Exam Type : A, Number of Exam Page : 27, Number of Questions : 21

※ 다음 물음에 답하시오 [1~5]

1. 다음 프로그램은 ungetc()를 호출하여 파일로부터 입력 받은 숫자와 연산자를 분리하여 출력한다. 아래 실행결과를 보고 소스코드를 완성하시오. [2점]

<ssu_expr.txt>

123+456*789

1.c

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

int main(void)
{
    char operator;
    FILE *fp;
    int character;
    int number = 0;

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

    while (!feof(fp)) {
        while ( [ 1-4.txt ] )
            number = 10 * number + [ 1-2.txt ] ;

        fprintf(stdout, " %d\n", number);
        number = 0;
  if (character != EOF) {
            [ 1-1.txt ] ;
            [ 1-3.txt ] ;
            printf("Operator => %c\n", operator);
        }
    }

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

실행결과

```
root@localhost:/home/oslab# vi ssu_expr.txt
root@localhost:/home/oslab# ./ssu_ungetc
 123
Operator => +
 456
Operator => *
 789
Operator =>
 0
```

2. 다음 프로그램은 setvbuf()를 사용하여 버퍼를 설정한다. setvbuf()를 확인하기 앞서 tty 명령어를 통해 터미널의 번호를 확인한다. tty 명령어는 현재 표준 입력에 접속된 터미널 장치 파일 이름을 출력하는 명령어로 현재 장치 파일 이름을 알아낸 후, 해당 장치의 버퍼를 조작한다. 아래 실행결과를 보고 소스코드를 완성하시오. [2점]

2.c

```
#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))
        mode = [ 2-1.txt ] ;
    else
        mode = [ 2-2.txt ] ;
```

```
    if (buf == NULL) {
        mode = [ 2-4.txt ] ;
        size = 0;
    }
    else
        size = BUFFER_SIZE;

    [ 2-3.txt ] ;
}
```

실행결과

```
oslab@localhost:~$ tty
/dev/pts/17
oslab@localhost:~$ ./ssu_setvbuf
Hello, UNIX!!
HOW ARE YOU?
```

3. 다음 프로그램은 chmod()를 통해 파일의 접근 권한을 변경한다. 아래 실행결과를 보고 소스코드를 완성하시오. [1.5점]

3.c

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

int main(void)
{
    struct [ 3-3.txt ] statbuf;
    char *fname1 = "ssu_file1";
    char *fname2 = "ssu_file2";

    if ( [ 3-2.txt ] )
        fprintf(stderr, "stat error %s\n", fname1);

    if (chmod(fname1, (statbuf.st_mode & ~S_IXGRP) | S_ISUID) < 0)
        fprintf(stderr, "chmod error %s\n", fname1);

    if ( [ 3-1.txt ] )
        fprintf(stderr, "chmod error %s\n", fname2);

    exit(0);
}
```

실행결과

```
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# ./ssu_chmod_1
root@localhost:/home/oslab# ls -l ssu_file1 ssu_file2
-rwSrw-rw- 1 root root 0 Jan  8 22:09 ssu_file1
-rw-r--r-x 1 root root 0 Jan  8 22:09 ssu_file2
```

4. 다음 프로그램은 rename()를 호출하여 파일의 이름을 변경한다. 인자로 지정한 파일을 rename() 호출을 통하여 변경한다. 아래 실행결과를 보고 소스코드를 완성하시오. [2점]

4.c

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

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

    if ( [ 4-3.txt ] ) {
        fprintf(stderr, "usage: %s <oldname> <newname>\n", argv[0]);
        exit(1);
    }

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

    if ( [ 4-4.txt ] )       {
        fprintf(stderr, "rename error\n");
        exit(1);
    }


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

    if ( [ 4-2.txt ] )      {
        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# ./ssu_rename 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
```

