※ 각 문제에서 주어진 프로그램 실행 시 주어진 실행결과가 나올 수 있도록, 빈 칸을 채우시오.

<<주의 사항>>

(1) 답을 저장할 파일의 확장자는 .txt이며 각 문제 번호와 소문제 번호(문제마다 네모 박스에 기입된 번호)를 이용하여 파일을 생성해야 함

(2) 총 44개의 소문제가 있음

(3) 각 문제의 답을 모르거나 답이 틀리더라도 파일은 생성해야 함 (NULL 파일이라도 상관 없음)

1. 다음 프로그램은 dup2()를 호출하여 표준 출력 1번 파일 디스크립터를 4번으로 복사하고 4번 파일 디스크립터를 인자로 하여 write()를 호출하면 표준 출력에 쓰는 것과 같은 결과를 보여준다. 아래 실행결과를 보고 빈칸을 채우시오.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
1-2.txt

#define BUFFER_SIZE 1024
int main(void)
{
    char buf[BUFFER_SIZE];
    char *fname = "ssu_test.txt";
    int fd;
    int length;

    if (fd = open( 1-3.txt ) < 0) {
        fprintf(stderr, "open error for %s\n", fname);
        exit(1);
    }

    if ( 1-1.txt ) {
        fprintf(stderr, "dup2 call failed\n");
        exit(1);
    }

    while (1) {
        length = 1-5.txt ;
        if (length <= 0)
            break;

        1-4.txt ;
    }

    exit(0);
}
```

실행결과

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

2. 다음 프로그램은 rename()을 호출하여 파일 이름을 ssu_test1.txt에서 ssu_test2.txt로 변경하고 두 번째 open()이 실패하는 것을 보여준다. 아래 실행결과를 보고 빈칸을 채우시오.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
2-1.txt

int main(int argc, char *argv[])
{
    int fd;
    if (argc != 3) {
        fprintf(stderr, "usage: %s <oldname> <newname>\n", argv[0]);
        exit(1);
    }

    if (fd = open( 2-4.txt ) < 0) {
        fprintf(stderr, "first open error for %s\n", argv[1]);
        exit(1);
    }
    else
        close(fd);

    if ( 2-2.txt < 0) {
        fprintf(stderr, "rename error\n");
        exit(1);
    }

    if (fd = open( 2-5.txt ) < 0)
        printf("second open error for %s\n", argv[1]);
    else {
        fprintf(stderr, "it's very strange!\n");
        exit(1);
    }

    if (fd = open( 2-3.txt ) < 0) {
        fprintf(stderr, "third open error for %s\n", argv[2]);
        exit(1);
    }

    printf("Everything is good!\n");
    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# vi ssu_test1.txt
root@localhost:/home/oslab# ./a.out ssu_test1.txt ssu_test2.txt
second open eror for ssu_test1.txt
Everything is god!
root@localhost:/home/oslab# ls ssu_test2.txt
ssu_test2.txt
```

3. 다음 프로그램은 setvbuf()를 사용하여 버퍼를 조작하는 것을 보여준다. setvbuf()를 테스트하기 앞서 tty 명령어를 통해 터미널의 번호를 확인한다. tty 명령어는 현재 표준입력에 접속된 터미널 장치 파일 이름을 출력하는 명령어로 현재 장치 파일 이름을 알아낸 후, 해당 장치의 버퍼를 조작한다. 다음 프로그램을 실행하기 위해 현재 버퍼인 /dev/pts/19 (시스템마다 다를 수 있음)를 오픈하고, setvbuf()를 설정하는 ssu_setbuf()를 호출한다. ssu_setbuf()의 첫 번째 인자인 파일에 대해 버퍼를 설정하고, 해당 파일의 파일 디스크립터를 얻는다. 얻은 파일 디스크립터를 통해 그에 맞는 모드와 크기를 설정한 후

setvbuf()를 호출하여 해당 파일의 버퍼를 설정한다. “Hello, UNIX!“ 출력은 버퍼가 설정된 후 실행되기 때문에 버퍼에 넣은 후 한 번에 출력하고, “HOW ARE YOU?“ 출력은 버퍼가 NULL로 설정된 후 실행되기 때문에 fprintf()를 호출할 때마다 버퍼에 넣지 않고 바로 출력한다. 아래 실행결과를 보고 빈칸을 채우시오.

