※ 이번 실습부터 이제 여러분이 직접 cmd.c에 각 명령어를 구현하는 함수를 추가해 보도록 하자.

1. 이제 cmd 프로그램이 실행될 때 나오는 명령 프롬프트를 변경하여 보자. 현재의 프롬프트는 "**$**"로 나오는데 이를 "**</home/jhshim/up/cmd> 23:**" 처럼 현재 작업 디렉토리와 명령어 번호를 출력하도록 변경하여 보자.
   1) 먼저 cmd.c의 맨 앞 쪽 전역변수 선언하는 곳에

   `int argc, optc; //기존의 이 줄 밑에 입력할 것`
   `char cur_work_dir[SZ_STR_BUF];` //현재 작업 디렉토리 이름을 저장하는 버퍼

   위처럼 주석문도 함께 추가 한다. cur_work_dir[] 배열을 전역변수로 선언하는 이유는 나중에 cd 명령어 구현 시 변경된 디렉토리 이름을 여기에 저장하기 위해서이다.
   이제 main() 함수로 내려 가서(G), 아래 문장을 help(); 문장 다음에 추가하라.

   `// 현재 작업 디레토리 이름을 구해 와 cur_work_dir[]에 저장함`
   `// SZ_STR_BUF: cur_work_dir[SZ_STR_BUF]의 크기`
   `getcwd(cur_work_dir, SZ_STR_BUF);`
   ---
   cur_work_dir[]에는 문자열로 저장되어 있다. (강의노트 4장 p.62, 6장 p.16 참조하고, 따라 하지는 마라.)
   2) 이제 명령어 번호를 처리하자. main() 함수의 지역변수 선언하는 곳에 아래 지역변수 선언을 추가한다.

   `int cmd_count = 1;` //현재 명령어 번호

   이 변수는 하나의 명령어가 처리될 때마다 1씩 증가해야 한다. 따라서 "proc_cmd();" 와 "}" 사이에 **이 변수 값을 1 증가시키는 문장을 추가하라**. (강의노트 따라 하지 말 것)
   3) 이제 프롬프트를 출력하는 printf() 문장을 수정하자. 주석문도 수정하고 강의노트는 따라 하지 마라(참조만).

   `// 명령 프롬프트 출력: "<현재작업디렉토리> 명령어번호: "`
   `printf("<%s> %d: ", 변수명 직접 입력할 것);`
   ---
   4) 유닉스 API 함수인 getcwd()가 선언된 헤드파일을 명령창에서 **`$ man -s3 getcwd`**로 확인하고, include되어 있지 않다면 cmd.c의 앞쪽에 include시킨다.
   이처럼 새로운 API 함수를 프로그램에서 호출할 때마다 man 명령어를 실행하여 그 **함수가 선언된 헤드파일**을 찾아 여러분 프로그램에 반드시 include 시켜 주어야 한다.
   예) **`$ man -s3 getcwd`**(찾고자 하는 함수이름)[엔터]하면, 화면에 **`#include <unistd.h>`**가 보일 것이다. 이것이 UNIX API 함수 getcwd()가 선언된 헤드파일이다. 헤드 파일이 여러 개이면 모두 include시키고, 이미 프로그램에 include되어 있으면 include할 필요 없다. 옵션은 **항상 `-s2`**로 주고 만약 해당 함수를 찾을 수 없다고 에러 메시지가 나오면 옵션을 위처럼 **`-s3`**로 지정할 것. 참고로 모든 헤드 파일은 /usr/include 디렉토리 또는 그 하부에 존재한다.
   `$ ls /usr/include`를 해 보라.

   5) 이제 다른 명령 창에서 make하고, cmd를 실행시켜 보자. 명령어 하나를 실행할 때마다 명령어 번호가 증가하고 현재 작업 디렉토리도 프롬프트에 출력되어야 한다. 그냥 엔터 치면 번호가 증가하지 않는다.

