULL) {
        fprintf(stderr, "fopen error for %s\n",ssu_answer);
        exit(1);
```

```
}

for(j = 0 ; j < STUDENT_NUM ; j++) {
    list[j].score = 0;
    if (fgets(list[j].name, BUFFER_SIZE, fp) == NULL) {
        fprintf(stderr, "fgets error for %s\n", ssu_answer);
        exit(1);
    }
    i = 0;

    while((list[j].name[i] !='\n'))
        i++;

    list[j].name[i] = '\0';

            if(fgets(list[j].res, BUFFER_SIZE, fp) == NULL){
                fprintf(stderr, "fgets error for %s\n", ssu_answer);
                exit(1);
            }

    i = 0;

    while((list[j].res[i]  !='\n')) i++;

    list[j].res[i] = '\0';
    i = 0 ;

    for( ; i < Q_SIZE ; i++) {
        if (list[j].res[i] == answer[i]) {
            list[j].score += 10;
            list[j].res[i] = 'O';
        }
        else
            list[j].res[i] = 'X';
    }
    printf("Student name : %s , score : %d , res : %s \n", list[j].name, list[j].score, list[j].res);
}

fclose(fp);
if ((fp = fopen(ssu_res, "wb")) == NULL) {
    fprintf(stderr, "fopen error for %s \n",ssu_res);
    exit(1);
}

for (i = 0 ; i < STUDENT_NUM ; i++) {

        char temp[BUFFER_SIZE];

        sprintf(temp, "%s !%d! %s\n", list[i].name, list[i].score, list[i].res);

        if(fputs(temp, fp) == EOF){
            fprintf(stderr, "fputs error for %s\n", "student info");
            exit(1);
```

```
                }
        }

        fclose(fp);
        exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
Student name : Jung DJ , score : 70 , res : OOOXXOOOOX
Student name : Lee SS , score : 80 , res : OOOOOOOOOO
Student name : Hong GD , score : 50 , res : XOOOOXXOXX
root@localhost:/home/oslab# cat ssu_res.txt
Jung DJ !70! OOOXXOOOOX
Lee SS !80! OOOOOOXXOO
Hong GD !50! XOOOOXXOXX
```

17. 프로그램 상에서 주어진 Person 구조체 내용을 파일에 저장한 후, 해당 파일을 읽어 두 번 출력을 하는 프로그램을 작성하시오.

< 조 건 >

1. 주어진 변수를 그대로 사용할 것, 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. 구조체 내용을 저장하기 위한 ftest.txt 파일을 위해 fopen()를 쓰기 모드로 1번 사용할 것
4. 구조체의 내용을 파일에 저장하기 위하여 fwrite()를 1번 사용할 것
5. ftest.txt 파일을 읽기 위하여 fopen()를 읽기 모드로 1번 사용할 것
6. fopen()의 에러 처리를 위해 fprintf()를 2번 사용할 것
7. fread(), fclose()를 각 2번, fseek()를 1번 사용할 것

17.c

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

typedef struct _person {
        char name[10];
        int age;
        double height;
} Person;

int main(void)
{
        FILE *fp;
        int i, res;
        Person ary[3] = {{"Hong GD", 500, 175.4},
                {"Lee SS", 350,180.0},
                {"King SJ", 500, 178.6}};
        Person tmp;

        if((fp = fopen("ftest.txt", "wb")) == NULL) {
                fprintf(stderr, "fopen error!\n");
                exit(1);
        }
```

```
    fwrite(&ary, sizeof(ary), 1, fp);
    fclose(fp);

    if ((fp = fopen("ftest.txt", "rb")) == NULL) {
        fprintf(stderr , "fopen error!\n");
        exit(1);
    }
    printf("[ First print]\n");

    while (!feof(fp)) {
        if ((res = fread(&tmp, sizeof(Person), 1, fp)) != 1)
            break;
        printf("%s %d %.2lf\n", tmp.name, tmp.age, tmp.height);
    }
    fseek(fp, 0, SEEK_SET);
    printf("[ Second print]\n");

    while (!feof(fp)) {
        if ((res = fread(&tmp, sizeof(Person), 1, fp)) != 1)
            break;
        printf("%s %d %.2lf\n", tmp.name, tmp.age, tmp.height);
    }
    fclose(fp);
    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
[ First print]
Hong GD 500 175.40
Lee SS 350 180.00
King SJ 500 178.60
[ Second print]
Hong GD 500 175.40
Lee SS 350 180.00
King SJ 500 178.60
root@localhost:/home/oslab#
```

18. system() 함수를 사용하여 diff 명령어를 실행하는 프로그램을 작성하시오.

< 조 건 >

1. 주어진 변수를 그대로 사용할 것, 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. 프로그램 실행 시 프로그램의 인자는 diff를 위한 파일 2개를 받을 것
4. 프로그램 실행 시 프로그램의 인자로 받은 두 파일의 정보를 얻기 위해 lstat()를 사용할 것
5. lstat()의 에러 처리를 위해 fprintf()를 1번 사용할 것
6. 프로그램 실행 시 프로그램의 인자로 받은 두 파일 중 일반 파일이 아닌 파일이 있으면 fprintf()를 사용하여 에러 처리할 것
7. 프로그램 실행 시 프로그램의 인자로 받은 두 파일이 모두 일반 파일이라면 system()함수를 사용하여 diff를 실행할 것. 단, diff의 인자의 처리를 위하여 strcpy()를 1번, strcat()를 3번 사용할 것

<ssu_input_1.txt>

```
ABCDE
```

FGHIJ
KLMNO
PQRST
UVWXYZ

<ssu_input_2.txt>

ABCDE
fghij
KLMNO
pqrst
UVWXYZ

18.c

```
#include <stdio.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#define BUF_SIZE 1024

int main(int argc, char *argv[])
{
	char buf[BUF_SIZE];
	struct stat statbuf_1, statbuf_2;

	if(argc != 3)
	{
		fprintf(stderr, "usage : %s <filename1> <filename2>\n", argv[0]);
		exit(1);
	}

	if(lstat(argv[1], &statbuf_1) < 0)
	{
		fprintf(stderr, "lstat error %s\n", argv[1]);
		exit(1);
	}
	if(lstat(argv[2], &statbuf_2) < 0)
	{
		fprintf(stderr, "lstat error %s\n", argv[2]);
		exit(1);
	}

	if(!S_ISREG(statbuf_1.st_mode) || !S_ISREG(statbuf_2.st_mode))
	{
		fprintf(stderr, "not a regular file\n");
		exit(1);
	}

	strcpy(buf, "diff ");
	strcat(buf, argv[1]);
	strcat(buf, " ");
	strcat(buf, argv[2]);

	system(buf);
```

```
    exit(1);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out ssu_input_1.txt ssu_input_2.txt
2c2
< FGHIJ
---
> fghij
4c4
< PQRST
---
> pqrst
```

19. 다음은 1번 설계과제의 일부분을 수정한 것이다. “문제번호.c”를 컴파일 후 실행파일은 “문제번호.stdexe”으로 생성하고 실행파일의 실행결과를 “문제번호.stdout”으로 저장한다.

< 조 건 >

1. 주어진 변수를 그대로 사용할 것. 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. 프로그램 실행 시 프로그램의 인자는 문제번호.c 형태임
4. access()와 creat()의 에러처리를 위해 fprintf()를 2번 사용할 것
5. 문제번호 추출을 위해 string.h에 정의된 함수를 사용할 것
6. 실행파일의 이름을 지정하기 위해 컴파일 시 -o 옵션을 사용할 것
7. 출력의 결과를 리다이렉션하기 위해 dup2()를 사용할 것

a.c

```
#include <stdio.h>
int main()
{
    printf("oslab\n");
}
```

19.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>

#define BUFLEN 512

void make_files(char *filename);

int main(int argc, char *argv[])
{
    char buf[BUFLEN];

    if(argc < 2){
        fprintf(stderr, "usage : %s <file1>\n", argv[0]);
        exit(1);
    }
```

```
    if(access(argv[1], F_OK) < 0){
        fprintf(stderr, "%s doesn't exist.\n", argv[1]);
        exit(1);
    }

    make_files(argv[1]);
}

void make_files(char *filename)
{
    int fd;
    char name[15];
    char buf[BUFLEN];

    memset(name, 0, sizeof(buf));
    strncpy(name, filename, strlen(filename) - strlen(strrchr(filename, '.')));

    sprintf(buf, "gcc -o %s.stdexe %s.c", name, name);
    system(buf);

    sprintf(buf, "%s.stdout", name);
    if((fd = creat(buf, 0666)) < 0){
        fprintf(stderr, "creat error for %s\n", buf);
        return;
    }

    sprintf(buf, "./%s.stdexe", name);
    dup2(fd, 1);
    system(buf);

    close(fd);
}
```

실행결과

```
root@localhost:/home/oslab# ls
a.c   a.out
root@localhost:/home/oslab# ./a.out a.c
root@localhost:/home/oslab# ls
a.c   a.out   a.stdexe   a.stdout
root@localhost:/home/oslab# cat a.stdout
oslab
```

20. getc(), putc(), getchar(), putchar(), fgets(), fputs(), scanf(), printf()를 사용하여 입출력하는 프로그램을 작성하시오.

< 조 건 >

1. 주어진 변수를 그대로 사용할 것. 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. getc(), putc(), getchar(), putchar(), fgets(), fputs(), scanf(), printf()를 각 1번씩 사용할 것
4. 입·출력 함수의 에러처리를 위해 fprintf()를 3번 사용할 것
5. 각 주석에 해당하는 입·출력 함수를 사용할 것
6. 실행결과와 같이 문자열을 출력할 것

20.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#define BUFFER_SIZE 1024

int main(void)
{
        char character;
        char buf[BUFFER_SIZE];

        //getchar(), putchar() 사용하여 표준 입·출력 수행
        while((character = getchar()) != EOF)
                if(putchar(character) == EOF){
                        fprintf(stderr, "standard output error\n");
                        exit(1);
                }
        //getc(), putc() 사용하여 표준 입·출력 수행
        while((character = getc(stdin)) != EOF)
                if(putc(character, stdout) == EOF){
                        fprintf(stderr, "standard output error\n");
                        exit(1);
                }

        if(ferror(stdin)){
                fprintf(stderr, "standard input error\n");
                exit(1);
        }

        //fgets(), fputs() 사용하여 표준 입·출력 수행
        while(fgets(buf, BUFFER_SIZE, stdin) != NULL)
                if(fputs(buf, stdout) == EOF){
                        fprintf(stderr, "standard output error\n");
                        exit(1);
                }

        if(ferror(stdin)){
                fprintf(stderr, "standard input error\n");
                exit(1);
        }

        memset(buf, 0, sizeof(buf));

        //scanf(), printf() 사용하여 표준 입·출력 수행
        scanf(" %[^\n]", buf);
        printf("%s\n", buf);

        exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
hello getchar, putchar!
```

```
hello getchar, putchar!
^D -> (Ctrl+D)을 의미
hello getc, putc!
hello getc, putc!
^D
hello fgets, fputs!
hello fgets, fputs!
^D
hello scanf, printf!
hello scanf, printf!
```

21. 다음 프로그램은 SIGUSR1 시그널을 BLOCK 후 해당 시그널을 보냈을 때 pending되어 있는 시그널을 확인한다. 아래 조건과 실행결과를 보고 프로그램을 완성하시오.