5. 다음 프로그램은 setjmp()를 호출했을 때 저장되는 변수를 확인한다. 프로그램을 컴파일 할 때 최적화 옵션을 사용하지 않으면 지역 변수, volatile 변수, register 변수는 메모리에 저장된다. 그리고 메모리에 저장된 변수는 longjmp() 호출 시 값을 유지한다. 아래 실행결과를 보고 소스코드를 완성하시오. [1.5점]

5.c

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

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

[ 5-2.txt ] glob_buffer;

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

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

    if ( [ 5-1.txt ] ) {
        printf("after longjmp, loc_var = %d, loc_volatile = %d,
                loc_register = %d\n", loc_var, loc_volatile, loc_register);
        exit(0);
    }

    loc_var = 80;
    loc_volatile = 81;
    loc_register = 83;
    ssu_nested_func(loc_var, loc_volatile, loc_register);
    exit(0);
}
void ssu_nested_func(int loc_var, int loc_volatile, int loc_register) {
    printf("before longjmp, loc_var = %d, loc_volatile = %d,
            loc_register = %d\n", loc_var, loc_volatile, loc_register);
    [ 5-3.txt ] ;
}
```

실행결과

```
root@localhost:/home/oslab# gcc -o ssu_setjmp ssu_setjmp.c
root@localhost:/home/oslab# ./ssu_setjmp
before longjmp, loc_var = 80, loc_volatile = 81, loc_register = 83
after longjmp, loc_var = 80, loc_volatile = 81, loc_register = 83
root@localhost:/home/oslab# gcc -o ssu_setjmp ssu_setjmp.c -O
root@localhost:/home/oslab# ./ssu_setjmp
before longjmp, loc_var = 80, loc_volatile = 81, loc_register = 83
after longjmp, loc_var = 10, loc_volatile = 81, loc_register = 12
```

※ 다음 물음에 답하시오 [6]

6. vi 에디터에서 최소의 명령어로 origin 코드를 modified 코드로 변경할 수 있는 명령어를 쓰시오 (커서 옮기는 것 모두 포함) [5점]

before	after

01. #include <stdio.h>
02. #include <string.h>
03.
04. int invertStr(char *a)
05. {
06. int len, i;
07. char b[20] = {0x00,};
08. len = strlen(a);
09. for (i = 0; i < len; i += 2)
10. {
11. b[len-i-2] = a[i];
12. b[len-i-1] = a[i+1];
13. }
14. printf("%s\n", b);
15.
16. return 0;
17. }
18.
19. int main()
20. {
21. int i = 0;
22. int len = 0;
23. char a[][5] ={"은", "는","이","가","하는","하면","과","나는",""};
24. char b[20] = {0x00,};
25. char *c;
26.
27. c = *a;
28. while(strlen(c) > 0)
29. {
30. printf("%s\n",c);
31. invert_str(c);
32. c=c+5;
33. }
34. invert_str("가나다라마바사");
	01. #include <stdio.h>
	02. #include <string.h>
	03.
	04. int invert_str(char *a)
	05. {
	06. int len, i;
	07. char b[20] = {0x00,};
	08. len = strlen(a);
	09. for (i = 0; i < len; i += 2)
	10. {
	11. b[len-i-2] = a[i];
	12. b[len-i-1] = a[i+1];
	13. }
	14. printf("%s\n", b);
	15.
	16. return 0;
	17. }
	18.
	19. int main()
	20. {
	21. int i = 0;
	22. int len = 0;
	23. char a[][5] ={"은", "는","이","가","하는","하면","과","나는",""};
	24. char b[20] = {0x00,};
	25. char *c;
	26.
	27. c = *a;
	28. while(strlen(c) > 0)
	29. {
	30. printf("%s\n",c);
	31. invert_str(c);
	32. c=c+5;
	33. }
	34. invert_str("가나다라마바사");
	35.
	36. return 0;
	37. }

※ 다음 물음에 답하시오 [7~21]

7. 다음 프로그램은 두 개의 파일을 인자로 입력받아 첫 번째 파일의 내용을 두 번째 파일에 복사하는 프로그램이다. open()을 통해 두 개의 파일을 오픈하고 read()를 통해 첫 번째 파일에서 문자열을 읽어 write()를 통해 출력할 파일에 쓴다. 아래 실행결과를 보고 소스코드를 완성하시오. [5점]

7.c 실행결과
root@localhost:/home/oslab# ./ssu_write_2 /etc/passwd password root@localhost:/home/oslab# ls -l /etc/passwd password -rw-r—-r-- 1 root root 2240 Jan 2 23:55 /etc/passwd -rw-r—-r-— 1 root root 2240 Jan 3 05:27 password

8. 다음 프로그램은 실행 시 인자로 파일 하나를 입력받아 인자로 받은 파일에 대해 stat()을 사용하여 파일의 정보를 읽은 후 파일 정보의 수정시간을 참고하여 2초마다 수정을 확인한다. 파일이 수정되었다면 해당 파일에 대한 수정 확인 문구를 출력하고 수정 기준시간을 갱신한다. 아래 실행결과를 보고 소스코드를 완성하시오. 단, void ssu_checkfile(char *fname, time_t *time) 함수를 통해 파일의 수정 확인 [5점]

8.c 실행결과
root@localhost:/home/oslab# ./ssu_stat_2 ssu_hole.txt Warning : ssu_hole.txt was modified!. Warning : ssu_hole.txt was modified!. Warning : ssu_hole.txt was modified!. Warning : ssu_checkfile() error!

9. system() 함수를 사용하여 diff 명령어를 실행하는 프로그램을 작성하시오. [5점]

< 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. stat()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
```