2. echo 명령어를 구현하라.
   1) 먼저 기존에 정의된 echo() 함수를 찾기 기능(/echo)으로 찾아가라. 함수 위에 있는 주석을 읽어보라.
   2) echo 명령을 "**`echo This is a test.`**"로 주면,
   `argv[0] -> "This"`
   `argv[1] -> "is"`
   `argv[2] -> "a"`
   `argv[3] -> "test."`
   `argc = 4;` (argv[] 배열 원소의 개수)
   위처럼 echo() 함수가 호출되었을 때는 이미 각 단어별로 분리되어 있고, 각 단어 문자열 시작 주소가 argv[i]에 저장되어 있다. (main()에서 호출하는 get_argv_optv() 함수에 의해 처리 되었음)
   3) 이제 echo() 함수를 구현하자.
   i) 함수 안에 먼저 **`int i;`** 지역변수를 선언하라.
   ii) 변수 i를 증가시키면서 for () 내에서 각 단어 문자열(argv[i])을 출력("**`%s `**")하는 printf() 문장을 삽입하라. %s 다음에 공백문자(' ')가 있음. 배열 원소의 개수는 argc이다.
   iii) for() 문장이 끝나면 줄 바꾸기("\n")을 출력하라.
   4) cmd_tbl[] 배열에 이미 추가된 echo 관련 배열 원소를 확인하라. 인자 개수는 상수 AC_ANY로 개수 제한이 없음을 의미한다.
   5) 이제 명령 창에서 make하여 실행파일 cmd를 생성하고, cmd를 실행시켜라. cmd에서 echo 명령어를 아래처럼 입력해 보자. "**>** "는 cmd의 명령 프롬프트다. 결과가 동일하게 출력되어야 한다.
   `> echo This is a(탭 입력)test.`
   `This is a test.`
   `>`

3. pwd 명령어를 구현해 보자.
   1) pwd 명령어를 처리하는 pwd() 함수를 정의하자. 먼저 찾기기능(/rm)으로 rm 함수를 찾은 후, 함수이름의 알파벳 순서로 pwd() 함수가 들어 갈 곳을 찾는다. 즉, rm() 함수 정의의 **앞에** 아래 함수 정의를 추가하라.

```
----------------------------------------
// 현재 작업 디렉토리 이름을 출력하는 명령어
// 사용법: pwd
void
pwd(void)
{
}
----------------------------------------
```

   함수의 리턴과 인자는 void 인데, 리턴 값이 없고 함수 인자도 없다는 의미이다. **앞으로 모든 명령어 처리 함수는 이런 형태로 만들어라.**
   2) 이제 pwd() 함수를 구현하자.
   현재 작업 디렉토리는 main()에서 getcwd() 호출하여 cur_work_dir[]에 이미 저장해 두었다.
   pwd() 함수 안에 cur_work_dir[]에 저장된 현재 작업 디렉토리 문자열("%s")을 출력하는 **printf() 문장을 추가하라.** printf() 안의 문자열 끝에 줄 바꾸기("\n")도 반드시 추가하라.
   3) 찾기 기능(/cmd_tbl_t)으로 cmd_tbl_t cmd_tbl[ ] 배열을 찾는다. 이 배열에, 아래의 배열 원소를 명령어 알파벳 순서로 "rm" 행 앞에 삽입하라.

```
----------------------------------------
{ "pwd", pwd, 0, "", "" },
----------------------------------------
```

   각 칸(column)의 위치도 탭을 사용하여 **다른 행의 열과 맞추어라.** 위 배열원소는 명령어 이름 "pwd", 이 명령어를 처리할 함수이름 pwd(= 함수주소 = 함수 첫 명령어 주소), 명령어 인자의 개수 0, 옵션은 없고(""), 명령어 사용법을 출력할 때 사용할 인자도 없음("")을 의미한다.
   새로운 명령어가 구현될 때마다 그 명령어 관련 정보를 cmd_tbl[ ]에 추가해야 한다.
   4) 이제 명령 창에서 make 하여 실행파일 cmd 를 생성하고, cmd 를 실행시킨 후 명령어로 pwd 를 입력해 보자. 위 2)의 printf() 속에 '\n'이 있을 때와 없을 때의 출력 결과를 확인하라.