< 조 건 >

1. 각 함수가 정의된 해더파일을 정확히 쓸 것
2. sigemptyset(), sigaddset(), sigprocmask()를 각각 한 번씩 사용할 것
3. 자식 프로세스와 부모 프로세스 각각 sigpending()을 한 번씩 사용하고 sigismember()로 pending된 시그널을 검사할 것

21.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

void ssu_signal(int signo){
    printf("SIGUSR1 catched!!\n");
}

int main(void)
{
    pid_t pid;
    sigset_t sigset;
    sigset_t pending_sigset;

    sigemptyset(&sigset);
    sigaddset(&sigset, SIGUSR1);
    sigprocmask(SIG_BLOCK,  &sigset, NULL);

    signal(SIGUSR1, ssu_signal);
    kill(getpid(), SIGUSR1);

    if((pid = fork()) < 0){
        fprintf(stderr, "fork error\n");
        exit(1);
    }
    else if(pid == 0){
        sigpending(&pending_sigset);

        if(sigismember(&pending_sigset, SIGUSR1))
```

```
            printf("child : SIGUSR1 pending\n");
    }
    else{
        sigpending(&pending_sigset);

        if(sigismember(&pending_sigset, SIGUSR1))
            printf("parent : SIGUSR1 pending\n");
    }
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
parent : SIGUSR1 pending
```

22. 다음 프로그램은 두 개의 쓰레드를 생성하여 producer 쓰레드는 buf에 값을 넣는 작업을 하는 것을, consumer 쓰레드는 buf에 있는 값을 사용하여 총 합을 구한 후 출력하는 것을 보여준다. 아래 조건과 실행결과를 보고 프로그램을 완성하시오.

< 조 건 >

1. 각 함수가 정의된 해더파일을 정확히 쓸 것
2. 뮤텍스 관련 변수 mutex와 cond1 및 cond2 는 메크로를 사용하여 초기화 할 것
3. 프로그램의 실행이 끝난 후 mutex와 cond1 및 cond2 변수를 해제할 것
4. producer 쓰레드와 consumer 쓰레드에서 pthread_mutex_lock(), pthread_mutex_unlock(), pthread_cond_signal(), pthread_cond_wait()을 각각 한 번씩 사용할 것

22.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond1 = PTHREAD_COND_INITIALIZER;
pthread_cond_t cond2 = PTHREAD_COND_INITIALIZER;

int length;
int buf[100];

void *ssu_thread_producer(void *arg){
    int i;

    for(i = 1; i <= 300; i++){
        pthread_mutex_lock(&mutex);
        buf[length++] = i;

        if(i % 100 == 0)
            pthread_cond_signal(&cond2);
        if(length == 100)
            pthread_cond_wait(&cond1, &mutex);

        pthread_mutex_unlock(&mutex);
    }
}
```

```
void *ssu_thread_consumer(void *arg){
    int i;
    int sum = 0;

    for(i = 1; i <= 300; i++){
        pthread_mutex_lock(&mutex);

        if(i % 100 == 0)
            pthread_cond_signal(&cond1);
        if(length == 0)
            pthread_cond_wait(&cond2, &mutex);

        sum += buf[--length];
        pthread_mutex_unlock(&mutex);
    }

    printf("%d\n", sum);
}

int main(void){
    pthread_t producer_tid, consumer_tid;

    pthread_create(&producer_tid, NULL, ssu_thread_producer, NULL);
    pthread_create(&consumer_tid, NULL, ssu_thread_consumer, NULL);
    pthread_join(producer_tid, NULL);
    pthread_join(consumer_tid, NULL);

    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&cond1);
    pthread_cond_destroy(&cond2);

    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
45150
```

23. 다음 프로그램은 /var/log/system.log 파일에 자신의 pid를 로그 메시지로 남기는 디몬 프로세스를 생성한다. 아래 조건과 실행결과를 보고 프로그램을 완성하시오.

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
2. 디몬 프로그램 작성 규칙에 따라 작성할 것
3. openlog()의 옵션은 없고, facility는 LOG_LPR을 사용할 것
4. 로그는 에러 상태로 출력하고 출력을 한 다음에는 닫을 것
5. getdtablesize()를 사용하여 모든 파일 디스크립터를 닫을 것
6. 로그를 출력한 후 5초간 정지 후 프로그램을 종료할 것

23.c

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <unistd.h>

int ssu_daemon_init(void);

int main(void)
{
	printf("daemon process initialization\n");

	if(ssu_daemon_init() < 0){
		fprintf(stderr, "ssu_daemon_init failed\n");
		exit(1);
	}
	openlog("ex8", 0, LOG_LPR);
	syslog(LOG_ERR, "My pid is %d", getpid());
	sleep(5);
	closelog();

	exit(0);
}

int ssu_daemon_init(void) {
	int fd, maxfd;
	pid_t pid;

	if((pid = fork()) < 0){
		fprintf(stderr, "fork error\n");
		exit(1);
	}
	else if(pid != 0)
		exit(0);

	setsid();
	signal(SIGTTIN, SIG_IGN);
	signal(SIGTTOU, SIG_IGN);
	signal(SIGTSTP, SIG_IGN);

	maxfd = getdtablesize();
	for(fd = 0; fd < maxfd; fd++)
		close(fd);

	umask(0);
	chdir("/");

	fd = open("/dev/null", O_RDWR);
	dup(0);
	dup(0);

	return 0;
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
daemon process initialization
root@localhost:/home/oslab# ps -e ¦ grep a.out
3409      ?      00:00:00 a.out
root@localhost:/home/oslab# tail -1 /var/log/syslog
Jan 17 18:32:59 oslab ex8: My pid is 3409
(PID는 바뀔 수 있음)
```

24. 다음 프로그램은 alarm() 호출을 통해서 시간을 설정하고 지정된 시간이 지나면 SIGALRM에 대한 시그널 핸들러를 통해서 문자열과 값을 출력하는 것을 보여준다. 아래 조건과 실행 결과를 바탕으로 프로그램을 작성하시오.

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
2. signal()을 sigaction()으로 변경하여 ssu_signal_handler()를 등록할 것
3. sigaction() 사용을 위한 추가 변수는 사용 가능함
4. 실행 결과는 코드 변경 후에도 동일해야 함
5. alarm()을 main() ssu_signal_handler()에서 각각 1번씩 사용하여 1초마다 ssu_signal_handler()가 실행되게 할 것

24.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

void ssu_signal_handler(int signo);

int count = 0;

int main(void)
{
    struct sigaction sig_act;

    sigemptyset(&sig_act.sa_mask);
    sig_act.sa_flags = 0;
    sig_act.sa_handler = ssu_signal_handler;
    sigaction(SIGALRM,&sig_act,NULL);

    alarm(1);

    while(1);

    exit(0);
}

void ssu_signal_handler(int signo) {
    printf("alarm %d\n", count++);
    alarm(1);

    if(count > 3)
        exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ./a.out
alarm 0
alarm 1
alarm 2
alarm 3
```

25. 다음 프로그램은 kill() 호출을 통해서 main()함수의 인자로 들어온 프로세스 ID에 시그널을 보내는 것을 보여준다. 아래 조건과 실행결과를 보고 프로그램을 완성하시오.

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
2. ./a.out [-signal] <pid>의 형태로 입력을 받으며 signal의 위치는 변경될 수 없고 '-' 로 시작해야 함
3. 프로그램 실행 시 지정된 시그널이 없는 경우 SIGTERM을 기본으로 사용함
4. 프로그램 실행 시 시그널은 하나만 지정 가능하고 프로세스 ID는 여러 개 지정 가능함
5. 시그널이 정수로 입력된 경우 모든 시그널은 사용 가능하고 문자로 입력될 경우 SIGKILL, SIGINT, SIGUSR1만 사용 가능함
6. 시그널의 입력은 대수문자 구분이 없음

25.c

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <ctype.h>

#define pid_max 20

int main(int argc, char *argv[])
{
    int pid[pid_max];
    int pid_count = 0;
    int sig_pst = 0;            //signal position in argv[]
    int n = 0;

    if (argc <1) {
        fprintf(stderr, "usage: %s <-signal> <pid> ...\n", argv[0]);
        exit(1);
    }

    for(int i=1; i<argc;i++){
        if(strstr(argv[i],"-")!=NULL){
            if(sig_pst!=0){      //check <-signal> use times
                fprintf(stderr,"<-signal> only can use 1 time\n");
                exit(1);
            }
            sig_pst = i;
        }
    }

    if(strstr(argv[1],"-")!=NULL){
```

```
        pid_count = argc-2;

        for(int i=0;i<pid_count;i++){
            if(sig_pst == i+1 &&sig_pst!=1)
                continue;

            pid[n] = atoi(argv[2+i]);
            printf("pid %d\n",pid[n]);

            if(sig_pst != 1){   //EX: ./ssu_kill pid -signal
                printf("SIGTERM\n");
                kill(pid[n],SIGTERM);
            }
            else if(strcasecmp(argv[1],"-SIGINT")==0)
                kill(pid[n],SIGINT);
            else if(strcasecmp(argv[1],"-SIGUSR1")==0)
                kill(pid[n],SIGUSR1);
            else if(strcasecmp(argv[1],"-SIGKILL")==0)
                kill(pid[n],SIGKILL);
            else if(isdigit(argv[1][1])) //-signal is consisted by number [-9,-10]
                kill(pid[n],atoi(argv[1]+1));

            n++;
        }
    }
    else{
        pid_count = argc-1;
        printf("sig_pst: %d\n",sig_pst);
        for(int i=0;i<pid_count;i++){
            if(sig_pst == i+1)
                continue;

            pid[n] = atoi(argv[1+i]);
            kill(pid[n],SIGTERM);
            n++;
        }
    }

    exit(0);
}
```

실행 결과1

```
root@localhost:/home/oslab# ./ssu_loop &
[1] 32212
root@localhost:/home/oslab# ./a.out 32212
[1]+  Terminated                 ./ssu_loop
```

실행 결과2

```
root@localhost:/home/oslab# ./ssu_loop & ./ssu_loop & ./ssu_loop &
[2] 32220
[3] 32221
[4] 32222
root@localhost:/home/oslab# ./a.out -sigusr1 32220 32221 32222
```

```
[4]+  User defined signal 1    ./ssu_loop
[2]-  User defined signal 1    ./ssu_loop
[3]+  User defined signal 1    ./ssu_loop
```

실행 결과3

```
root@localhost:/home/oslab# ./ssu_loop & ./ssu_loop &
[2] 32229
[3] 32230
root@localhost:/home/oslab# ./a.out -10 32229 32230
[3]+  User defined signal 1    ./ssu_loop
[2]+  User defined signal 1    ./ssu_loop
```

실행 결과4

```
root@localhost:/home/oslab# ./ssu_loop & ./ssu_loop &
[1] 32293
[2] 32294
root@localhost:/home/oslab# ./a.out 32293 -sigint 32294
[1]-  Terminated                ./ssu_loop
[2]+  Terminated                ./ssu_loop
```

실행 결과5