9.c

```
#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
```

10. getc(), putc(), getchar(), putchar(), fgets(), fputs(), scanf(), printf()를 사용하여 입출력하는 프로그램을 작성하시오. [5점]

< 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. 파일 타입은 주어진 print_file_type()를 통해 실행결과와 같이 출력할 것
7. 실행결과와 같이 문자열을 출력할 것

10.c

```
#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!
```

11. 프로그램 상에서 주어진 Person 구조체 내용을 파일에 저장한 후, 해당 파일을 읽어 두 번 출력을 하는 프로그램을 작성하시오 [5점]

< 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

11.c

```
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#
```

12. 주어진 ssu_test.txt 파일에 포함된 문자 'm'과 'M'의 개수의 합을 구하는 프로그램을 작성하시오. 단, 아래 조건과 실행 결과를 바탕으로 프로그램을 작성 할 것. 어떠한 텍스트 파일에도 정상수행 되어야함. [5점]

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
2. open(), read()를 한번 씩 사용할 것
3. open()함수의 에러 처리를 위해 fprintf()를 사용할 것

<ssu_test.txt>

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

12.c

```
int main(void)
{
        int fd;
        char *fname = "ssu_test.txt";
        char ch;
        int count = 0;




}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
m + M = 8
```

13. 다음 프로그램은 특정 c 프로그램(문제에서는 test1.c와 test2.c)을 컴파일 하여 컴파일 시 (1)에러가 발생하는 경우에는 에러의 결과를 “예제*_err.txt”(문제에서는 test1.c를 컴파일 한 경우 test1_err.txt, test2.c를 컴파일 한 경우 test2_err.txt)에 저장(리다이렉션)하고 “예제*_err.txt”(문제에서는 test2_err.txt) 파일의 크기를 출력한다. (2)에러가 발생하지 않은 경우 즉, “예제*_err.txt”의 크기가 0으로 생성될 경우 파일 크기를 확인(출력은 하지 않음)한 후 “예제*_err.txt”(문제에서는 test1_err.txt)를 즉시 삭제한다. [8점]

< 조 건 >

1. compile_program() 함수
   가. 파일 생성은 creat() 만 사용하고, 에러 처리를 해야 함
   나. “예제*”와 sprintf()를 이용하여 “예제*.exe”(문제에서는 test1.exe, test2.exe)을 출력함
   다. sprintf()의 인자로 “예제*.c”와 “예제*.exe”를 사용하여 컴파일 명령어를 출력함
   가. “예제*”와 sprintf()를 이용하여 “예제*_err.txxt”을 출력함
2. redirection() 함수
   가. 기존 파일 디스크립터의 복제본 생성을 위해 dup()과 dup2() 중 임의대로 사용함

test1.c

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

test2.c

```
#include <stdio.h>
int main() {
        printf("Linux System Programing\n")
}
```