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

#define BUFFER_SIZE 1024

void ssu_setbuf(FILE *fp, char *buf);

int main(void)
{
    char buf[BUFFER_SIZE];
    char *fname = "/dev/pts/19";
    FILE *fp;

    if (fp = fopen(fname, "w") == NULL) {
        fprintf(stderr, "fopen error for %s", fname);
        exit(1);
    }

    ssu_setbuf(fp, buf);
    fprintf(fp, "Helo, ");
    sleep(1);
    fprintf(fp, "UNIX!");
    sleep(1);
    fprintf(fp, "\n");
    sleep(1);
    ssu_setbuf(fp, NULL);
    fprintf(fp, "HOW");
    sleep(1);
    fprintf(fp, " ARE");
    sleep(1);
    fprintf(fp, " YOU?");
    sleep(1);
    fprintf(fp, "\n");
    sleep(1);
    exit(0);
}
void ssu_setbuf(FILE *fp, char *buf) {
    size_t size;
    int fd;
    int mode;

    fd = fileno(fp);
    if (isatty(fd)
        3-2.txt                                    ;
    else
        mode = _IOFBF;

    if (buf == NULL) {
```

```
        3-5.txt                                 ;
        3-1.txt                                 ;
    }
    else
        3-3.txt                                 ;

    3-4.txt                                     ;
}
```

실행결과

```
root@localhost:/home/oslab# tty
/dev/pts/19
root@localhost:/home/oslab# ./a.out
Hello, UNIX!
HOW ARE YOU?
```

4. 다음 프로그램은 ssu_hole.txt 파일을 생성한 후 첫 부분에 buf1 스트링을 쓰고, 오프셋의 위치를 파일의 처음부터 15000 만큼 이동시키고 buf2 스트링을 쓰는 프로그램이다. 아래 실행결과를 보고 빈칸을 채우시오.

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

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

char buf1[] = "1234567890";
char buf2[] = "ABCDEFGHIJ";

int main(void)
{
    char *fname = "ssu_hole.txt";
    int fd;

    if ((fd = 4-2.txt                           ) < 0) {
        fprintf(stderr, "creat error for %s\n", fname);
        exit(1);
    }

    if ( 4-3.txt                    != 12) {
        fprintf(stderr, "buf1 write error\n");
        exit(1);
    }

    if ( 4-1.txt                            < 0) {
        fprintf(stderr, "lseek error\n");
        exit(1);
    }

    if ( 4-4.txt                    != 12) {
        fprintf(stderr, "buf2 write error\n");
        exit(1);
    }
```

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

실행결과

```
root@localhost:/home/oslab# ./a.out
root@localhost:/home/oslab# ls -l ssu_hole.txt
-rw-r—r— 1 root root 15012 Jan 3 03:53 ssu_hole.txt
root@localhost:/home/oslab# od -c ssu_hole.txt
0000000   1   2   3   4   5   6   7   8   9   0  \0  \0  \0  \0  \0  \0
0000020  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
*
0035220  \0  \0  \0  \0  \0  \0  \0  \0  \0   A   B   C   D   E   F   G   H
0035240   I   J  \0  \0
0035244
```

5. 다음은 주어진 파일을 umask()를 통해 마스크 값을 지정하고, 생성된 파일의 접근 권한을 확인한다. umask()를 호출하여 파일 생성 시 파일의 마스크 값을 0으로 만들고 생성한 ssu_file1의 파일 접근 권한을 RW_MODE로 변경한다. umask()로 그룹 사용자와 다른 사용자의 읽기, 쓰기 권한에 대해 마스크를 설정하여 ssu_file2 파일은 파일의 소유자만 읽기, 쓰기가 가능하게 한다. 아래 실행결과를 보고 빈칸을 채우시오.