```
root@localhost:/home/oslab# ./a.out 32293 -sigint 32294 -sigkill
<-signal> only can use 1 time
```

26. 다음 프로그램은 두 개의 쓰레드를 생성하여 permutation 결과를 출력하는 것을 보여준다. main()에서 ssu_thread1()과 ssu_thread2()를 각각 호출하여 pthread_cond_signal()과 pthread_cond_wait()을 통해 번갈아가며 permutation 순열을 출력한다. 아래 조건과 실행결과를 보고 프로그램을 완성하시오.

< 조 건 >

1. 각 함수가 정의된 해더파일을 정확히 쓸 것
2. mutex와 cond는 메크로를 사용하여 초기화를 할 것
3. 프로그램의 실행이 끝난 후 mutex와 cond의 해제를 할 것
4. scanf()를 한 번 사용하여 입력을 받으며 10P4와 같은 형태로 입력을 받을 것
5. 잘못된 입력이 있을 경우 다시 입력을 받을 것
6. 생성된 쓰레드는 각각 pthread_cond_wait(), pthread_cond_signal()을 두 번씩 사용하고 pthread_mutex_lock(), pthread_mutex_unlock()을 한 번씩 사용할 것
7. result는 결과를 저장하고 buf는 sprintf()를 사용하여 수식을 저장할 것

26.c

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

void *ssu_thread1(void *arg);
void *ssu_thread2(void *arg);

pthread_mutex_t mutex1
pthread_mutex_t mutex2
pthread_cond_t cond1
pthread_cond_t cond2

int count = 0;
int input1 = 0, input2 = 0;
int result;
char buf[BUFSIZ];
```

```
int main(void)
{
    pthread_t tid1, tid2;
    int status;

    if(pthread_create(&tid1, NULL, ssu_thread1, NULL) != 0){
        fprintf(stderr, "pthread_create error\n");
        exit(1);
    }

    if(pthread_create(&tid2, NULL, ssu_thread2, NULL) != 0){
        fprintf(stderr, "pthread_create error\n");
        exit(1);
    }

    while(1){
        //입력, 양수P양수의 형태만 가능
        scanf("%dP%d", &input1, &input2);

        if((input1 > 0 && input2 > 0) && input1 >= input2){
            pthread_cond_signal(&cond1);
            break;
        }
        else{
            input1 = 0;
            input2 = 0;
            while(getchar() != '\n');
        }
    }

    pthread_join(tid1, (void *)&status);
    pthread_join(tid2, (void *)&status);

    pthread_mutex_destroy(&mutex1);
    pthread_mutex_destroy(&mutex2);
    pthread_cond_destroy(&cond1);
    pthread_cond_destroy(&cond2);

    printf("complete \n");
    exit(0);
}

void *ssu_thread1(void *arg){
    while(1){
        pthread_mutex_lock(&mutex1);

        if(input1 == 0 && input2 == 0)
            pthread_cond_wait(&cond1, &mutex1);
```

```
        if(input2 == count){
            pthread_cond_signal(&cond2);
            break;
        }

        if(count == 0){
            sprintf(buf, "%d", input1);
            result = input1;
            count++;
            printf("Thread 1 : %s=%d\n", buf, result);
        }
        else if(count % 2 == 0){
            sprintf(buf, "%s x %d", buf, input1-count);
            result *= (input1-count);
            count++;
            printf("Thread 1 : %s=%d\n", buf, result);
        }

        pthread_cond_signal(&cond2);
        pthread_cond_wait(&cond1, &mutex1);
        pthread_mutex_unlock(&mutex1);
    }

    return NULL;
}

void *ssu_thread2(void *arg){
    while(1){
        pthread_mutex_lock(&mutex2);

        if(input1 == 0 && input2 == 0)
            pthread_cond_wait(&cond2, &mutex2);

        if(input2 == count){
            pthread_cond_signal(&cond1);
            break;
        }

        if(count % 2 == 1){
            sprintf(buf, "%s x %d", buf, input1-count);
            result *= (input1-count);
            count++;
            printf("Thread 2 : %s=%d\n", buf, result);
        }

        pthread_cond_signal(&cond1);
        pthread_cond_wait(&cond2, &mutex2);
        pthread_mutex_unlock(&mutex2);
    }
```

```
    return NULL;
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
12p8
12P22
12P8
Thread 1 : 12=12
Thread 2 : 12 x 11=132
Thread 1 : 12 x 11 x 10=1320
Thread 2 : 12 x 11 x 10 x 9=11880
Thread 1 : 12 x 11 x 10 x 9 x 8=95040
Thread 2 : 12 x 11 x 10 x 9 x 8 x 7=665280
Thread 1 : 12 x 11 x 10 x 9 x 8 x 7 x 6=3991680
Thread 2 : 12 x 11 x 10 x 9 x 8 x 7 x 6 x 5=19958400
complete
```

27. 다음 프로그램은 파일과 파일의 접근 권한 모드를 메인함수의 인자로 입력받아 입력된 파일의 모드를 변경한다. 아래 조건과 실행결과를 보고 프로그램을 완성하시오.

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
2. ./a.out MODE FILE의 형태로 입력을 받고 MODE는 8진수임
3. ./a.out -b MODE FILE의 형태로 입력을 받을 경우 MODE는 2진수임
4. MODE는 한 개만 지정 가능하고 FILE은 여러 개 지정 가능함
5. getopt()를 한 번 사용하여 옵션 처리를 할 것
6. MODE가 잘 못 입력된 경우 에러 처리를 할 것

27.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <string.h>

int main(int argc, char *argv[]){
    mode_t mode=0;
    int digit;
    int flag_b=0;
    int i, j;
    char mode_arg[30] = {0, };

    if(argc < 3){
        fprintf(stderr, "usage : ssu_chmod MODE FILE...\n");
        exit(1);
    }

    strcpy(mode_arg, argv[1]);

    while((i = getopt(argc, argv, "b:")) > 0){
        switch(i){
            case 'b':
```

```
                flag_b = 1;
                strcpy(mode_arg, optarg);
                break;
            default:
                fprintf(stderr, "unknown option\n");
                exit(1);
        }
    }

    for(i = strlen(mode_arg)-1, j=0; i >= 0; i--, j++){
        digit = mode_arg[i] - '0';

        if(flag_b){
            if(digit < 0 || digit > 1){
                fprintf(stderr, "MODE error : %s\n", mode_arg);
                exit(1);
            }

            mode = mode | (digit << j);
        }
        else{
            if(digit < 0 || digit > 7){
                fprintf(stderr, "MODE error : %s\n", mode_arg);
                exit(1);
            }

            mode = mode | (digit << j*3);
        }
    }

    for(i = flag_b==0?2:3; i < argc; i++)
        if(chmod(argv[i], mode) < 0)
            fprintf(stderr, "chmod() error\n");

    return 0;
}
```

실행결과

```
root@localhost:/home/oslab# ls -al a.txt b.txt
---------- 1 kym kym 0  6월 14 11:12 a.txt
---------- 1 kym kym 0  6월 14 11:45 b.txt
root@localhost:/home/oslab# ./a.out 4744 a.txt b.txt
root@localhost:/home/oslab# ls -al a.txt b.txt
-rwsr--r-- 1 kym kym 0  6월 14 11:12 a.txt
-rwsr--r-- 1 kym kym 0  6월 14 11:45 b.txt
```

실형결과

```
root@localhost:/home/oslab# ls -al a.txt
---------- 1 kym kym 0  6월 14 11:12 a.txt
root@localhost:/home/oslab# ./a.out -b 0100111110110 a.txt
root@localhost:/home/oslab# ls -al a.txt
-rwsrw-rw- 1 kym kym 0  6월 14 11:12 a.txt
```

실행결과