13.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#define STDERR 2
#define BUFLEN 512

void compile_program(char *filename);
void redirection(char *command, int new, int old);

int main(int argc, char *argv[])
{
        char buf[BUFLEN];
        if(argc < 2){
                fprintf(stderr, "usage: %s <file1> \n", argv[0]);
                exit(1);
        }

        compile_program(argv[1]);
}

void compile_program(char *filename)
{
        char question_name[BUFLEN];          // 문제이름
        char command[BUFLEN];                // 실행 명령어를 위한 버퍼
        char tmp[BUFLEN];           // 문자열 생성을 위한 임시 버퍼
```

```
        int fd;                                 // 에러 파일의 파일 디스크립터
        int size;                               // 에러 파일의 크기

        strcpy(question_name, filename);
        question_name[strlen(filename) - strlen(strrchr(filename, '.'))] = '\0';

        sprintf(tmp, "%s.exe", question_name);
        sprintf(command, "gcc -o %s %s -lpthread", tmp, filename);

        sprintf(tmp, "%s_error.txt", question_name);

}

void redirection(char *command, int new, int old)
{
        int saved_fd;

        system(command);

}
```

실행결과

```
root@localhost:/home/oslab# ls
a.out    test1.c    test2.c
root@localhost:/home/oslab# ./a.out test1.c
compile success!
root@localhost:/home/oslab# ls
a.out    test1.c    test1.exe    test2.c
root@localhost:/home/oslab# ./a.out test2.c
compile error! error file size: 97
root@localhost:/home/oslab# ls
a.out    test1.c    test1.exe    test2.c    test2_error.txt
root@localhost:/home/oslab# cat test2_error.txt
test2.c: In function ‘main’:
test2.c:4:1: error: expected ‘;’ before ‘}’ token
}
```

14. 다음 프로그램에서 dup2()를 호출하여 “ssu_test.txt” 파일의 디스크립터를 fd=0 으로 복사하여 해당 파일에서 주어진 문자열 스트림 소스에서 지정된 형식으로 데이터를 읽도록(scanf()) 하는 프로그램을 작성하시오. [5점]

14.c

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

#define BUFFER_SIZE 1024

int main()
{
        char buf[BUFFER_SIZE];
        char *fname = “ssu_test.txt“;
        int fd;



        exit(0);
}
```

실행결과

```
oslab@oslab:/home/oslab# echo “Linux System“ > ssu_test.txt
oslab@oslab:/home/oslab# cat ssu_test.txt
Linux System
oslab@oslab:/home/oslab# ./a.out
fd scanf : Linux
fd scanf : System
```

15. 주어진 파일의 접근 시간(atime)과 수정시간(mtime)을 두줄로 출력하는 다음 프로그램을 접근 시간(atime)과 수정시간(mtime)한 줄에 프린트(개행 없이 출력)되게 수정하시오. [7점]

< 조 건 >

1. strncpy()를 한번 사용할 것
2. TIME_STRLEN을 이용할 것

15.c

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

#define TIME_STRLEN 26

struct stat statbuf;

void ssu_accmodtime(char *fname);

int main(int argc, char *argv[])
{
        time_t     acc_time;
        time_t     mod_time;

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

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

        ssu_accmodtime(argv[1]);

        exit(0);
}

void ssu_accmodtime(char *fname)
{
        printf("%s accessed : %s modified : %s", fname, ctime(&statbuf.st_atime), ctime(&statbuf.st_mtime));

        exit(0);
}
```

실행결과

```
test.txt accessed : Mon May 25 16:41:01 20 modified : Mon May 25 16:41:01 2020
```

16. 다음 프로그램 실행 시 프로그램의 인자로 주어진 디렉토리를 읽어 각 파일의 타입이 나오는 프로그램을 작성하시오. [5점]

< 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, 문자 특수 파일, 블록 특수 파일, FIFO 파일, 심볼릭 링크 파일만 있다고 가정
4. 파일 타입은 주어진 print_file_type()를 통해 실행결과와 같이 출력할 것

16.c