4. hostname 명령어를 구현해 보자.
   1) 실습 3. 1) pwd()처럼 hostname() 함수를 정의하자.

```
----------------------------------------
// 현 컴퓨터의 이름을 출력해 주는 명령어
// 사용법: hostname
void hostname(void) {} // 3.1) 함수 형태로 할 것
----------------------------------------
```

   2) 강의노트 6장 pp.15~16, 교재 p.233(206)를 참조하여 hostname() 함수를 구현하자.
      i) 함수 **안에서** 먼저 아래처럼 배열을 선언한다
      `char hostname[SZ_STR_BUF];`
      ii) gethostname() API 함수를 호출하라. 이 함수는 컴퓨터 이름을 구해 와 hostname[]에 저장한다.
      iii) 실습 3. 2)의 printf() 문장을 참조하여 hostname[]에 저장된 문자열을 출력하라.
      iv) 함수 hostname()과 배열 hostname[]의 이름이 같은데 문제 없을까?
   3) 실습 3. 3)을 참조하여 cmd_tbl[] 배열에 hostname과 관련한 배열원소를 추가하라.
   4) API 함수인 gethostname()이 선언된 헤드파일을 명령어 $ man으로 확인하고 include시킨다.
   5) make, cmd 실행; cmd의 명령어로 hostname를 입력해 보자. cmd를 종료하고 명령 창에서 $ hostname을 실행하라. cmd와 결과가 동일한가?

5. whoami 명령어를 구현해 보자.
   1) 실습 3. 1) pwd()처럼 whoami() 함수를 정의하자.

```
----------------------------------------
// 사용자의 계정 이름을 출력하는 명령어
// 사용법: whoami
void whoami(void) {} // 3.1) 함수 형태로 할 것
----------------------------------------
```

   2) 강의노트 8장 pp.57~58, 교재 p.339를 참조하여 whoami() 함수를 구현하자. [강의노트 6장 p.16 참조]
      i) 함수 안에 먼저 username이라는 변수를 선언하라.
      `char *username;`
      ii) 강의노트를 참조하여 getlogin() API 함수를 호출한 후 리턴 값을 username에 저장하라. username은 계정이름을 저장하고 있는 문자열 시작주소이다.
      iii) if username이 NULL이 아니라면, 실습 3. 2)의 printf() 문장을 참조하여 username이 포인터하는 문자열을 출력하라. (강의노트 printf() 문장 참조)
      iv) else // NULL이면

```
printf("터미널 장치가 아니라서 사용자 계정정보를 구할 수 없습니다.\n");
```

   3) cmd_tbl[ ] 에 whoami 관련 배열원소를 추가하라.
   4) getlogin()이 선언된 헤드파일을 include시킨다.
   5) make, cmd 실행하여 whoami를 테스트하라.

6. 명령 창에서 다음을 실행하라.

```
$ eshw 5          // 에러가 발생한 항목만 보여 줌
$ progtest 5      // 실습 5 전체를 테스트함
$ progtest 5 2 // 실습 5의 2번 문제만 테스트함
      // 원하는 문제를 골라 테스트 할 수 있음
```

Homework

※ 아래 코드는 ls 명령어를 구현한 함수들이다. 기존의 cmd.c에 지시대로 추가시켜라.

1. 아래 두 문장을 cmd.c의 앞쪽에 있는 include 문장들 중 맨 마지막에 추가시켜라.

```
#include <sys/stat.h>
#include <dirent.h>
#include <pwd.h>
#include <grp.h>
#include <time.h>
```

2. 아래 코드를 get_argv_optv(char *cmd_line) 함수 정의({ }) 뒤에,
   즉 "여기서부터 각 명령어 구현 시작" 주석문 앞쪽에 추가시켜라.

