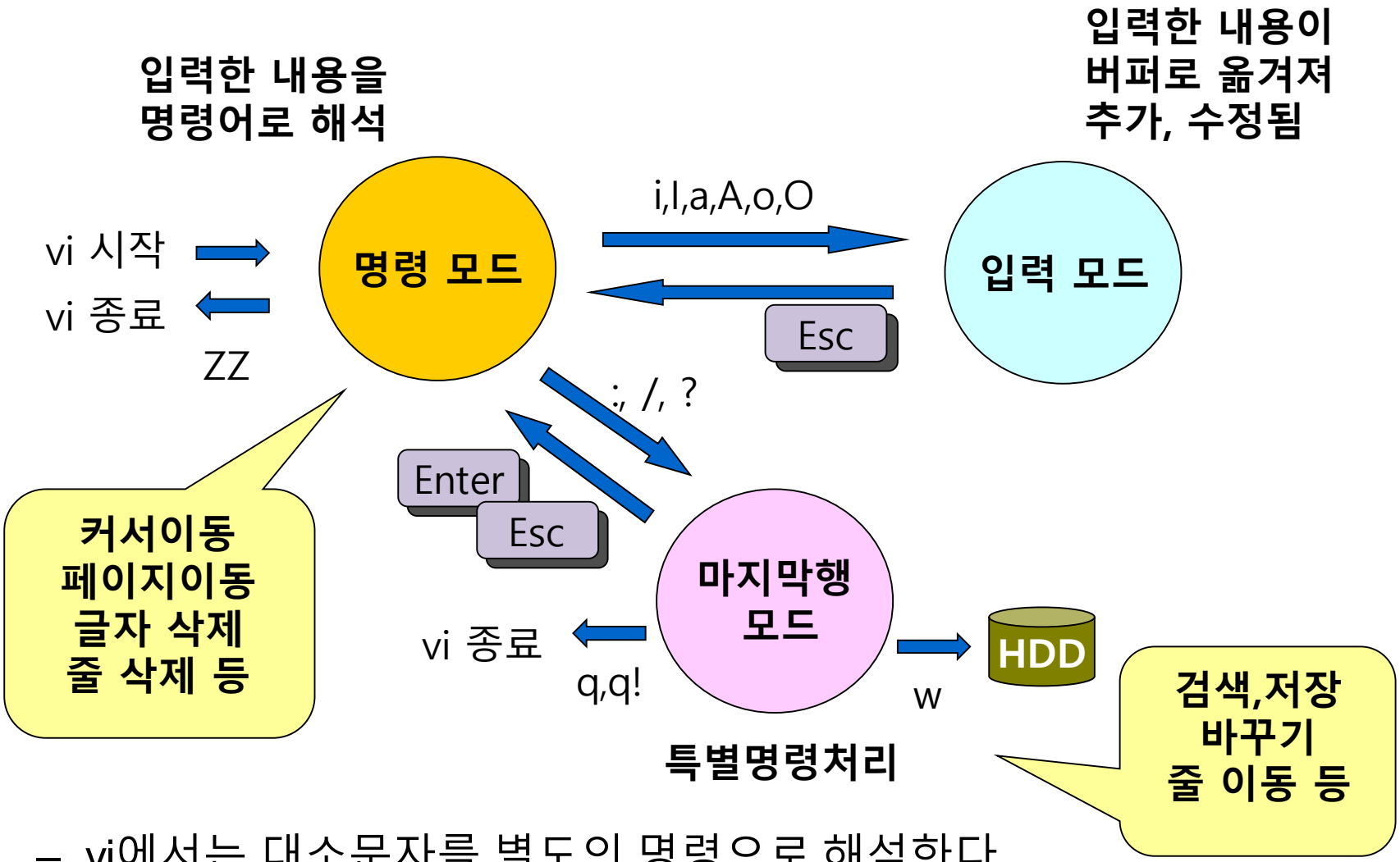
C programming on UNIX

VI 사용법

- vi 란 ?
  - UNIX의 대표적 텍스트 편집기
  - 모드형 편집기 (명령모드, 입력모드)
- vi 사용법
  - vi filename <enter>
  - 파일이 존재하면 기존 파일 편집
  - 파일이 존재하지 않으면 파일을 생성하고 편집