```
#define DIRECTORY_SIZE MAXNAMLEN

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

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

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

}
```

실행결과

```
oslab@oslab:~$ cd ssu_dir/
oslab@oslab:~/ssu_dir$ mkdir directory
oslab@oslab:~/ssu_dir$ sudo mknod character c 3 10
oslab@oslab:~/ssu_dir$ sudo mknod block b 3 10
oslab@oslab:~/ssu_dir$ sudo mknod fifo p
oslab@oslab:~/ssu_dir$ ls -al
합계 12
drwxr-xr-x  3 lsp  lsp   4096  5월 25 22:34 .
drwxr-xr-x 17 lsp  lsp   4096  5월 25 22:32 ..
brw-r--r--  1 root root 3, 10  5월 25 22:34 block
crw-r--r--  1 root root 3, 10  5월 25 22:34 charactor
drwxr-xr-x  2 lsp  lsp   4096  5월 25 22:34 directory
prw-r--r--  1 root root    0  5월 25 22:34 fifo
oslab@oslab:~/ssu_dir$ cd ..
oslab@oslab:~$ ./a.out ssu_dir
block     : block special
directory : directory
character : character special
.         : directory
fifo      : FIFO
..        : directory
```

17. 다음 프로그램은 한 줄씩 읽고 라인을 출력하며 라인 수를 계산하는 프로그램이다. 아래 실행결과를 보고 소스코드를 완성하시오. [5점]

< 조 건 >

1. 주어진 변수를 그대로 사용할 것, 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. open(), read(), close()를 각 1번 씩 사용 할 것
4. open()함수에 대한 에러 처리를 위해 fprintf()를 사용할 것
5. 입력 텍스트 파일이 변경되어도 정상적으로 작동되어야 할 것

<ssu_test.txt>

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

17.c

```
#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;


    for (i = 0 ; i < count ; i++)
        printf("%s\n", buf[i]);

    printf("line number : %d \n", count);
    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./ssu_count_sentence
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
```

18. 다음 프로그램은 채점결과 파일 “score.csv”를 기반으로 학생별로 0점을 받은 문제의 “문제파일명”을 출력하고, 0점 받은 문제의 개수를 출력한다. 0점 받은 문제가 없을 경우 “perfect!!”을 출력한다. [5점]

< 조 건 >

1. print_wrong_answer() 함수
   가. “score.csv”의 2번째 행부터 파일이 끝날 때까지 fscanf()를 사용하여 한 행을 읽음
   나. 읽은 행에서 구분자 “,”를 기준으로 추출된 첫 번째 토큰은 학번, 두 번째 토큰부터는 점수를 의미하며 토큰이 더 이상 없을 때까지 반복함.
   다. atof()를 사용하여 점수의 배정밀도를 올려야 함

score.csv
(참고) 채점 결과가 학번,점수,점수,점수,점수, 형태로 출력되는 파일)

```
,1-1.txt,1-2.txt,2.c,3.c,4.c,
20200001,0,10,9.5,0,5,
20200002,10,10,10,0,5,
20200003,0,0,0,0,0,
20200004,10,10,10,10,8.5,
```

18.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define BUFLEN 512
#define QNUM 5
void print_wrong_answer(char *filename);

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

void print_wrong_answer(char *filename)
{
        FILE *fp;                       // 파일 오픈을 위한 FILE 구조체 포인터
        char question_name[QNUM][15];       // 문제파일명 배열 (5문제 제한)
        char line[BUFLEN];          // 파일 내용 저장을 위한 임시 버퍼
        char *ptr;                          // 토큰의 포인터
        int question_num = 0;               // 문제번호
        int error_cnt;                      // 틀린 문제 개수

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

        fscanf(fp, "%s\n", line);
        strcpy(question_name[question_num++], strtok(line, ","));

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

