Soongsil University

School of Computing

Midterm Exam

Course Title	리눅스 시스템 프로그래밍 설계 및 실습
Instructor	**Jiman Hong**
Date of Exam	**April 29, 2018**
Duration of Exam	**4 hours**
Number of Exam Page (including this cover page)	**23**
Number of Questions	**15**
Exam Type	**Closed Book**
Additional Materials Allowed	**None**

Student ID (학번)	
Name (이름)	
File Size (학생이 직접 입력)	**(bytes)**
Hash Value	

Show Your Work and Reasoning Clearly!
Write legibly!
Write only to the space provide for each question!
Good Luck!

Grade	

주의 사항

0. 가상화 프로그램인 VMWare에 리눅스 운영체제가 실행된 상태에서 id : oslab , passwd : oslab 으로 로그인
(1)^-Alt (컨트롤-알트) 키 등을 눌러 리눅스 콘솔 외 윈도우에 접근할 경우 부정행위로 간주
(2) 하나의 콘솔 터미널만 실행 가능하기 때문에 기 실행된 터미널 외 추가 터미널 생성을 시도할 경우 부정행위로 간주

1. 현재 작업 디렉토리(/home/oslab)에 문제 type과 자기 "학번"을 파일 이름으로 하는 디렉토리를 생성(mkdir 명령어) 할 것
<예. 주어진 문제지의 타입이 A형이고 학생 학번이 20122336 일 경우 %mkdir A20122336 반드시 문제 type은 대문자로 써야 함>

2. 위에서 만든 "문제타입학번"을 파일 이름으로 하는 디렉토리 아래 문제별 파일을 생성하여 답을 생성된 파일에 입력할 것
(1) 1번~4번 문제의 경우 답을 저장할 파일의 확장자 명은 .txt이며 각 문제 번호와 소문제 번호(문제마다 네모 박스에 기입된 번호)를 이용하여 파일을 생성해야 함
<예. 4번 문제의 두 번째 소문제(네모 박스 안 번호가 4-2.txt인 경우) 4-2.txt 파일에 답을 입력할 것>
(2) 5번~15번 문제의 경우 답을 저장할 파일의 확장자는 .c 이며 각 문제 번호를 이용하여 파일을 생성해야 함
<예. 7번 문제의 답은 7.c 파일을 생성하여 답안을 작성해야 함>
(3) 각 문제의 답을 모르거나 답이 틀리더라도 NULL 파일을 생성해야 함
<예. 8번 문제를 못 풀 경우 8.c 파일은 만들어야 하나, 파일에는 내용이 없어야 함.>
(4) 모든 문제에 대해 답을 파일에 작성할 경우 18개의 .txt 파일(1번~4번)과 11개 .c 파일(5번~15번)이 /home/oslab/문제타입학번 디렉토리에 존재해야 함
(5) 네모 박스 안 번호가 순차적으로 기입되어 있지 않으니 반드시 번호를 확인할 것

3. 5번~15번 프로그램 문제의 경우
(1) 주어진 기능을 모두 구현하지 못하고 일부 기능만 구현했을 경우라도 반드시 컴파일이 에러 없이 진행되어야 하며 에러가 발생할 경우 해당 문제는 0점 처리
(2) 컴파일 시 warning이 한 개 발생할 때마다 해당 문제의 점수에서 10% 감점
(3) 프로그램 구현 문제에서 주어진 조건을 변경하거나, 변수를 추가/변경할 경우 해당 문제는 0점 처리
(4) 주어진 출력 결과와 하나라도 다를 경우 해당 문제는 0점 처리

4. 시험을 완료하면
(1) /home/oslab/ 디렉토리로 이동하여 문제타입학번을 이용하여 자신이 생성한 디렉토리의 크기(du -b 명령어)를 확인하고 시험지 첫장에 File Size라고 써 있는 칸에 디렉토리의 크기(byte)를 쓸 것
<예. 주어진 문제 타입이 A이고 학생 학번이 20122336 일 경우 %du -b A20122336 >
(2) 감독관에게 USB(2개 복사) 복사를 요청
(3) USB에 복사된 파일의 크기와 (1)에서 확인한 파일 크기와 (2) 해쉬 값을 확인
(4) 사용한 PC에 디렉토리와 파일은 지우지 말고 그대로 둘 것

5. 위 주의 사항을 어긴 문제는 무조건 0점 처리하고 규칙을 어긴 학생의 총점 - 100* (규칙을 어긴 답 수*10%) 부여.
(1) 규칙을 어긴 답의 개수가 10개 이상이면 총점은 0점 처리 (중간시험 또는 기말시험 0점의 경우 F 학점 처리)
<예. 어떤 학생이 규칙을 어긴 답을 4개 제출하고 총점이 50점이면 50-100*4*10%=10점>

6. 다음 사항을 어길 경우 F학점 처리
(1) 감독관이 허락하기 전에 USB 등을 마운트 시킬 경우 (자신이 작성한 답을 복사하기를 원할 경우 시험지를 제출하고 감독관의 감독 하에 USB 등에 복사)
(2) 유무선 네트워크 액세스 디바이스(동글, 에그 등)를 이용할 경우
(3) 기타 부정 행위

※ 각 문제에서 주어진 프로그램 실행 시 주어진 실행결과가 나올 수 있도록, 빈 칸을 채우시오. [1-3, 각 5점, 총 15점]

<<주의 사항>>

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

(2) 1번~3번 문제는 각각 (1) ~ (5)번의 5개 소문제가 있기 때문에 1-1.txt, 1-2.txt, 1-3.txt, 1-4.txt, 1-5.txt / 2-1.txt, 2-2.txt, 2-3.txt, 2-4.txt, 2-5.txt / 3-1.txt, 3-2.txt, 3-3.txt, 3-4.txt, 3-5.txt 등 각 문제에 대해 각 5개의 파일을 생성하고 답을 작성해야 함.