> **<<<<<<<<<< 주의 >>>>>>>>>>**
>
> 아래 코드의 인덴트(여백공간)는 8자 크기로 보여 주는데, 사실은 탭 글자이다. cmd.c에 추가할 때는 인덴트 부분을 **반드시 탭 키**를 이용하라. 그러면 여기서 보이는 인덴트 길이보다는 4글자 정도의 작게 보일 것이다.

```
//*************************************************************************
// ls() 함수에서 호출되는 함수들 시작
//
// "ls -l" 옵션 준 경우 하나의 파일에 대한 상세정보 출력하기
static void
print_attr(char *path, char *fn)
{
        struct passwd *pwp;
        struct group  *grp;
        struct stat st_buf;
        char full_path[SZ_STR_BUF], buf[SZ_STR_BUF], c;
        char time_buf[13];
        struct tm *tmp;

        sprintf(full_path, "%s/%s", path, fn);
        if (lstat(full_path, &st_buf) < 0)
                PRINT_ERR_RET();

        if      (S_ISREG(st_buf.st_mode))    c = '-';
        else if (S_ISDIR(st_buf.st_mode))    c = 'd';
        else if (S_ISCHR(st_buf.st_mode))    c = 'c';
        else if (S_ISBLK(st_buf.st_mode))    c = 'b';
        else if (S_ISFIFO(st_buf.st_mode))   c = 'p';
        else if (S_ISLNK(st_buf.st_mode))    c = 'l';
        else if (S_ISSOCK(st_buf.st_mode))   c = 's';
        buf[0] = c;
        buf[1] = (st_buf.st_mode & S_IRUSR)? 'r': '-';
        buf[2] = (st_buf.st_mode & S_IWUSR)? 'w': '-';
        buf[3] = (st_buf.st_mode & S_IXUSR)? 'x': '-';
        buf[4] = (st_buf.st_mode & S_IRGRP)? 'r': '-';
        buf[5] = (st_buf.st_mode & S_IWGRP)? 'w': '-';
        buf[6] = (st_buf.st_mode & S_IXGRP)? 'x': '-';
        buf[7] = (st_buf.st_mode & S_IROTH)? 'r': '-';
        buf[8] = (st_buf.st_mode & S_IWOTH)? 'w': '-';
        buf[9] = (st_buf.st_mode & S_IXOTH)? 'x': '-';
        buf[10] = '\0';
        pwp = getpwuid(st_buf.st_uid);
        grp = getgrgid(st_buf.st_gid);
        tmp = localtime(&st_buf.st_mtime);
        strftime(time_buf, 13, "%b %d %H:%M", tmp);
```

```
        // 아래의 " " 안의 l은 영문 소문자 L임; 모든 % 개수 정확히 입력할 것
        sprintf(buf+10, " %3ld %-8s %-8s %8ld %s %s",
                        st_buf.st_nlink, pwp->pw_name, grp->gr_name,
                        st_buf.st_size, time_buf, fn);
        if (S_ISLNK(st_buf.st_mode)) {
                int len, bytes;
                strcat(buf, " -> ");
                len = strlen(buf);
                bytes = readlink(full_path, buf+len, SZ_STR_BUF-len-1);
                buf[len+bytes] = '\0';
        }
        printf("%s\n", buf);
}

// "ls -l" 옵션 준 경우: 디렉토리의 모든 파일에 대해 상세정보 출력하기
static void
print_detail(DIR *dp, char *path)
{
        struct dirent *dirp;

        while ((dirp = readdir(dp)) != NULL)
                print_attr(path, dirp->d_name);
}

// "ls"만 한 경우: 가장 긴 파일이름의 길이를 계산한 후
//              이를 기준으로 한 줄에 출력할 수 있는 열(column)의 개수를 결정함
static void
get_max_name_len(DIR *dp, int *p_max_name_len, int *p_num_per_line)
{
        struct dirent *dirp;
        int max_name_len = 0; // 가장 긴 파일이름 길이

        // 모든 파일 이름을 읽어 내 가장 긴 이름의 길이를 결정함
        while ((dirp = readdir(dp)) != NULL) {
                int name_len = strlen(dirp->d_name);
                if (name_len > max_name_len)
                        max_name_len = name_len;
        }
        // 디렉토리 읽는 위치 처음으로 되돌리기
        rewinddir(dp);
        // 가장 긴 파일이름 + 이름 뒤의 여유 공간
        max_name_len += 4;
        // 한 줄에 출력할 파일이름의 개수 결정
        *p_num_per_line = 80 / max_name_len;
        *p_max_name_len = max_name_len;
}

// "ls"만 한 경우: 디렉토리의 모든 파일에 대해 이름만 출력하기
static void
print_name(DIR *dp)
{
        struct dirent *dirp;
        int max_name_len, num_per_line, cnt = 0;

        // max_name_len: 가장 긴 파일이름 길이 + 4(이름 뒤의 여유 공간)
        // num_per_line: 한 줄에 출력할 수 있는 총 파일이름의 개수
        get_max_name_len(dp, &max_name_len, &num_per_line);

        while ((dirp = readdir(dp)) != NULL) {
                printf("%-*s", max_name_len, dirp->d_name);
```

```
                // cnt: 현재까지 한 줄에 출력한 파일이름 개수
                // 한 줄에 출력할 수 있는 파일이름의 개수만큼
                // 이미 출력 했으면 줄 바꾸기
                if ((++cnt % num_per_line) == 0)
                        printf("\n");
        }
        // 마지막 줄에 줄 바꾸기 문자 출력
        // 앞에서 이미 한 줄에 출력할 수 있는 파일이름의 개수만큼 이미 출력하여
        // 줄 바꾸기를 했으면 또 다시 줄 바꾸기 하지 않음
        if ((cnt % num_per_line) != 0)
                printf("\n");
}
//
// ls() 함수에서 호출되는 함수들 끝
//*****************************************************************************
```