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

#define RW_MODE
(S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH)

int main(void)
{
    char *fname1 = “ssu_file1”;
    char *fname2 = “ssu_file2”;

    [ 5-3.txt          ];

    if ([ 5-1.txt                    ] < 0) {
        fprintf(stderr, “creat error for %s\n”, fname1);
        exit(1);
    }
    [ 5-2.txt                                        ];

    if ([ 5-4.txt                    ] < 0) {
        fprintf(stderr, “creat error for %s\n”, fname2);
        exit(1);
    }

    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# umask 002
root@localhost:/home/oslab# umask
0002
root@localhost:/home/oslab# ./a.out
root@localhost:/home/oslab# ls -l ssu_file1 ssu_file2
-rw-rw-rw-   1 root root 0 Jan  8 22:09 ssu_file1
```

```
-rw-------- 1 root root 0 Jan  8 22:09 ssu_file2
root@localhost:/home/oslab# umask
0002
```

6. 다음 프로그램은 chmod()를 호출하여 인자로 지정한 모드로 파일 접근 권한을 변경한다. 일반 사용자 권한으로 파일 접근 권한을 변경할 수 없기 때문에 에러 메시지가 출력된다. 접근 권한이 변경된 파일은 쉘에서 " ls -l " 명령어를 통해 확인할 수 있다. 아래 실행결과를 보고 빈칸을 채우시오.

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

#define MODE_EXEC (S_IXUSR|S_IXGRP|S_IXOTH)

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

    if (argc < 2) {
        fprintf(stderr, "usage: %s <file1><file2> ... <fileN>\n", argv[0]);
        exit(1);
    }

    for(i = 1; i < argc; i++) {
        if ( 6-2.txt          < 0) {
            fprintf(stderr, "%s : stat error\n", argv[i]);
            continue;
        }

        statbuf.st_mode  6-1.txt
        statbuf.st_mode  6-4.txt
        if( 6-3.txt                        < 0)
            fprintf(stderr, "%s : chmod error\n", argv[i]);
        else
            printf("%s : file permission was changed.\n", argv[i]);
    }
    exit(0);
}
```

실행결과 [일반 사용자 권한으로 실행]

```
root@localhost:/home/oslab# mkdir ssu_test_dir
root@localhost:/home/oslab# ./a.out /bin/su /home/oslab/ssu_test_dir
/bin/su : chmod error
/home/oslab/ssu_test_dir : file permission was changed.
```

7. 다음 프로그램은 첫 번째 인자로 입력받은 파일의 심볼릭 링크 파일을 두 번째 인자로 입력받은 파일이름으로 생성하는 프로그램이다. 아래 실행결과를 보고 빈칸을 채우시오.

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

```
int main(int argc, char *argv[])
{
    if(argc != 3){
        fprintf(stderr, "usage: %s <actualname> <symname>\n", argv[0]);
        exit(1);
    }

    if( 7-1.txt                    < 0){
        fprintf(stderr, "symlink error\n");
        exit(1);
    }
    else
        printf("symlink: %s -> %s\n", argv[2], argv[1]);

    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out /home/oslab/oslab /bin/oslab
symlink: /bin/oslab -> /home/oslab/oslab
root@localhost:/home/oslab# oslab
This is oslab file
```