(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 error for ssu_test1.txt
Everything is good!
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, "Hello, ");
    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?
```

※ 다음 물음에 답하시오. [4, 총 6점]

<<주의 사항>>

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

(2) 4번 문제는 (1) ~ (3)번의 3개 소문제가 있기 때문에 4-1.txt, 4-2.txt, 4-3.txt 등 3개의 파일을 생성하고 답을 작성해야 함.

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

4. (1) 다음 프로그램을 gcc로 컴파일 시에 특정 옵션을 주면 아래 보이는 컴파일 결과를 확인할 수 있는지 빈칸에 적절한 옵션을 쓰시오. 단, 정확한 옵션만 파일에 쓰시오. 옵션이 필요 없는 경우에는 NULL 파일을 생성하시오.

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

int main (void)
{
        double x = -3.14;
        double y = abs(x);  /* fabs(x)*/
        printf ("x = %g |x| = %g\n", x, y);
        return 0;
}
```

컴파일 결과

```
root@localhost:/home/oslab# gcc [4-2.txt                ]
w_ex9.c: In function ‘main’:
w_ex9.c:7:17: warning: conversion to ‘int’ from ‘double’ may alter its value [-Wfloat-conversion]
  double y = abs(x);  /* fabs(x)*/
                 ^
root@localhost:/home/oslab#
```

4. (2) 다음은 주어진 프로그램을 gcc로 컴파일 후 gdb로 디버깅 하는 과정을 보여준다. 빈칸에 gdb 적절한 명령어를 쓰시오. 명령이 필요 없는 경우에는 NULL 파일을 생성하시오.

```
#include <stdio.h>
#define NUM 5

int score[NUM];

int sum(int cnt){
        int i;
        int sum;

        for(i = 0; i < cnt ; i++){
                sum += score[i];
        }
        return sum;
}

int main()
{
        int i = 0;
        int cnt = 0;

        printf("input scores. input -1 to finish.\n");

        for(i = 0;i < NUM;i++)  {
                printf("score #%d : ", cnt+1);
                scanf("%d", score[cnt]);
                if(score[cnt] == -1)
                        break;
                cnt++;
        }

        printf("%d scores read.\n", cnt);
        printf("--- result ---\n");
        printf("sum : %d avg : %d\n", sum(cnt), sum(cnt)/cnt);

        return 0;
}
```

컴파일 및 gdb 실행 결과

```
root@localhost:/home/oslab# gcc -Wall -W -g bug.c
bug.c: In function ‘main’:
bug.c:25:9: warning: format ‘%d’ expects argument of type ‘int *’, but argument 2 has type ‘int’ [-Wformat=]
   scanf("%d", score[cnt]);
         ^
root@localhost:/home/oslab#
root@localhost:/home/oslab# ./a.out
input scores. input -1 to finish.
score #1 : 102
세그멘테이션 오류 (core dumped)
root@localhost:/home/oslab#
root@localhost:/home/oslab# gdb a.out -q
Reading symbols from a.out...done.
```

```
(gdb) 4-1.txt
Starting program: /home/oslab/source/gdb/a.out
input scores. input -1 to finish.
score #1 : 102

Program received signal SIGSEGV, Segmentation fault.
_IO_vfscanf_internal (s=0xb7fbb5a0 <_IO_2_1_stdin_>, format=0x804865f "%d", argptr=0xbffff604 "", errp=0x0)
    at vfscanf.c:1902
1902    vfscanf.c: 그런 파일이나 디렉터리가 없습니다.
(gdb) 4-3.txt
#0  _IO_vfscanf_internal (s=0xb7fbb5a0 <_IO_2_1_stdin_>, format=0x804865f "%d", argptr=0xbffff604 "",
    errp=0x0) at vfscanf.c:1902
#1  0xb7e6513e in __isoc99_scanf (format=0x804865f "%d") at isoc99_scanf.c:37
#2  0x08048520 in main () at bug.c:25
(gdb) q
A debugging session is active.

        Inferior 1 [process 3812] will be killed.

Quit anyway? (y or n) y
```

※ 주어진 조건에 맞게 프로그램을 완성하시오. [5-15, 총79점]

<<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 0점 처리

(6) 주어진 출력 결과와 하나라도 다를 경우 해당 문제는 0점 처리

5. 실행 시 인자로 주어진 파일의 타입이 나오는 프로그램을 작성하시오. 단, 아래 조건과 실행 결과를 바탕으로 프로그램을 작성 할 것. (5점)

< 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

5.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
```

6. 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. (5점)

< 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
```

6.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];
}
```

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 |100| OOOOOOOOOO
Hong GD |50| XOOOOXXOXX
```

7. 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)

< 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
```

7.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
```

8. 구조체 내용을 파일에 저장한 후, 해당 파일을 읽어 출력을 하는 프로그램을 작성하시오. 단, 아래 조건과 실행 결과를 바탕으로 프로그램을 작성 할 것. (5점)

< 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

8.c

```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#
```

9. read()와 write()를 사용하여 파일을 복사하는 프로그램을 작성하시오, 단, 아래 조건과 실행 결과를 바탕으로 프로그램을 작성 할 것. (6점)

< 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

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

10. 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

< 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!
```

10.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!
```

11. 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

< 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.
```

11.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.out
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
```

12. 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

< 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

12.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 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
```

13. 다음은 컴파일 과정에서 컴파일이 필요한 파일만 찾아 컴파일을 시켜주는 프로그램이다. 단, 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

< 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
```

13.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);
	}

	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;


}
```

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#
```

14. 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

< 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
```

14.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
```

15. 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

15.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
```