```
root@localhost:/home/oslab# ./a.out 944 a.txt
MODE error : 944
```

28. 다음 프로그램은 디렉터리를 복사한다. 아래 조건과 실행결과를 보고 프로그램을 완성하시오.

< 조 건 >

1. 다음 프로그램은 일반 파일은 복사하지 않고 디렉터리 파일만 복사하는 프로그램임
2. 명령어는 ‘./a.out [-d][N] [SOURCE][TARGET]’ 의 형식을 따름
3. [SOURCE]는 복사대상 디렉터리, [TARGET]은 복사된 디렉터리
4. [-d] 옵션은 필수 옵션임
5. [N] 옵션은 1~9까지 숫자를 입력할 수 있으며, 입력 된 N개만큼 자식 프로세스(fork())를 생성함을 의미하고 N개의 자식 프로세스들이 하위 디렉터리를 복사함. 단
   (1) 하위 디렉터리의 개수 < N이면 하위 디렉터리 개수만큼의 자식 프로세스가 디렉터리를 복사하고 (N-하위 디렉터리 개수)의 자식프로세스는 생성하지 않음
   (2) 하위 디렉터리의 개수 > N이면 첫 번째 자식 프로세스는 첫 번째 하위 디렉터리를 복사, 두 번째 자식프로세스는 두 번째 하위 디렉터리를 복사 하는 등 N개의 자식프로세스들이 순서대로 하나의 하위 디렉터리를 복사하고 부모 프로세스가 나머지 하위 디렉터리 복사
   (3) 하위 디렉터리의 개수 = N이면 N개의 자식 프로세스가 N개의 하위 디렉터리를 순서대로 복사
6. 디렉터리 복사를 완료한 후에는 [SOURCE] 디렉터리 이름과 [TARGET] 디렉터리 이름을 표준출력
7. 생성된 자식 프로세스들은 각자 복사한 디렉터리의 이름과 pid를 표준출력
8. [TARGET] 디렉터리가 이미 존재하는 경우 덮어씀
9. [SOURCE]와 [TARGET] 이 동일한 이름의 디렉터리인 경우 오류 처리
10. scandir() 과 같은 메모리 동적할당 관련 함수들을 사용할 경우 반드시 메모리 회수를 해야 함
11. 주어진 변수 외에 추가적인 변수 사용, 구조체 변경, 불필요한 헤더파일 삽입은 허용하지 않음

28.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <dirent.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>

#define BUFFER_SIZE 1024
#define ERROR_MSG “ssu_cp error\nusage : cp [-d][N] (1 <= N < 10)\n“
#define isdigit(x) (x>='0'&&x<='9')?1:0 //숫자 판단 매크로

void check_flag(int argc, char *argv[]);
static int filter(const struct dirent *dirent);
void copy_dir(char *source, char *target);

int flag_d ,N;

int main(int argc, char *argv[])
{
        char opt;
        struct timeval start, end;
        char source[PATH_MAX+1];
        char target[PATH_MAX+1];
        struct stat source_info, target_info;
        while((opt = getopt(argc, argv, “d:“)) != -1) //옵션 인자처리
        {
```

```
                switch(opt)
                {
                        case 'd':
                                for( int i = 0; i < strlen(optarg); i++)
                                {
                                        if(!isdigit(optarg[i]))
                                        {
                                                fprintf(stderr, "put the depth N number.\n");
                                                exit(1);
                                        }
                                N = optarg[i] - '0';
                                }
                                flag_d = 1;
                                break;

                        case '?' :
                                break;
                }
        }
        strcpy(source, argv[argc-2]);
        strcpy(target, argv[argc-1]);

        printf("target : %s\n", target);
        printf("source : %s", source);

        if (access(source, F_OK) < 0) {
                fprintf(stderr, "\nssu_cp: %s: No such file or directory\n", source);
                printf(ERROR_MSG);
                exit(1);
        }
        if (lstat(source, &source_info) < 0) {
                fprintf(stderr, "\nlstat error for %s\n", source);
                exit(1);
        }
        if (flag_d) {
                if (!(1<=N && N<10)) {
                        fprintf(stderr, "not invalid N size\n");
                        printf(ERROR_MSG);
                        exit(1);
                }
        }
        umask((mode_t)0000);
if ( flag_d) {
                int length = strlen(source);
                if (source[length - 1] == '/') {
                        if (length != 1)
                                source[length - 1] = '\0';
                }
                length = strlen(target);
                if (target[length - 1] == '/') {
                        if (length != 1)
                                target[length - 1] = '\0';
                }
                copy_dir(source, target);
```

```
        }
        exit(0);
}

void copy_dir(char *source, char *target) {
        struct stat source_info;
        struct stat target_info;
        struct dirent **namelist;
        char source_name[PATH_MAX];
        pid_t check_pid;
        int status;
        int source_length;
        int target_length;
        int count;
        int i;

        if (stat(source, &source_info) < 0) {
                fprintf(stderr, "stat error for %s\n", source);
                return;
        }
        printf( "src : %s , dst : %s \n",source, target);
        source_length = strlen(source);
        target_length = strlen(target);
        if (access(target, F_OK) == 0) {
                if (lstat(target, &target_info) < 0) {
                        fprintf(stderr, "lstat error for %s\n", target);
                        return;
                }
                if (!S_ISDIR(target_info.st_mode)) { //일반파일 일 때
                        fprintf(stderr, "ssu_cp: cannot overwrite non-directory '%s' with directory '%s'\n", target,
source);
                        return;
                }
                else {
                        char *temp_source = (char *)malloc(sizeof(char) * (strlen(source) + 1));
                        strcpy(temp_source, source);
                        char *token = strtok(temp_source, "/");
                        while(token != NULL) {
                                strcpy(source_name, token);
                                token = strtok(NULL, "/");
                        }
                        if ((count = scandir(target, &namelist, filter, alphasort)) == -1) {
                                fprintf(stderr, "%s Directory Scan Error : %s\n", target, strerror(errno));
                                return;
                        }
                        for (i = 0; i < count; i++) { //target directory안에 source directory이름과 같은 file or
directory가 있는지 확인
                                if (strcmp(source_name, namelist[i]->d_name) == 0) {
                                        strcat(target, "/");
                                        strcat(target, source_name);
                                        if (lstat(target, &target_info) < 0) {
                                                fprintf(stderr, "lstat error for %s\n", target);
                                                return;
                                        }
```

```
                                        if (!S_ISDIR(target_info.st_mode)) {
                                                fprintf(stderr, "ssu_cp: cannot overwrite non-directory '%s' with directory '%s'\n", target, source);
                                                return;
                                        }
                                        else {
                                                target_length = strlen(target);
                                                goto copy;
                                        }
                                }
                        }
                        for (i = 0; i < count; i++)
                                free(namelist[i]);
                        free(namelist);
                        strcat(target, "/");
                        strcat(target, source_name);
                        copy_dir(source, target);
                        free (temp_source);
                }
        }
        else {
                if (mkdir(target, source_info.st_mode) < 0) {
                        fprintf(stderr, "mkdir error for %s\n", target);
                        return;
                }
copy:
                if ((count = scandir(source, &namelist, filter, alphasort)) == -1) {
                        fprintf(stderr, "%s Directory Scan Error : %s\n", source, strerror(errno));
                        return;
                }
                for (i = 0; i < count; i++) {
                        strcat(source, "/");
                        strcat(source, namelist[i]->d_name);
                        if (stat(source, &source_info) < 0) {
                                fprintf(stderr, "stat error for %s, pid : %d\n", source, getpid());
                                return;
                        }
                        strcat(target, "/");
                        strcat(target, namelist[i]->d_name);
                        if (!S_ISDIR(source_info.st_mode)) {
                                target[target_length] = '\0';
                        }
                        else {
                                if (flag_d) {
                                        if (1 <= N) {
                                                N--;
                                                check_pid = fork();

                                                if (check_pid == 0) {
                                                        printf("Directory %s copied by Process id : %d\n", source, getpid());
                                                }
                                                if (check_pid > 0) {
                                                        source[source_length] = '\0';
```

```
                                                target[target_length] = '\0';
                                                continue;
                                        }
                                }
                        }
                        copy_dir(source, target);
                        target[target_length] = '\0';
                        if (flag_d) {
                                if (check_pid == 0)
                                        exit(0);
                        }
                }
                source[source_length] = '\0';
        }
        for (i = 0; i < count; i++)
                free(namelist[i]);
        free(namelist);
        if (flag_d) {
                if (check_pid > 0)
                        waitpid(check_pid, &status, 0);
        }
    }
}
static int filter(const struct dirent *dirent) {
    if (!(strcmp(dirent->d_name, ".")) || !(strcmp(dirent->d_name, "..")))
        return 0;
    else
        return 1;
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out -d3 test2 test3
target : test3
source : test2
src : test2 , dst : test3
Directory test2/test3 copied by Process id : 32466
src : test2/test3 , dst : test3/test3
Directory test2/test1 copied by Process id : 32465
src : test2/test1 , dst : test3/test1
Directory test2/test1/aa copied by Process id : 32467
src : test2/test1/aa , dst : test3/test1/aa

root@localhost:/home/oslab# tree -fa test2 test3
test2
├── test2/test1
│   └── test2/test1/aa
└── test2/test3
test3
├── test3/test1
│   └── test3/test1/aa
└── test3/test3
6 directories, 0 files
```

29. 다음 프로그램은 이 프로그램은 인자로 받은 디렉토리 내의 모든 일반파일에 대한 생성, 수정, 삭제에 대해 모니터링한

다. (설계과제 3번 ssu_backup의 답안과 동일한 구조로 구현했지만 파일의 백업을 제외하고 모니터링만을 수행) 모든 일반 파일에 대해 각 파일마다 하나의 쓰레드를 통해 모니터링을 수행하며, 각 쓰레드는 파일의 변화에 대해 표준출력으로 출력하기 때문에 동기화를 해야 한다. 아래 조건과 실행결과를 보고 프로그램을 완성하시오.

< 조 건 >

1. 프로그램의 인자로 디렉토리명을 받음. 단, 절대경로와 상대경로 디렉터리를 모두 처리해야 함
2. 명령어는 ‘./a.out [DIRECTORY]’ 형식을 따름
3. 인자가 디렉토리가 아니거나 존재하지 않는 파일(일반 파일 포함)일 경우 stderr으로 에러 메시지 출력
4. 처음 프로그램을 실행했을 때 모니터링 디렉토리에 있는 파일에 대해서는 별도의 메시지를 출력하지 않음
5. 프로그램이 실행된 후, 모니터링 디렉토리와 하위 디렉토리에 있는 모든 일반파일에 대한 생성, 수정, 삭제 행위를 모니터링하며, 관련 행위의 메시지를 표준출력으로 출력
6. 일반파일이 생성될 때 마다 하나의 모니터링 쓰레드가 만들어지고 그 파일에 대해 모니터링을 수행
7. 모니터링 쓰레드는 모니터링 대상이 되는 파일이 삭제될 때까지 종료하지 않음
8. 모니터링 대상이 되는 파일이 수정되어 st_mtime이 변화되었을 때, 수정 메시지를 표준출력으로 출력
9. 동시에 여러 개의 파일에 대해 모니터링 쓰레드가 실행되어야 하기 때문에 생성, 수정, 삭제 행위 관련 메시지 표준출력 시 반드시 mutex를 사용하여 동기화
10. 생성, 수정, 삭제 행위 메시지는 다음의 형식을 반드시 준수하여야 함, 아래와 같이 출력 메시지는 경로가 아닌 파일명을 사용해야 함
- 생성 시 : [MMDD HH:MM:SS] FILE is created [size:--/mtime:MMDD HH:MM:SS]
- 수정 시 : [MMDD HH:MM:SS] FILE is modified [size:--/mtime:MMDD HH:MM:SS]
- 삭제 시 : [MMDD HH:MM:SS] FILE is deleted
11. 주어진 함수 구조, 변수 및 구조체는 변경 가능

29.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>

#define MAX_DIR 64
#define  true 1
#define false 0
typedef int bool;

char pathname[PATH_MAX];
char dirname[PATH_MAX];
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

typedef struct thread_struct
{
        struct thread_struct *next;
        pthread_t tid;
        char *data;
} ssu_thread;

ssu_thread *HEAD = NULL, *TAIL = NULL;
```

```
char *dir_list[MAX_DIR];
int dir_cnt = 0;
bool isnewfile = false;
int startfile_fin = 0;

void directory_monitor(char *link);
ssu_thread *make_thread(char *data);
void thread_function(void *arg);
void monitor_function(char *name);
void print_log(char *name, int size, time_t mt, int index);
int check_list(char *data);
void delete_list(char *name);

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

        if (argc != 2){
                fprintf(stderr, "usage : ./<<<MW2>>> <directory>\n");
                exit(1);
        }

        realpath(argv[1], pathname);
        strcpy(dirname, argv[1]);

        if (access(argv[1], F_OK) != 0){
                fprintf(stderr, "usage : Target directory not exist\n");
                exit(1);
        }

        struct stat st;
        stat(pathname, &st);

        if(!S_ISDIR(st.st_mode)){
                fprintf(stderr, "usage : Target must be directory\n");
                exit(1);
        }

        time_t old = 0, new = 0;
        time_t dir_time_now[MAX_DIR] = {0};
        time_t dir_time_new;
        while (1){
                stat(pathname, &st);
                if (access(pathname, F_OK) != 0){
                        print_log(pathname, 0, 0, 3);
                        exit(0);
                }

                new = st.st_mtime;
                if (new != old){
                        directory_monitor(pathname);
                        old = new;
```

```
                }

                for (int i = 0; i < dir_cnt; i++){
                        if (access(dir_list[i], F_OK) == 0) {
                                stat(dir_list[i], &st);
                                dir_time_new = st.st_mtime;
                                if (dir_time_now[i] != dir_time_new)
                                {
                                        directory_monitor(dir_list[i]);
                                        dir_time_now[i] = dir_time_new;
                                }
                        }
                }

                if(!isnewfile){
                        int startfile_cnt = 0;
                        ssu_thread *cur;
                        for (cur = HEAD; cur != NULL; cur = cur->next)
                        {
                                startfile_cnt++;
                        }
                        while(startfile_cnt != startfile_fin)
                                ;
                        isnewfile = true;
                }
        }
}

void directory_monitor(char *link)
{
        struct dirent **namelist;
        int cont_cnt = scandir(link, &namelist, NULL, alphasort);

        for (int i = 0; i < cont_cnt; i++)
        {
                if ((!strcmp(namelist[i]->d_name, ".") || !strcmp(namelist[i]->d_name, ".."))
                        continue;

                char *resource = (char *)malloc(PATH_MAX);
                struct stat src;
                sprintf(resource, "%s/%s", link, namelist[i]->d_name);
                stat(resource, &src);

                if (S_ISDIR(src.st_mode))
                {
                        int k;
                        for (k = 0; k < dir_cnt; k++)
                        {
                                if (strcmp(dir_list[k], resource) == 0)
                                        break;
```