```
                strcpy(question_name[question_num++], ptr);

}
```

실행결과

```
root@localhost:/home/oslab# ./a.out score.csv
20200001's wrong answer: 1-1.txt    3.c
The number of wrong answer: 2
20200002's wrong answer: 3.c
The number of wrong answer: 1
20200003's wrong answer: 1-1.txt    1-2.txt    2.c    3.c    4.c
The number of wrong answer: 5
20200004's wrong answer: perfect!!
```

19. 다음 프로그램은 점수 테이블 파일인 "score_table.csv"을 생성하여 문제당 점수(0.5)를 우선 할당한다. 수정할 문제의 번호와 점수를 표준 입력으로 받아 이를 "score_table.csv"에 반영한다. 단, 점수를 수정할 번호가 "score_table.csv"에 없을 경우 다음 수정할 번호를 계속 입력받는다. [5점]

< 조 건 >

1. modify_scoreTable() 함수
   가. 변경할 문제 번호를 scanf()를 사용하여 입력 받음
   나. score_table.csv에서 문제의 점수를 수정하는 작업은 strcmp()를 사용하여 "n", "N", "no", "NO", "No", "nO" 문자열이 입력으로 들어올 때까지 계속 입력을 받음
   다. getchar()를 사용하여 입력 버퍼를 비움
2. init_scoreTable() 함수
   가. 함수 인자로 들어온 "score_table.csv" 파일 생성은 creat()만 사용하고 에러 처리를 해야 함
   나. sprintf()와 write()를 사용하여 "score_table.csv"에 각 문제 번호(5문제)와 점수를 입력함

19.c

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

#define FILELEN 128
#define QNUM 5
#define BUFLEN 1024

struct ssu_scoreTable{
        char qname[FILELEN];
        double score;
};

struct ssu_scoreTable score_table[QNUM];

void modify_scoreTable(void);
void init_scoreTable(char *filename);

int main(void)
{
        int i;
```

```
        for(i=0; i<QNUM; i++){
                sprintf(score_table[i].qname,"%d", i+1);
                score_table[i].score = 0.5;
        }

        init_scoreTable("./score_table.csv");

        modify_scoreTable();

        init_scoreTable("./score_table.csv");
}

void modify_scoreTable(void)
{
        int i;
        double new_score;                       // 변경될 점수
        char modify_qname[FILELEN];             // 변경하는 문제 번호

        while(1){

        }
}

void init_scoreTable(char *filename)
{
        int num, i;
        char tmp[BUFLEN];                                       // csv 파일에 write할 때 사용되는 임시 문자열
        int fd;                                                 // 파일 디스크립터

        num = sizeof(score_table) / sizeof(score_table[0]);     // csv 파일의 행의 개수

        close(fd);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
Input question's number to modify >> 6 // 문제의 번호가 없는 경우
Input question's number to modify >> 7 // 문제의 번호가 없는 경우
Input question's number to modify >> 2
Current Score : 0.50
New score : 1
Input question's number to modify >> 3
Current Score : 0.50
New score : 1
Input question's number to modify >> no
root@localhost:/home/oslab# cat score_table.csv
1,0.50
2,1.00
3,1.00
4,0.50
5,0.50
```

20. 특정 한 디렉토리 내에 포함된 모든 서브 디렉토리의 파일('.' , '..' 파일은 제외하고 재귀적으로 모든 서브디렉토리를 순회해야 함)들의 실제 파일크기('.' , '..' 파일은 제외한 디렉토리에 포함된 모든 파일 크기의 합)를 출력하는 프로그램을 작성하시오. 단, 아래 조건과 실행 결과를 바탕으로 프로그램을 작성 할 것. (실행결과의 디렉토리 구조는 예시임) [8점]