8. 다음 프로그램은 readdir()을 호출하여 인자로 지정한 디렉토리를 읽어 이름과 파일의 크기를 출력한다. 파일 접근 권한을 변경한다. 파일이름이 있다면 stat() 호출을 통해 파일 정보를 리턴받고 stat 구조체의 파일 정보를 가지고 일반 파일인지 확인 후 이름과 크기를 출력한다. 아래 실행결과를 보고 빈칸을 채우시오.

```
#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

int main(int argc, char *argv[])
{
    struct dirent *dentry;
    struct stat statbuf;
    char filename[DIRECTORY_SIZE+1];
    DIR *dirp;

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

    if( 8-5.txt                    == NULL || 8-2.txt                    == -1){
        fprintf(stderr, "opendir, chdir error for %s\n", argv[1]);
        exit(1);
    }
```

```
    while( 8-4.txt                          != NULL){
        if( 8-3.txt          == 0)
            continue;

        memcpy(filename, 8-1.txt           , DIRECTORY_SIZE);

        if(stat(filename, &statbuf) == -1){
            fprintf(stderr, "stat error for %s\n", filename);
            break;
        }

        if((statbuf.st_mode & S_IFMT) == S_IFREG)
            printf("%-14s %ld\n", filename, statbuf.st_size);
        else
            printf("%-14s\n", filename);
    }
    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# mkdir ssu_oslab
root@localhost:/home/oslab# cd ssu_oslab
root@localhost:/home/oslab/ssu_oslab# vi ssu_oslab_1.c
root@localhost:/home/oslab/ssu_oslab# gcc -o ssu_oslab_exec ssu_oslab_1.c
root@localhost:/home/oslab/ssu_oslab# mkdir ssu_dir
root@localhost:/home/oslab/ssu_oslab# cd ..
root@localhost:/home/oslab# ./a.out ssu_oslab
..
.
ssu_dir
ssu_oslab.c     102
ssu_oslab_exec 8656
```

9. 다음 프로그램은 getcwd()를 호출하여 chdir()로 변경한 현재 작업 디렉토리를 출력한다. 작업 디렉토리를 변경 후 getcwd()를 통해 작업디렉토리가 변경된 것을 출력한다. 아래 실행결과를 보고 빈칸을 채우시오.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define PATH_MAX 1024

int main(void)
{
    char *pathname;

    if(chdir("/home/oslab") < 0){
        fprintf(stderr, "chdir error\n");
        exit(1);
    }

    pathname = malloc(PATH_MAX);

    if( 9-1.txt                          == NULL){
        fprintf(stderr, "getcwd error\n");
        exit(1);
```

```
    }

    printf("current directory = %s\n", pathname);
    exit(0);
}
```

실행결과

```
root@localhost:~/ssu_osidr$ ./a.out
current directory = /home/oslab
```

10. 다음 프로그램은 내용 첫 줄은 버퍼를 설정하여 출력할 내용을 한꺼번에 출력하고, 두 번째 줄은 설정되어 있던 버퍼를 해제하여 출력할 내용을 즉시 출력하도록 하는 프로그램이다. 아래 실행결과를 보고 빈칸을 채우시오.

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

#define BUFFER_SIZE 1024

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

    [ 10-1.txt ];
    printf("Hello, ");
    sleep(1);
    printf("OSLAB!!");
    sleep(1);
    printf("\n");
    sleep(1);

    [ 10-2.txt ];
    printf("How");
    sleep(1);
    printf(" are");
    sleep(1);
    printf(" you?");
    sleep(1);
    printf("\n");
    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
Hello, OSLAB!!
How are you?
```

※ 주어진 조건에 맞게 프로그램을 완성하시오.

<<주의 사항>>

(1) 각 문제의 답을 저장할 파일의 확장자는 문제번호 .c 임

(2) 각 문제의 답을 모르거나 답이 틀리더라도 NULL 파일을 생성해야 함

(3) 주어진 기능을 모두 구현하지 못하고 특정 부분만 수행되는 프로그램을 작성했더라도 컴파일 시에 에러가 발생할 경우 문제 파일 이름만 생성하고 NULL 파일(문제번호.c)로 만들어야 함. 즉, 일부 기능만 구현했을 경우라도 반드시 컴파일이 에러 없이 진행되어야 하며 에러가 발생할 경우 해당 문제는 0점 처리

(4) 컴파일 시 warning이 한 개 발생할 때마다 해당 문제의 점수에서 10% 감점 처리

(5) 프로그램 구현 문제에서 주어진 조건을 변경하거나, 변수를 추가/변경할 경우 해당 문제는 1개 추가/변경 시 10% 감점
(6) 주어진 출력 결과와 하나라도 다를 경우 해당 문제는 0점 처리