3. 아래 코드를 echo(void) 와 rm(void) 두 함수 사이에 있는 기존 void ls(void) 함수를 다음과 같이 수정하라.

```
void
ls(void)
{
        char *path;
        DIR     *dp;

        // 디렉토리 이름을 주지 않았다면 현재 디렉토리 설정
        path = (argc == 0)? ".": argv[0];

        if ((dp = opendir(path)) == NULL)     // 디렉토리 열기
                PRINT_ERR_RET();
        if (optc == 0)
                print_name(dp);
        else
                print_detail(dp, path);
        closedir(dp);                         // 디렉토리 닫기
}
```

4. 위 코드를 모두 삽입했으면 make를 실행시켜라. 아마 많은 에러가 발생할 것이다. 이러한 에러는 당연히 발생하는 것이라 생각하고 천천히 컴파일 에러를 수정해라. 그래도 수정이 되지 않으면 실습 시간에 교수님께 협조를 구하라.

5. 이제 cmd를 실행시켜 다음의 명령어를 입력해 보라. "**>** "는 cmd의 명령 프롬프트다.

```
> ls                    // 현재 디렉토리 파일 이름 목록
> ls -l                 // 현재 디렉토리 파일 상세 목록
> ls ../pr4             // ../pr4 디렉토리 파일 이름 목록
> ls -l ../pr4          // ../pr4 디렉토리 파일 상세 목록
> ls -l -a              // 옵션 에러
> ls ../jhshim          // 디렉토리 없음 에러
> ls /home/jhshim       // 접근권한 에러
> ls a b c              // 인자 개수 에러
```

7. 명령 창에서 다음을 실행하라.

```
$ eshw 6 0          // 실습 6의 0번 문제만 체크하되 에러가 발생한 항목만 보여 줌
$ progtest 6 0      // 실습 6의 0번 문제만 테스트함
```