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
2. access(), stat(), opendir(), readdir()을 사용할 것
3. "tmp"가 디렉토리가 아닐 경우 fprintf()를 이용하여 에러 처리

20.c

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

int sum_size(char *filename);

int main(int argc, char *argv[])
{
        char filename[1024];
        struct stat statbuf;
        int sum;

        strcpy(filename, argv[1]);



        exit(0);
}

int sum_size(char *filename)
{
        int sum = 0;//sum of file size
        struct stat statbuf;
        struct dirent *dentry;
        DIR *dirp;
        char temp[1024];

}
```

실행결과

```
root@localhost:/home/oslab# ls -al
total 44
drwxr-xr-x  3 oslab oslab  4096  5월 25 10:15 .
drwxr-xr-x 32 oslab oslab  4096  5월 25 10:15 ..
-rw-r--r--  1 oslab oslab   470  5월 22 02:28 1.c
-rw-r--r--  1 oslab oslab  1477  5월 25 10:08 3.c
-rw-r--r--  1 oslab oslab  1238  5월 22 04:11 4.c
-rwxr-xr-x  1 oslab oslab 13016  5월 25 10:08 a.out
-rw-r--r--  1 oslab oslab    74  5월 22 02:26 ssu_test.txt
drwxr-xr-x  4 oslab oslab  4096  5월 18 01:37 tmp
root@localhost:/home/oslab# ./a.out ./tmp
```

```
filename : tmp/test/tt          size : 0
filename : tmp/test size : 0
filename : tmp/dir1 size : 238
filename : tmp      size : 1448
```

21. 현재 디렉토리(Present Working Directory) 내 모든 파일('.' , '..' 파일은 제외)의 이름을 출력하는 프로그램을 작성하시오, 단, 아래 조건과 실행 결과를 바탕으로 프로그램을 작성 할 것. [8점]

< 조 건 >

1. opendir(), readdir(), closedir()을 사용할 것
2. 파일이름을 Node로 하는 Linked List로 구현할 것
3. 노드 생성시 동적할당이 안될 경우, fprintf() 이용하여 에러처리

21.c

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

typedef struct list* list_ptr;
typedef struct list{
        char filename[1024];// 파일이름
        list_ptr right;// 링크드 리스트
}list;

list_ptr make_node(char *filename);// 노드생성 함수
void insert_node(list_ptr new_node);// 노드 삽입 함수

list_ptr first; // 리스트 시작
list_ptr last;// 리스트 마지막

int main(int argc, char *argv[])
{
        struct dirent *dirp;
        DIR *dp;
        char path_name[1024];

        exit(0);
}

list_ptr make_node(char *filename)
{
        list_ptr root = (list_ptr)malloc(sizeof(list));

        return root;
}

void insert_node(list_ptr new_node)
{

}
```

실행결과

```
root@localhost:/home/oslab# a.out
tmp
ssu_test.txt
1.c
3.c
```

a.out
4.c

<참고> 코드 작성에 필요한 함수

strrchr : 문자 배열 src 내에 문자 c가 있는지 뒷부분에서부터 검사하여, 문자 c가 있는 번지를 반환하는 함수

```
#include <string.h>
char *strrchr(const char *src, int c);
                                리턴값: 문자열 src에서 발견된 문자 c의 포인터, 문자 c가 발견되지 않을 경우 NULL
```

system : 프로그램 안에서 하나의 명령 문자열을 실행하는 라이브러리 함수

```
#include <stdlib.h>
int system(const char *cmdstring);
                                                리턴값: 성공시 0이 아닌 값, 실패시 0
```

strtok : 문자열에서 token을 찾는 함수

```
#include <string.h>
int strtok(char *s1, char *s2);
                                                리턴값: 성공시 찾아낸 token의 포인터, 실패시 NULL
```

atof: 문자열을 double형 부동 소수점으로 바꾸는 함수

```
#include <stdlib.h>
double atof(const char *s);
                                                리턴값: 성공시 변환된 실수, 실패시 0
```

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

...
```