```
                        }
                        if (k == dir_cnt)
                        {
                                dir_list[k] = resource;
                                dir_cnt++;
                        }
                        directory_monitor(resource);
                }
                else
                {
                        if (check_list(resource))
                        {
                                if (HEAD == NULL)
                                {
                                        HEAD = make_thread(resource);
                                        TAIL = HEAD;
                                }
                                else
                                {
                                        TAIL->next = make_thread(resource);
                                        TAIL = TAIL->next;
                                }
                        }
                }
        }

        for (int i = 0; i < cont_cnt; i++)
                free(namelist[i]);
        free(namelist);

        return;
}

ssu_thread *make_thread(char *data){
        ssu_thread *temp = (ssu_thread *)malloc(sizeof(ssu_thread));
        temp->data = data;
        temp->next = NULL;

        if(pthread_create(&(temp->tid), NULL, (void *)(&thread_function), (void *)data) != 0){
                fprintf(stderr, "pthread_crate error\n");
        }
        return temp;
}

void thread_function(void *arg){
        char *data = (char *)arg;
        monitor_function(data);
}

void monitor_function(char *name){
```

```
        struct stat src_sc;
        time_t old = 0, new = 0;
        int i = 0;
        int check = 0;

        while (true)
        {
                if (access(name, F_OK) != 0){
                        print_log(name, 0, new, 3);
                        delete_list(name);
                        pthread_exit(0);
                }

                stat(name, &src_sc);
                new = src_sc.st_mtime;

                if (new == old){
                        continue;
                }

                if (old == 0){
                        if(!isnewfile)
                                ;
                        else
                                print_log(name, src_sc.st_size, new, 1);
                }
                else if(old != new){
                        print_log(name, src_sc.st_size, new, 2);
                }


                old = new;
                if(!isnewfile)
                        startfile_fin++;


        }
}

void print_log(char *name, int size, time_t mt, int index){
        time_t timer = time(NULL);
        struct tm *t = localtime(&timer);
        char temp[14];
        sprintf(temp, "%02d%02d %02d:%02d:%02d", t->tm_mon + 1, t->tm_mday, t->tm_hour, t->tm_min, t->tm_sec);
        struct tm *tt = localtime(&mt);
        int k;

        for (k = strlen(name) - 1; k >= 0; k--)
        {
                if (name[k] == '/')
```

```
                        break;
        }
        k++;

        pthread_mutex_lock(&mutex);
        switch (index)
        {
        case 1:
                printf("[%s] %s is created [size:%d/mtime:%02d%02d %02d:%02d:%02d]\n", temp, name + k, size,
tt->tm_mon + 1, tt->tm_mday, tt->tm_hour, tt->tm_min, tt->tm_sec);
                break;
        case 2:
                printf("[%s] %s is modified [size:%d/mtime:%02d%02d %02d:%02d:%02d]\n", temp, name + k, size,
tt->tm_mon + 1, tt->tm_mday, tt->tm_hour, tt->tm_min, tt->tm_sec);
                break;
        case 3:
                printf("[%s] %s is deleted\n", temp, name + k);
                break;
        }
        pthread_mutex_unlock(&mutex);
}


int check_list(char *data)
{
        ssu_thread *cur;
        for (cur = HEAD; cur != NULL; cur = cur->next)
        {
                if (strcmp(cur->data, data) == 0)
                        return 0;
        }
        return 1;
}

void delete_list(char *name){
        ssu_thread *cur;
        for (cur = HEAD; cur != NULL; cur = cur->next)
        {
                if (strcmp(cur->data, name) == 0){
                        cur->data[0] = '\0';
                        return;
                }
        }
        return;
}
```

실행결과 // 테스트를 위해 컨트롤 터미널 2개가 필요, 하나의 터미널은 프로그램 실행, 하나는 파일 수정

```
root@localhost:/home/oslab# ls
1.txt  2.txt  dir2
root@localhost:/home/oslab# ls dir2
3.txt
root@localhost:/home/oslab# vi 4.txt                    -> (1) dir1/4.txt 생성
```

```
root@localhost:/home/oslab# ls
1.txt  2.txt  4.txt  dir2
root@localhost:/home/oslab# vi 1.txt                  -> (2) dir1/1.txt 수정
root@localhost:/home/oslab# rm ./dir2/3.txt          -> (3) dir2/3.txt 삭제
root@localhost:/home/oslab1# vi ./dir2/4.txt          -> (4) dir2/4.txt 생성
root@localhost:/home/oslab# rm *                      -> (5) dir1/1.txt,2.txt,4.txt 동시삭제
```

실행결과 // 파일수정시 vi에디터를 사용할 경우 .swp 파일에 대한 메시지가 출력될 수 있음(고려하지 않아도 됨)

```
root@localhost:/home/oslab# ./a.out dir1
[0612 01:40:32] 4.txt is created [size:11/mtime:0612 01:40:32]     -> (1)
[0612 01:40:35] 1.txt is modified [size:23/mtime:0612 01:40:36]    -> (2)
[0612 01:40:37] 3.txt is deleted                                    -> (3)
[0612 01:40:40] 4.txt is created [size:8/mtime:0612 01:40:40]      -> (4)
[0612 01:40:42] 1.txt is deleted                                    -> (5) *순서가 바뀔 수 있음
[0612 01:40:42] 2.txt is deleted                                    -> (5) *순서가 바뀔 수 있음
[0612 01:40:42] 4.txt is deleted                                    -> (5) *순서가 바뀔 수 있음
```

30. 다음 프로그램은 setjmp()와 longjmp()를 호출하여 함수 경계를 넘나드는 분기를 수행할 때 변수의 타입에 따른 값을 확인하는 것을 보여준다. 아래 조건과 실행결과를 보고 프로그램을 완성하시오.

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
2. 변수의 값을 출력하기 위해 printf()를 두 번 사용하고 ret_val의 출력을 위해 printf()를 한 번 사용할 것
3. setjmp()로 분기를 위해 longjmp()를 한 번 사용할 것
4. ssu_func()를 재귀적으로 호출할 것

30.c

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

void ssu_func(int loc_var, int loc_volatile, int loc_register);

int count = 0;
static jmp_buf glob_buffer;

int main(void)
{
    register int loc_register;
    volatile int loc_volatile;
    int loc_var;
    int ret_val;

    loc_var = 10;  loc_volatile = 11; loc_register = 12;

    if ((ret_val = setjmp(glob_buffer)) != 0) {
        printf("after longjmp, loc_var = %d, loc_volatile = %d, loc_register = %d\n", loc_var, loc_volatile, loc_register);

        printf("ret_val : %d\n", ret_val);
        exit(0);
    }
```

```
    loc_var = 80; loc_volatile = 81; loc_register = 82;
    ssu_func(loc_var, loc_volatile, loc_register);
    exit(0);
}

void ssu_func(int loc_var, int loc_volatile, int loc_register) {
    if (count == 3)
        longjmp(glob_buffer, 1);
    count++;
    printf("ssu_func, loc_var = %d, loc_volatile = %d, loc_register = %d\n", loc_var, loc_volatile, loc_register);

    ssu_func(loc_var + 1, loc_volatile + 1, loc_register + 1);

    printf("ssu_func, loc_var = %d, loc_volatile = %d, loc_register = %d\n", loc_var, loc_volatile, loc_register);
}
```

실행결과

```
root@localhost:/home/oslab# gcc 20.c -O2
root@localhost:/home/oslab# ./a.out
ssu_func, loc_var = 80, loc_volatile = 81, loc_register = 82
ssu_func, loc_var = 81, loc_volatile = 82, loc_register = 83
ssu_func, loc_var = 82, loc_volatile = 83, loc_register = 84
after longjmp, loc_var = 10, loc_volatile = 81, loc_register = 12
ret_val : 1
```

31. 두 개의 쓰레드의 실행 순서를 확인하는 프로그램을 작성하시오 아래의 조건과 실행결과를 보고 프로그램을 완성하시오.

< 조 건 >

1. 주어진 변수를 그대로 사용할 것, 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. mutex와 cond의 초기화는 매크로를 사용할 것
4. pthread_create() 2번 사용하여 2개의 쓰레드를 생성할 것
5. pthread_join()을 2번 사용하여 생성된 쓰레드가 종료될 때까지 기다리게 할 것
6. ssu_thread1()은 시그널이 오기 전까지 블록 상태가 되기 위해 pthread_cond_wait()를 1번 사용할 것
7. ssu_thread2()은 cond로 시그널을 보내기 위해 pthread_cond_signal()을 1번 사용할 것
8. pthread_mutex_lock() 2번, pthread_mutex_unlock() 2번 사용하여 공유변수 glo_val을 번갈아 사용할 것
9. glo_val이 VALUE_STOP1보다 작거나 VALUE_STOP2보다 클 때 cond로 시그널을 보낼 것

31.c

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

#define VALUE_DONE 10
#define VALUE_STOP1 3
#define VALUE_STOP2 6

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

void *ssu_thread1(void *arg);
```

```
void *ssu_thread2(void *arg);

int glo_val = 0;

int main(void)
{
        pthread_t tid1, tid2;

        pthread_create(&tid1, NULL, &ssu_thread1, NULL);
        pthread_create(&tid2, NULL, &ssu_thread2, NULL);
        pthread_join(tid1, NULL);
        pthread_join(tid2, NULL);
        printf("final value: %d\n", glo_val);
        exit(0);
}

void *ssu_thread1(void *arg)
{
        while(1){
                pthread_mutex_lock(&lock);
                pthread_cond_wait(&cond, &lock);
                glo_val++;
                printf("global value ssu_thread1: %d\n", glo_val);
                pthread_mutex_unlock(&lock);

                if(glo_val >= VALUE_DONE)
                        return NULL;
        }
}

void *ssu_thread2(void *arg)
{
        while(1){
                pthread_mutex_lock(&lock);
                if(glo_val < VALUE_STOP1 || glo_val > VALUE_STOP2)
                        pthread_cond_signal(&cond);
                else{
                        glo_val++;
                        printf("global value ssu_thread2: %d\n", glo_val);
                }
                pthread_mutex_unlock(&lock);

                if(glo_val >= VALUE_DONE)
                        return NULL;
        }
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
global value ssu_thread1: 1
global value ssu_thread1: 2
global value ssu_thread1: 3
global value ssu_thread2: 4
global value ssu_thread2: 5
global value ssu_thread2: 6
```