11. 실행 시 인자로 주어진 파일의 타입이 나오는 프로그램을 작성하시오. 단, 아래 조건과 실행 결과를 바탕으로 프로그램을 작성 할 것. (5점), 조건을 만족하지 않는 경우 0점

< 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

11.c

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

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

}

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

}
```

실행결과

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

12. 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. (5점), 조건을 만족하지 않는 경우 1개당 -2점

< 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
```

12.c

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

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
```

13. 구조체 내용을 파일에 저장한 후, 해당 파일을 읽어 출력을 하는 프로그램을 작성하시오. 단, 아래 조건과 실행 결과를 바탕으로 프로그램을 작성 할 것. (5점), 조건을 만족하지 않는 경우 1개당 -2점

< 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

13.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;

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

    while (!feof(fp)) {
        printf("%s %d %.2lf\n", tmp.name, tmp.age, tmp.height);
    }

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

    while (!feof(fp)) {
        printf("%s %d %.2lf\n", tmp.name, tmp.age, tmp.height);
    }

    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#
```

14. read()와 write()를 사용하여 파일을 복사하는 프로그램을 작성하시오, 단, 아래 조건과 실행 결과를 바탕으로 프로그램을 작성 할 것. (6점), 조건을 만족하지 않는 경우 1개당 -2점

< 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

14.c

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

```
#include <unistd.h>

#define BUFFER_SIZE 128

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

}
```

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
14  14.c  ssu_file1  ssu_file2
root@localhost:/home/oslab# cat ssu_file2
Linux System Programming!
root@localhost:/home/oslab#
```

15. 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. (8점 기준, 보너스 점수 : 완전히 구현한 학생 수(A)와 완전히 구현하지 못한 학생 수(B, 일부 기능 구현, warning 모두 포함)를 비교하여 (1) A < B/10 이면 완전히 구현한 학생들에게 6점 추가 부여, (2) A < B/9 이면 완전히 구현한 학생들에게 4점 추가 부여, (3) A < B/8 이면 완전히 구현한 학생들에게 2점 추가 부여, 조건을 만족하지 않는 경우 1개당 -2점

< 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!
```

15.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);
}
```

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!
```

16. system()으로 grep를 사용하여 타겟 파일에서 키워드를 검색하는 프로그램을 작성하시오. 단, 아래 조건과 실행 결과를 바탕으로 프로그램을 작성 할 것. (8점 기준, 보너스 점수 : 완전히 구현한 학생 수(A)와 완전히 구현하지 못한 학생 수(B, 일부 기능 구현, warning 모두 포함)를 비교하여 (1) A < B/10 이면 완전히 구현한 학생들에게 6점 추가 부여, (2) A < B/9 이면 완전히 구현한 학생들에게 4점 추가 부여, (3) A < B/8 이면 완전히 구현한 학생들에게 2점 추가 부여, 조건을 만족하지 않는 경우 1개당 -2점

< 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.

16.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;
}

void ssu_make_grep(int argc, char *argv[]) {
    int i;
}

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

17. link()와 unlink()를 사용하여 파일을 이동하거나 이름을 변경하는 프로그램을 작성하시오. 단, 아래 조건과 실행 결과를 바탕으로 프로그램을 작성 할 것. (8점 기준, 보너스 점수 : 완전히 구현한 학생 수(A)와 완전히 구현하지 못한 학생 수(B, 일부 기능 구현, warning 모두 포함)를 비교하여 (1) A < B/10 이면 완전히 구현한 학생들에게 6점 추가 부여, (2) A < B/9 이면 완전히 구현한 학생들에게 4점 추가 부여, (3) A < B/8 이면 완전히 구현한 학생들에게 2점 추가 부여, 조건을 만족하지 않는 경우 1개당 -2점