```
global value ssu_thread2: 7
global value ssu_thread1: 8
global value ssu_thread1: 9
global value ssu_thread1: 10
final value: 10
```

32. 디몬(daemon) 프로세스가 5초마다 로그 메시지를 남기게 하고 그 동작을 확인하는 프로그램을 작성하시오.

< 조 건 >

1. 주어진 변수를 그대로 사용할 것. 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. 로그 메시지를 남기기 위해 openlog(), syslog(), closelog(), sleep()을 각각 1번씩 사용할 것
4. 디몬 프로세스 생성을 위해 fork()를 1번 사용할 것
5. getpid(), setsid()를 각각 1번씩 사용하고 표준출력으로 디몬프로세스의 pid를 출력할 것
6. signal()을 3번 사용하여 작업제어와 연관된 시그널을 무시하도록 할 것
7. close()를 1번 사용하여 열려있는 모든 파일 디스크립터를 닫을 것
8. umask()를 1번 사용하여 디몬이 생성할 파일의 접근 허가 모드를 모두 허용하도록 할 것
9. chdir()을 1번 사용하여 현재 디렉토리를 루트 디렉토리로 설정할 것
10. open()을 1번 사용하여 "/dev/null" 파일을 읽기, 쓰기 모드로 열 것
11. dup()을 2번 사용할 것
12. ssu_daemon_init(), fork()의 에러 처리를 위해 fprintf()를 2번 사용할 것

32.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <signal.h>
#include <syslog.h>
#include <sys/stat.h>
#include <sys/types.h>

int ssu_daemon_init(void);

int main(void)
{
        printf("daemon process initialization\n");

        if(ssu_daemon_init() < 0){
                fprintf(stderr, "ssu_daemon_init failed\n");
                exit(1);
        }

        while(1){
                openlog("lpd", LOG_PID, LOG_LPR);
                syslog(LOG_ERR, "open failed lpd %m");
                closelog();
                sleep(5);
        }

        exit(0);
}
```

```
int ssu_daemon_init(void){
        pid_t pid;
        int fd, maxfd;

        if((pid = fork()) < 0){
                fprintf(stderr, "fork error\n");
                exit(1);
        }
        else if(pid != 0)
                exit(0);

        pid = getpid();
        printf("process %d running as daemon\n", pid);
        setsid();
        signal(SIGTTIN, SIG_IGN);
        signal(SIGTTOU, SIG_IGN);
        signal(SIGTSTP, SIG_IGN);
        maxfd = getdtablesize();

        for(fd = 0; fd < maxfd; fd++)
                close(fd);

        umask(0);
        chdir("/");
        fd = open("/dev/null", O_RDWR);
        dup(0);
        dup(0);
        return 0;
}
```

실행결과[Ubuntu]

```
root@localhost:/home/oslab# ./a.out
daemon process initialization
process 12279 running as daemon
root@localhost:/home/oslab# ps .ejfc ¦ grep a.out
root 12279 1 12279 12279 TS 19 21:46 ? 00:00:00 ./a.out
root 12281 12038 12280 12007 TS 19 21:46 pts/19 00:00:00 grep —color=auto a.out
root@localhost:/home/oslab# tail .1 /var/log/syslog
Jan 13 21:46:36 oslab-ZBOX-ID91 lpd[12279]: open failed lpd Bad file descriptor
```

33. fcntl()을 사용하여 Nonblock 플래그를 설정하는 프로그램을 작성하시오.

< 조 건 >

1. 주어진 변수를 그대로 사용할 것, 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. read()를 1번 사용하여 표준입력에서 읽을 것
4. fprintf()를 1번 사용하여 stderr에 read()로 읽은 byte 수를 출력할 것
5. set_flags(), clr_flags()을 각각 1번씩 사용하여 표준출력에 nonblock 플래그를 설정 및 해제할 것
6. write()를 1번 사용하여 표준출력에 read()로 읽은 내용을 쓰고 쓰여진 byte만큼 ptr 포인터를 이동하고 전체 읽은 byte수를 감소시킬 것
7. fprintf()를 1번 사용하여 stderr에 기록된 byte 수 및 errno를 출력할 것
8. nonblock 플래그 설정 및 해제를 위해 fcntl()을 set_flags(), clr_flags()에서 각각 2번씩 사용할 것
9. 비트연산자를 사용하여 nonblock 플래그를 설정 및 해제할 것
10. fcntl()의 에러처리를 위해 fprintf()를 4번 사용할 것

33.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>

void set_flags(int fd, int flags);
void clr_flags(int fd, int flags);

char buf[500000];

int main()
{
        int ntowrite, nwrite;
        char *ptr;

        ntowrite = read(STDIN_FILENO, buf, sizeof(buf));
        fprintf(stderr, "reading %d btyes\n", ntowrite);

        set_flags(STDOUT_FILENO, O_NONBLOCK);

        ptr = buf;
        while(ntowrite > 0){
                errno = 0;
                nwrite = write(STDOUT_FILENO, ptr, ntowrite);
                fprintf(stderr, "nwrite = %d, errno = %d\n", nwrite, errno);

                if(nwrite > 0){
                        ptr += nwrite;
                        ntowrite -= nwrite;
                }
        }

        clr_flags(STDOUT_FILENO, O_NONBLOCK);

        exit(0);
}

void set_flags(int fd, int flags)
{
        int val;

        if((val = fcntl(fd, F_GETFL, 0)) < 0){
                fprintf(stderr, "fcntl F_GETFL failed");
                exit(1);
        }

        val |= flags;

        if(fcntl(fd, F_SETFL, val) < 0){
                fprintf(stderr, "fcntl F_SETFL failed");
                exit(1);
```

```
        }
}

void clr_flags(int fd, int flags)
{
        int val;

        if((val = fcntl(fd, F_GETFL, 0)) < 0){
                fprintf(stderr, "fcntl F_GETFL failed");
                exit(1);
        }

        val &= ~flags;

        if(fcntl(fd, F_SETFL, val) < 0){
                fprintf(stderr, "fcntl F_SETFL failed");
                exit(1);
        }
}
```

실행결과

```
root@localhost:/home/oslab# ls .l ssu_test1.txt
-rw-r.r. 1 root root 500000 Jun 8 04:11 ssu_test1.txt
root@localhost:/home/oslab# ./a.out <ssu_test1.txt >ssu_test2.txt
reading 500000 bytes
nwrite = 500000, errno = 0
root@localhost:/home/oslab# ls .l ssu_test2.txt
-rw-r.r. 1 root root 500000 Jun 8 04:12 ssu_test2.txt
root@localhost:/home/oslab# ./a.out <ssu_test1.txt 2> ssu_test3.txt
[ssu_test3.txt]
reading 500000 bytes
nwrite = 8192, errno = 0
nwrite = -1, errno = 11
nwrite = -1, errno = 11
..
..
nwrite = -1, errno = 11
nwrite = 3840, errno = 0
nwrite = -1, errno = 11
..
..
```

34. 다음 첫 번째 인자로 받은 파일에 employee 구조체를 저장하는 프로그램이다. 파일디스크립트 fcntl()를 활용하여 플래그를 수정해서 출력의 결과가 EOF에 이어지게 하는 프로그램을 작성하시오.

< 조 건 >

1. 주어진 변수를 그대로 사용할 것. 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. 본 프로그램의 실행파일은 1개의 인자만 받도록 할 것.
4. 인자로 받은 파일을 읽기 쓰기 모드로 열기 위해 open()을 1번 사용할 것
5. fcntl()을 2번, getpid() 1번만 사용할 것
6. 비트연산자를 활용하여 플래그를 추가할 것
7. write()를 1번 사용할 것
8. fcntl(), write()에 대한 에러 처리를 위해 fprintf()를 3번 사용할 것
9. 인자로 받은 파일의 끝에 이어지게 작성할 것

34.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <error.h>
#include <fcntl.h>
#define NAMESIZE 50
#define DUMMY 0

struct employee{
        char name[NAMESIZE];
        int salary;
        int pid;
};

int main(int argc, char *argv[])
{
        struct flock lock;
        struct employee record;
        int fd, flags, length,pid;

        if(argc <2)
        {
                fprintf(stderr, "Usage :%s file \n",argv[0]);
                exit(1);
        }
        if((fd = open (argv[1],O_RDWR))<0)
        {
        fprintf(stderr, "Open error :%s file \n",argv[1]);
        exit(1);
        }
        if((flags = fcntl(fd,F_GETFL,DUMMY))==-1)
        {
                fprintf(stderr,"fcntl F_GETFL Error \n");
                exit(1);
        }
        flags |= O_APPEND;
        if(fcntl(fd,F_SETFL,flags)==-1){
                fprintf(stderr,"fcntl F_SETFL error \n");
                exit(1);
        }
```

```
        pid=getpid();
        while(1)
        {
                printf("Enter employee name : ");
                scanf("%s",record.name);
                if(record.name[0]=='.')
                        break;

                printf("Enter employee salary :");
                scanf("%d",&record.salary);
                record.pid=pid;
                length=sizeof(record);
                if(write(fd,(char *)&record,length)!=length)
                {
                        fprintf(stderr,"record write error \n");
                        exit(1);
                }
        }
        close(fd);
        exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# touch ssu_employeefile
root@localhost:/home/oslab# ./a.out ssu_employeefile
Enter employee name : BaekMaKang
Enter employee salary :3000000
Enter employee name : HanRaSan
Enter employee salary :13000000
Enter employee name : BakDooSan
Enter employee salary :1900000
Enter employee name : .
```

35. 다음 첫 번째 인자로 받은 파일에 입력받은 파일에서 수정을 원하는 레코드에 write락을 설정하고 수정하는 프로그램을 작성하시오.

< 조 건 >

1. 주어진 변수를 그대로 사용할 것. 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. 10번 문제의 ssu_employeefile을 인자로 사용할 것
4. 본 프로그램의 실행파일은 1개의 인자만 받도록 할 것
5. fcntl()을 3번 사용할 것
6. lseek()을 2번 사용할 것
7. read(), write()를 각각 1번 사용할 것
8. lock을 설정하기 위한 fcntl()의 예외 처리를 위해 perror()을 1번 사용할 것
9. 0이하의 record number 입력 받을 시 프로그램을 종료할 것
10. 데이터가 존재하지 않는 record number 입력 받을 시 lock을 해제하고 다음 입력을 받을 것
11. 기록을 시작할 때 기록할 부분을 lock하고 기록이 완료되면 unlock할 것
12. 수정된 레코드는 pid를 수정하여 저장 할 것
13. lock이 설정된 상태라면 락이 해제 될 때 까지 기다리게 할 것

35.c

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

```
#include <unistd.h>
#include <fcntl.h>

#define NAMESIZE 50

struct employee{
        char name[NAMESIZE];
        int salary;
        int pid;
};

int main(int argc, char *argv[])
{
        struct flock lock;
        struct employee record;
        int fd,recnum,pid;
        long position;

        if((fd = open(argv[1],O_RDWR)) ==-1)
        {
                perror(argv[1]);
                exit(1);
        }
        pid =getpid();
        while(1)
        {
                printf("\n Enter record number :");
                scanf("%d", &recnum);
                if(recnum<0)
                        break;
                position =recnum* sizeof(record);
                lock.l_type =F_WRLCK;
                lock.l_whence=0;
                lock.l_start= position;
                lock.l_len = sizeof(record);

                if(fcntl(fd, F_SETLKW, &lock)==-1){
                        perror(argv[1]);
                        exit(2);
                }
                lseek(fd, position, 0);
                if(read(fd, (char *)&record, sizeof(record))==0)
                {
                        printf("record %d not found \n",recnum);
                        lock.l_type=F_UNLCK;
                        fcntl(fd,F_SETLK,&lock);
                        continue;
                }
                printf("Employee : %s, salary :%d\n",record.name,record.salary);
                record.pid=pid;
                printf("Enter new salary :");
                scanf("%d", &record.salary);
                lseek(fd,position,0);
                write(fd,(char *)&record,sizeof(record));
```

```
                lock.l_type =F_UNLCK;
                fcntl(fd,F_SETLK,&lock);
        }
        close(fd);
        exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out ssu_employeefile
Enter record number :2
employee:HanRaSan, salary: 13000000
Enter new salary : 450000
Enter record number :1
employee:BakDooSan, salary: 1900000
Enter new salary : 500000
```

36. 다음 첫 번째 인자로 받은 파일의 내용을 두 번째 인자로 받은 파일의 이름으로 mmap()와 memcpy()를 이용하여 복사하는 프로그램을 작성하시오.

< 조 건 >

1. 주어진 변수를 그대로 사용할 것. 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. 본 프로그램의 실행파일은 2개의 인자만 받도록 할 것.
4. 인자로 받은 파일을 열기 위해 첫 번째 인자는 읽기모드로, 두 번째 인자는 읽기쓰기모드로 open()을 2번 사용할 것
5. fstat()을 1번, lseek()을 1번, write()를 1번 사용할 것
6. mmap()을 2번, memcpy()를 1번 사용할 것
7. open(), fstat(), lseek(), write(), mmap()에 대한 에러처리를 위해 printf()를 7번 사용할 것
8. 텍스트 파일이 변경되어도 정상적으로 작동되어야 할 것

36.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>

#define FILE_MODE (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)

int main(int argc, char *argv[])
{
        int fdin, fdout;
        void *src, *dst;
        struct stat statbuf;

        if (argc != 3) {
                printf("usage: %s <fromfile> <tofile>", argv[0]);
                exit(1);
        }

        if ((fdin = open(argv[1], O_RDONLY)) < 0) {
```

```
                printf("can't open %s for reading", argv[1]);
                exit(1);
        }

        if ((fdout = open(argv[2], O_RDWR | O_CREAT | O_TRUNC, FILE_MODE)) < 0) {
                printf("cannot creat %s for writing", argv[2]);
                exit(1);
        }

        if (fstat(fdin, &statbuf) < 0) {
                printf("fstat() error");
                exit(1);
        }

        if (lseek(fdout, statbuf.st_size - 1, SEEK_SET) == -1) {
                printf("lseek() error");
                exit(1);
        }

        if (write(fdout, "", 1) != 1) {
                printf("write() error");
                exit(1);
        }

        if ((src = mmap(0, statbuf.st_size, PROT_READ, MAP_SHARED, fdin, 0)) == MAP_FAILED) {
                printf("mmap() error for input");
                exit(1);
        }

        if ((dst = mmap(0, statbuf.st_size, PROT_READ | PROT_WRITE, MAP_SHARED, fdout, 0)) == MAP_FAILED) {
                printf("mmap() error for output");
                exit(1);
        }

        memcpy(dst, src, statbuf.st_size);
        exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# vim ssu_test.txt
root@localhost:/home/oslab# ./a.out ssu_test.txt ssu_test_1.txt
root@localhost:/home/oslab# ls -al ssu_test.txt ssu_test_1.txt
-rw-r--r-- 1 root root 26 6월 9 10:48 ssu_test.txt
-rw-r--r-- 1 root root 26 6월 9 10:48 ssu_test_1.txt
```

37. 다음 pthread_cond_signal()을 호출하여 피보나치 수열을 출력하는 프로그램을 작성하시오.

< 조 건 >

1. 주어진 변수를 그대로 사용할 것. 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. mutex와 cond의 초기화는 매크로를 사용할 것
4. main() 함수 내에서는 pthread_create() 2번, pthread_cond_signal()을 1번, pthread_join()을 2번 사용할 것
5. ssu_thread1() 함수 내에서 pthread_mutex_lock(), pthread_mutex_unlock()을 각각 1번, pthread_cond_wait(), pthread_cond_signal()을 각각 2번 사용할 것
6. ssu_thread2() 함수 내에서 pthread_mutex_lock(), pthread_mutex_unlock()을 각각 1번, pthread_cond_wait(), pthread_cond_signal()을 각각 2번 사용할 것
7. ssu_thread1()와 ssu_thread2()를 실행하는 Thread1, Thread2가 번갈아 가면서 출력되어야 함.

37.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

void *ssu_thread1(void *arg);
void *ssu_thread2(void *arg);

pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex2 = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond1 = PTHREAD_COND_INITIALIZER;
pthread_cond_t cond2 = PTHREAD_COND_INITIALIZER;

int count = 0;
int input = 0;
int t1 = 0, t2 = 0;

int main(void)
{
        pthread_t tid1, tid2;
        int status;

        if (pthread_create(&tid1, NULL, ssu_thread1, NULL) != 0) {
                fprintf(stderr, "pthread_create error\n");
                exit(1);
        }

        if (pthread_create(&tid2, NULL, ssu_thread2, NULL) != 0) {
                fprintf(stderr, "pthread_create error\n");
                exit(1);
        }

        while (1) {
                printf("2개 이상의 개수 입력 : ");
                scanf("%d", &input);

                if(input >= 2) {
                        pthread_cond_signal(&cond1);
                        break;
                }
        }
```

```
        pthread_join(tid1, (void *)&status);
        pthread_join(tid2, (void *)&status);

        printf("complete \n");
        exit(0);
}

void *ssu_thread1(void *arg) {
        while (1) {
                pthread_mutex_lock(&mutex1);

                if (input < 2){
                        pthread_cond_wait(&cond1, &mutex1);
                }
                if (input == count) {
                        pthread_cond_signal(&cond2);
                        break;
                }

                if (count == 0) {
                        t2++;
                        count++;
                        printf("Thread 1 : %d\n", t1);
                }
                else if (count % 2 == 0) {
                        t1 += t2;
                        count++;
                        printf("Thread 1 : %d\n", t1);
                }

                pthread_cond_signal(&cond2);
                pthread_cond_wait(&cond1, &mutex1);
                pthread_mutex_unlock(&mutex1);
        }
        return NULL;
}

void *ssu_thread2(void *arg) {
        while (1) {
                pthread_mutex_lock(&mutex2);

                if (input < 2){
                        pthread_cond_wait(&cond2, &mutex2);
                }

                if (input == count) {
                        pthread_cond_signal(&cond1);
                        break;
                }

                if (count == 1) {
                        count++;
                        printf("Thread 2 : %d\n", t2);
```

```
                }
                else if (count % 2 == 1) {
                        t2 += t1;
                        count++;
                        printf("Thread 2 : %d\n", t2);
                }

                pthread_cond_signal(&cond1);
                pthread_cond_wait(&cond2, &mutex2);
                pthread_mutex_unlock(&mutex2);
        }
        return NULL;
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
2개 이상의 개수 입력 : 20
Thread 1 : 0
Thread 2 : 1
Thread 1 : 1
Thread 2 : 2
Thread 1 : 3
Thread 2 : 5
Thread 1 : 8
Thread 2 : 13
Thread 1 : 21
Thread 2 : 34
Thread 1 : 55
Thread 2 : 89
Thread 1 : 144
Thread 2 : 233
Thread 1 : 377
Thread 2 : 610
Thread 1 : 987
Thread 2 : 1597
Thread 1 : 2584
Thread 2 : 4181
complete
```

38. 다음은 2번 설계과제의 일부분을 수정한 것이다. argv 인자를 이용해 q1_Makefile 을 생성해내는 프로그램을 작성하시오.

< 조 건 >

1. 주어진 변수를 그대로 사용할 것. 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. q1_Makefile 생성을 위해 읽기 쓰기 모드로 open()을 1번만 사용할 것. 단, q1_Makefile은 항상 새로 생성되어야 함.
4. open()의 에러처리를 위해 fprintf()를 1번만 사용할 것
5. q1_Makefile의 파일 내용 입력에는 sprintf()와 write()만 사용할 것. 단, q1_Makefile의 파일 내용 입력에 대한 횟수는 제한 없음. 이 과정에서 memset() 사용 가능
6. q1_Makefile 파일명은 argv 인자를 이용하여 sprintf()를 1번만 사용해서 만들 것
7. argv 인자로 받게 되는 파일명이 변경되어도 정상적으로 작동되어야 할 것

38.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

#define MAXLEN 256

int main(int argc, char *argv[])
{
        char * fname, * makefile;
        // fname : argv 인자를 받는 문자열, makefile : Makefile 파일명
        char * tempStr;
        // Makefile 파일 입력 시 임시로 사용되는 문자열
        int fd;
        // Makefile 파일 디스크립터

        makefile = malloc(sizeof(char) * MAXLEN);
        tempStr = malloc(sizeof(char) * MAXLEN);

        memset(makefile, 0, sizeof(makefile));
        memset(tempStr, 0, sizeof(tempStr));

        fname = strtok(argv[1], ".");

        sprintf(makefile, "%s_Makefile", fname);

        if((fd = open(makefile, O_RDWR|O_CREAT|O_TRUNC, 0644)) == 0){
                fprintf(stderr, "open error for %s\n", makefile);
                exit(1);
        }

        sprintf(tempStr, "%s : %s.o\n", fname, fname);
        write(fd, tempStr, strlen(tempStr));

        memset(tempStr, 0, sizeof(tempStr));
        sprintf(tempStr, "\tgcc -o %s %s.o\n", fname, fname);
        write(fd, tempStr, strlen(tempStr));

        memset(tempStr, 0, sizeof(tempStr));
        sprintf(tempStr, "%s.o : %s.c\n", fname, fname);
        write(fd, tempStr, strlen(tempStr));

        memset(tempStr, 0, sizeof(tempStr));
        sprintf(tempStr, "\tgcc -c -o %s.o %s.c\n", fname, fname);
        write(fd, tempStr, strlen(tempStr));

        memset(tempStr, 0, sizeof(tempStr));
        sprintf(tempStr, "clean : \n");
        write(fd, tempStr, strlen(tempStr));

        memset(tempStr, 0, sizeof(tempStr));
        sprintf(tempStr, "\trm %s\n", fname);
```

```
        write(fd, tempStr, strlen(tempStr));

        close(fd);
        printf("%s make success\n", makefile);
        return 0;
}
```

실행 결과

```
root@localhost:/home/oslab# ./a.out q1.c
q1_Makefile make success
root@localhost:/home/oslab# make .f q1_Makefile
gcc .c .o q1.o q1.c
gcc .o q1 q1.o
root@localhost:/home/oslab# ./q1
Enter the number : 10
Sum of Even number : 30
Sum of Odd number : 25
```

39. 다음은 2번 설계과제의 일부분을 수정한 것이다. argv 인자로 주어진 파일의 크기를 구한 후 라인 넘버를 구하는 프로그램을 작성하시오.

< 조 건 >

1. 주어진 변수를 그대로 사용할 것. 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. argv 인자로 주어지는 파일을 열기 위해 읽기 모드로 fopen()을 1번만 사용할 것.
4. fopen()의 에러처리를 위해 fprintf()를 1번만 사용할 것
5. 파일의 크기를 구하기 위해 fseek() 1번, ftell()를 1번만 사용할 것
6. 파일의 라인 넘버를 구하기 위해 fseek() 2번, fread() 1번, fgets()를 1번 사용할 것
7. argv 인자로 받게 되는 파일명이 변경되어도 정상적으로 작동되어야 할 것

39.c

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

#define BUFSIZE 256

int main(int argc, char * argv[])
{
        FILE * fp;
        int length = 0, cnt = 0; // length : 파일 크기, cnt : 라인 넘버
        char strJava[BUFSIZE];

        if((fp = fopen(argv[1], "r")) == NULL){
                fprintf(stderr, "fopen error\n");
                exit(1);
        }

        fseek(fp, 0, SEEK_END);
        length = ftell(fp);

        printf("%s file size if %d bytes\n", argv[1], length);

        fseek(fp, 0, SEEK_SET);
```

```
        while(!feof(fp)){

                if(fread(strJava, 1, 1, fp) == 0)
                        break;
                else
                        fseek(fp, -1, SEEK_CUR);

                fgets(strJava, sizeof(strJava), fp);
                cnt++;
        }

        fclose(fp);
        printf("%s line number is %d lines\n", argv[1], cnt);
        return 0;
}
```

실행 결과

```
root@localhost:/home/oslab# ./a.out q1.c
q1.c file size if 349 bytes
q1.c line number is 25 lines
```

40. 다음은 3번 설계과제의 일부분을 수정한 것이다. argv 인자에 백업할 파일을 넣고 정해진 시간이 지난파일들을 삭제하는 프로그램을 만드시오.