< 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

17.c

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

#define SEC_TO_MICRO 1000000

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

    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ls
1.txt 17.c a.out subdir
root@localhost:/home/oslab# ./a.out 1.txt 2.txt
Runtime : 0:15(sec:microsec)
root@localhost:/home/oslab# ls
17.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
17.c a.out subdir
root@localhost:/home/oslab# cd subdir
root@localhost:/home/oslab/subdir# ls
2.txt
```

18. 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. (9점 기준, 보너스 점수 : 완전히 구현한 학생 수(A)와 완전히 구현하지 못한 학생 수(B, 일부 기능 구현, warning 모두 포함)를 비교하여 (1) A < B/10 이면 완전히 구현한 학생들에게 7점 추가 부여, (2) A < B/9 이면 완전히 구현한 학생들에게 5점 추가 부여, (3) A < B/8 이면 완전히 구현한 학생들에게 3점 추가 부여, \0 처리를 안 해서 쓰레기값 출력 시 -2점

< 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
```

18.c

```
#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;

        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
```

19. 다음 주어진 ssu_test.txt 파일을 한 줄씩 읽어 줄의 번호와 함께 읽은 줄을 출력하는 프로그램을 작성하시오. (8점)

< 조 건 >

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

19.c

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


}
```

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 : ./ssu_cat 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
```

20. 다음 주어진 ssu_blank.txt 파일을 읽어, 공백으로 단어를 구분하여, 각 단어를 줄 단위로 ssu_line.txt에 저장하고 단어의 개수를 출력하는 프로그램을 작성하시오 (8점)

< 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
```

20.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);
    }

}
```

실행결과

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

21. 다음 프로그램 실행 시 프로그램의 인자로 주어진 파일의 타입이 나오는 프로그램을 작성하시오. (8점)

< 조 건 >

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

21.c

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

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

}

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

    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
```

22. 파일을 이동하거나 이름을 변경하는 프로그램을 작성하시오. <참고 7> (8점)

< 조 건 >

1. 주어진 변수를 그대로 사용할 것, 변수의 추가나 삭제 시 감점
1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
1. link(), unlink()를 각 1번씩 사용할 것
1. link(), unlink()의 에러 처리를 위해 fprintf()를 2번만 사용할 것
1. 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
1. 변경 전, 후 파일의 inode 번호를 알아내기 위해 stat 구조체의 멤버 변수를 사용할 것

22.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);
    }

    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
```

23. 다음 주어진 ssu_answer.txt에 있는 답 파일을 채점하여 그 결과를 ssu_res.txt에 저장하는 프로그램을 작성하시오. (9점)

< 조 건 >

1. 주어진 변수를 그대로 사용할 것, 변수의 추가나 삭제 시 감점
1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
1. fopen(), fgets(), fclose()를 각 2번씩 사용할 것
1. fopen()의 에러 처리를 위해 fprintf()를 2번만 사용할 것
1. 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
```

23.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];



    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);

    }

    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
Student name : Jung DJ , score : 70 , res : OOOXXOOOOX
Student name : Lee SS , score : 80 , res : OOOOOOXXOO
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
```

24. 프로그램 상에서 주어진 Person 구조체 내용을 파일에 저장한 후, 해당 파일을 읽어 두 번 출력을 하는 프로그램을 작성하시오. (9점)

< 조 건 >

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

24.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;



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

    while (!feof(fp)) {

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

    }

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

    while (!feof(fp)) {

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

    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#
```

25. system() 함수를 사용하여 diff 명령어를 실행하는 프로그램을 작성하시오. <참고 2, 3, 6, 8> (10점)
(보너스 점수 : 완전히 구현한 학생 수(A)와 완전히 구현하지 못한 학생 수(B, 일부 기능 구현, warning 모두 포함)를 비교하여
(1) A < B/10 이면 완전히 구현한 학생들에게 6점 추가 부여,
(2) A < B/9 이면 완전히 구현한 학생들에게 4점 추가 부여,
(3) A < B/8 이면 완전히 구현한 학생들에게 2점 추가 부여)

< 조 건 >

1. 주어진 변수를 그대로 사용할 것, 변수의 추가나 삭제 시 감점
1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
1. 프로그램 실행 시 프로그램의 인자는 diff를 위한 파일 2개를 받을 것
1. 프로그램 실행 시 프로그램의 인자로 받은 두 파일의 정보를 얻기 위해 lstat()를 사용할 것
1. lstat()의 에러 처리를 위해 fprintf()를 1번 사용할 것
1. 프로그램 실행 시 프로그램의 인자로 받은 두 파일 중 일반 파일이 아닌 파일이 있으면 fprintf()를 사용하여 에러 처리할 것
1. 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
```

25.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);
    }

    exit(1);
}
```

실행결과

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

26. getc(), putc(), getchar(), putchar(), fgets(), fputs(), scanf(), printf()를 사용하여 입출력하는 프로그램을 작성하시오. (8점)

< 조 건 >

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

26.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() 사용

        //getc(), putc() 사용

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

        //fgets(), fputs() 사용
```

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

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

        //scanf(), printf() 사용

        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!
```

27. 다음 프로그램은 하위 디렉터리를 재귀적으로 생성하는 프로그램 코드이다. 아래 조건과 실행 결과를 바탕으로 다음 프로그램을 작성하시오. (11점 기준, 보너스 점수 : 완전히 구현한 학생 수(A)와 완전히 구현하지 못한 학생 수(B, 일부 기능 구현, warning 모두 포함)를 비교하여 (1) A < B/10 이면 완전히 구현한 학생들에게 15점 추가 부여, (2) A < B/9 이면 완전히 구현한 학생들에게 13점 추가 부여, (3) A < B/8 이면 완전히 구현한 학생들에게 11점 추가 부여, (4) A < B/7 이면 완전히 구현한 학생들에게 9점 추가 부여, (5) A < B/6 이면 완전히 구현한 학생들에게 7점 추가 부여, (6) A < B/5 이면 완전히 구현한 학생들에게 5점 추가 부여, (7) A < B/4 이면 완전히 구현한 학생들에게 3점 추가 부여

< 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

27.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 '?' :
                break;
        }
    }
    if( argc < 3)
    {
    }
    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 ()
    {
        fprintf(stderr, "mkdir error\n");
        exit(1);
    }

    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++;
    if ( stat(cur_dir, &dir_st) < 0){
    }
    strcat(tmp_filename, cur_dir);
    if(o_flag)
    {
    }
    else if (e_flag)
    {
    }
    else if (!o_flag && !e_flag)
    {
    }
    if ( depth == 0)
        return 0;
    return create_dir(depth-1, cur_dir);
}

void writefile(char *in_f, char *out_f)
{
}

void change_mod(const char *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
```

<참고> 코드 작성에 필요한 함수

1. memset : s가 가르키는 메모리 영역의 처음 n바이트를 상수 바이트 c로 채우는 함수

```
#include <string.h>
void *memset(void *s, int c, size_t n);
                                                    리턴값: void *s에 대한 포인터, 실패시 NULL
```

2. diff 프로그램의 사용법

```
SYNOPSIS          top
       diff [OPTION]... FILES
DESCRIPTION          top
       Compare FILES line by line.

       Mandatory arguments to long options are mandatory for short options
       too.

       --normal
              output a normal diff (the default)

       -q, --brief
              report only when files differ

       -s, --report-identical-files
              report when two files are the same

       -c, -C NUM, --context[=NUM]
              output NUM (default 3) lines of copied context

       -u, -U NUM, --unified[=NUM]
              output NUM (default 3) lines of unified context


      ...
       --left-column
              output only the left column of common lines

       --suppress-common-lines
              do not output common lines

       -p, --show-c-function
              show which C function each change is in

       -F, --show-function-line=RE
              show the most recent line matching RE

       --label LABEL
              use LABEL instead of file name and timestamp (can be repeated)

       ...

       --suppress-blank-empty
```

```
        suppress space or tab before empty output lines

-l, --paginate
        pass output through 'pr' to paginate it

-r, --recursive
        recursively compare any subdirectories found

--no-dereference
        don't follow symbolic links

-N, --new-file
        treat absent files as empty

--unidirectional-new-file
        treat absent first files as empty

--ignore-file-name-case
        ignore case when comparing file names

--no-ignore-file-name-case
        consider case when comparing file names

...
```