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
2. getTimeBackupList()함수를 제외한 주어진 변수를 그대로 사용할 것, 변수의 추가나 삭제 시 감점
3. 백업은 3초 마다 수행하며, 백업한 파일 중에 현재 시간보다 10초 이상 차이가 난다면 해당파일 삭제
4. 기존의 백업된 파일이 있어도 프로그램 수행 시, 현재 시간보다 10초 이상 차이가 난다면 해당파일 삭제
5. 파일들을 삭제하기 위한 unlink() 1번 사용할 것
6. 백업해야 할 파일과 백업한 파일의 내용은 동일해야 함
7. argv 인자로 받게 되는 파일명이 변경되어도 정상적으로 작동되어야 할 것
8. 일반 파일만 인자로 받을 것

40.c

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>
#include <dirent.h>
#include <time.h>
#include <sys/stat.h>
#include<fcntl.h>

void file_backup(char *);
char **getTimeBackupList(char *, int *,int);
char *strTodate(char *);

void main(int argc, char **argv)
{
        if(argc <2)
        {
                perror("fewer parameters\n");
                exit(1);
```

```
        }
        else if(2<argc)
        {
                perror("usage :backup\n");
                exit(1);
        }
        if(access(argv[1],F_OK)<0)
        {
                perror("<FILENAME> not Exist\n");
                exit(1);
        }
        while(1)
        {
                file_backup(argv[1]);
                sleep(3);
                system("ls ");
        }
}

char *strTodate(char *old_str){
        char date[14] = {0};
        time_t timer;
        struct tm *t;

        timer = time(NULL);
        t = localtime(&timer);

        char *result = (char *)calloc(sizeof(char), PATH_MAX);

        strcpy(result,old_str);
        sprintf(date, "_%02d%02d%02d%02d%02d%02d",t->tm_year %100, t->tm_mon + 1, t->tm_mday, t->m_hour, t->t
m_min, t->tm_sec);
        strcat(result, date);
        return result;
}

char **getTimeBackupList(char *name, int *ret,int timer){
        char *real = strTodate(name);
        int length = strlen(real);
        struct dirent **namelist;
        char buf[255];
        int cnt = scandir(".", &namelist, NULL, alphasort);
        char **list = (char **)malloc(cnt * sizeof(char *));
        int many = 0;
        char ptr[255];
        char *pptr;
        int len=0;
        struct tm strparsing={0},strnowparsing={0}, *t;
        time_t now;
        long long int nowtime,filetime;
        now =time(NULL);
        t=localtime(&now);
        char nowtimestamp[255];
        sprintf(nowtimestamp, "%02d%02d%02d%02d%02d%02d",t->tm_year %100, t->tm_mon + 1, t->tm_mday, t->tm_h
```

```
our, t->tm_min, t->tm_sec);

        for (int i = 0; i <cnt; i++)
        {
                if ((!strcmp(namelist[i]->d_name, ".") || !strcmp(namelist[i]->d_name, ".."))
                        continue;
                if(strlen(namelist[i]->d_name)!= length)
                        continue;

                int namelen =strlen(namelist[i]->d_name);
                for(int j = 0 ; j< namelen;j++)
                {
                        if(namelist[i]->d_name[j] =='_')
                        {
                                if (!strncmp(namelist[i]->d_name,name,j))
                                {
                                        int timercnt=0;
                                        char timestamp[255];
                                        char datelist[6][3];
                                        char nowlist[6][3];

                                        strcpy(timestamp,(namelist[i]->d_name)+j+1);
                                        timercnt=strlen(timestamp);

                                        int iter =0,datelevel=0;
                                        int nowyear=0, fileyear=0;

                                        while(datelevel<6)
                                        {
                                                datelist[datelevel][0]=timestamp[iter];nowlist[datelevel][0]=nowtim
estamp[iter++];
                                                datelist[datelevel][1]=timestamp[iter];nowlist[datelevel][1]=nowtim
estamp[iter++];
                                                datelist[datelevel][2]='\0';nowlist[datelevel][2]='\0';
                                                datelevel++;
                                        }

                                        nowyear =atoi(nowlist[0]);
                                        fileyear=atoi(datelist[0]);
                                        strparsing.tm_mon=atoi(datelist[1]);strparsing.tm_mday=atoi(datelist[2]);
                                        strparsing.tm_hour=atoi(datelist[3]);strparsing.tm_min=atoi(datelist[4]);
                                        strparsing.tm_sec=atoi(datelist[5]);strnowparsing.tm_mon=atoi(nowlist[1]);
                                        strnowparsing.tm_mday=atoi(nowlist[2]);strnowparsing.tm_hour=atoi(nowlist
[3]);
                                        strnowparsing.tm_min=atoi(nowlist[4]);     strnowparsing.tm_sec=atoi(nowlist
[5]);

                                        nowtime=0; filetime=0;
                                        nowtime +=(2592000*12) * nowyear; nowtime +=2592000 *strnowparsing.t
m_mon;
                                        nowtime +=86400 *strnowparsing.tm_mday;nowtime +=3600 *strnowparsin
g.tm_hour;
                                        nowtime +=60 *strnowparsing.tm_min;nowtime +=strnowparsing.tm_sec;
                                        filetime +=(2592000 *12) *fileyear;filetime +=2592000 *strparsing.tm_mon;
                                        filetime +=86400 *strparsing.tm_mday; filetime +=3600 *strparsing.tm_ho
```

```
ur;
                                                filetime +=60 *strparsing.tm_min;filetime +=strparsing.tm_sec+timer;

                                                if(nowtime>filetime)
                                                {
                                                        list[many] = (char *)malloc(sizeof(char) * strlen(namelist[i]->d_
name));
                                                        strcpy(list[many],(namelist[i]->d_name));
                                                        many++;
                                                        break;
                                                }
                                        }
                                }
                        }
                }

                for (int i = 0; i <cnt; i++)
                        free(namelist[i]);

                free(namelist);
                *ret = many;
                return list;
}

void file_backup(char *filename)
{
                char *data;
                char *backuptime;
                struct stat src_st;
                int tcnt=0,fd =0,nfd=0,len=0;
                int i=0;
                char **gettime =getTimeBackupList(filename,&tcnt,10);

                for(int i = 0 ; i< tcnt ; i++)
                {
                        char buf[255];
                        unlink(gettime[i]);
                }

                if((fd=open(filename,O_RDONLY))<0)
                {
                        fprintf(stderr,"%s is not exit \n",filename);
                        return;
                }

                stat(filename,&src_st);
                backuptime=strTodate(filename);
                nfd =open(backuptime,O_CREAT | O_TRUNC | O_WRONLY ,666);

                if(nfd <0)
                {
                        printf("strtohex error %s \n",backuptime);
                        return;
                }
```

```
        data =(char*)malloc(src_st.st_size);

        while((len=read(fd,data,src_st.st_size))>0)
        {
                write(nfd,data,len);
        }

        free(data);
        close(fd);
}
```

실행결과

```
root@localhost:/home/oslab# ls
a.out ssu.c ssu.c_190611114600 ssu.c_190611114603 ssu.c_190611114606 ssu.c_190611114609
root@localhost:/home/oslab# ./a.out
a.out ssu.c ssu.c_190611114645
a.out ssu.c ssu.c_190611114645 ssu.c_190611114648
a.out ssu.c ssu.c_190611114645 ssu.c_190611114648
a.out ssu.c ssu.c_190611114645 ssu.c_190611114648 ssu.c_190611114651
a.out ssu.c ssu.c_190611114645 ssu.c_190611114648 ssu.c_190611114651 ssu.c_190611114654
a.out ssu.c ssu.c_190611114645 ssu.c_190611114648 ssu.c_190611114651 ssu.c_190611114654
a.out ssu.c ssu.c_190611114648 ssu.c_190611114651 ssu.c_190611114654 ssu.c_190611114657
a.out ssu.c ssu.c_190611114651 ssu.c_190611114654 ssu.c_190611114657 ssu.c_190611114700
a.out ssu.c ssu.c_190611114651 ssu.c_190611114654 ssu.c_190611114657 ssu.c_190611114700
^C
```

41. 다음은 3번 설계과제의 remove 명령어 일부를 수정한 것이다. 조건에 맞게 프로그램을 작성하시오.

< 조 건 >

1. 주어진 변수를 그대로 사용할 것. 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. malloc()을 1번 사용하여 listHEAD 변수에 backupList 구조체 크기만큼 메모리를 할당할 것
4. 표준출력으로 백업이 종료될 파일의 이름을 출력할 것
5. 백업리스트에서 백업이 종료될 파일을 제거할 것
6. 제거된 작업의 메모리 회수를 위해 free()를 1번 사용할 것
7. 명령어가 정상적으로 수행되면 1을 리턴할 것
8. 백업이 종료될 파일이 존재하지 않으면 -1을 리턴할 것

41.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <string.h>
#include <fcntl.h>

typedef struct backupList
{
        struct backupList* next;
        char filename[255];
}ssu_backupList;
```

```
int addFun(char *filename);
int RemoveFun(char *filename);
ssu_backupList *listHEAD;

int main(int argc, char *argv[])
{
        listHEAD=(ssu_backupList*)malloc(sizeof(ssu_backupList));
        addFun( “a.txt” );
        addFun( “b.txt” );
        addFun( “c.txt” );

        RemoveFun( “b.txt” );
        RemoveFun( “c.txt” );
        RemoveFun( “a.txt” );

        exit(0);
}

int addFun(char *filename)
{
        ssu_backupList *listadd =(ssu_backupList *)malloc(sizeof(ssu_backupList));

        strcpy(listadd->filename, filename);
        listadd->next=NULL;

        ssu_backupList *seek =listHEAD;

        while(seek->next!=NULL)
                seek=seek->next;

        seek->next=listadd;
        return 0;
}

int RemoveFun(char *filename)
{
        ssu_backupList *seek = listHEAD->next;
        ssu_backupList *prev=listHEAD;
        while(seek!=NULL)
        {
                if(!strcmp(seek->filename, filename))
                {
                        printf(“del name =%s\n“, seek->filename);
                        break;
                }
                seek=seek->next;
        }
        while(prev->next!=seek)
                prev=prev->next;

        if(seek ==NULL)
                return -1;

        prev->next=seek->next;
```

```
        free(seek);
        return 1;
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
del name =b.txt
del name =c.txt
del name =a.txt
```