vi의 동작 모드



– vi에서는 대소문자를 별도의 명령으로 해석한다

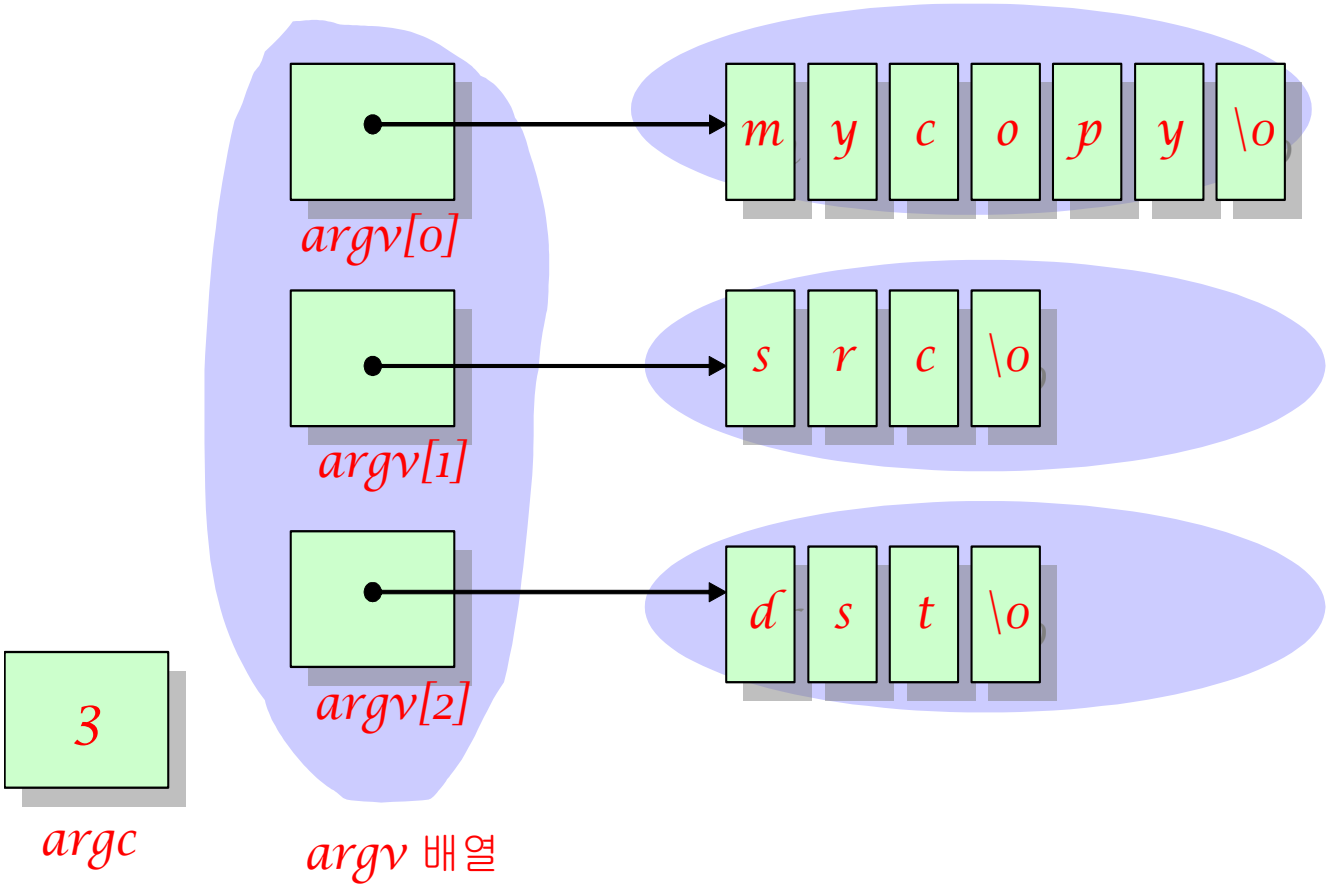
main() 함수의 인수

-
```
int main(void)
{
..
}
```

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

인수 전달 방법

```
$ mycopy src dst
```



main_arg.c

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
	int i = 0;

	for(i = 0;i < argc; i++)
		printf("명령어 라인에서 %d번째 문자열 = %s\n", i, argv[i]);

	return 0;
}
```

```
$ mainarg src dst
명령어 라인에서 0번째 문자열 = mainarg
명령어 라인에서 1번째 문자열 = src
명령어 라인에서 2번째 문자열 = dst
$
```

opt_arg.c

```
// 명령어의 옵션과 인수를 구분하는 프로그램

#include <stdio.h>

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

    // 첫 번째 인수를 제외한 모든 인수들에 대해
    for (i=1; i < argc; i++) {
        // 명령라인 인수의 첫 글자가 '-'면 옵션
        if(argv[i][0] == '-')
                printf("option : %s\n", argv[i]+1);
        //그렇지 않으면 인수
        else
                printf("argument : %s\n", argv[i]);
    }
}
```

UNIX 환경에서 C 프로그래밍

- **vi 사용하기**
  - 편집기 시작
    - vi filename.c
  - mode editor
    - 명령모드 -> 입력모드 (i, a, …)
    - 입력모드 -> 명령모드 (esc)
  - 저장 후 나가기: 명령모드에서 ZZ

- **gcc (GNU C compiler) 사용하기**
  - 사용법
    - gcc src_filename.c: 실행파일 (a.out) 생성
    - gcc src_filename.c –o filename: 실행파일 (filename) 생성
  - 실행 파일 수행
    - ./a.out 혹은 ./filename

파일 입출력 (File I/O)

- **파일 입출력 절차**
  - 파일 열기 (file open) -> 파일 입출력 -> 파일 닫기 (file close)
- **파일 열기 라이브러리 함수 및 시스템 호출**
  - fopen()
- **파일 입출력 라이브러리 함수 및 시스템 호출**
  - fgetc(), fputc(), fgets(), fputs(), fread(), fwrite(), fprintf(), fscanf() ....
- **파일 닫기 라이브러리 함수 및 시스템 호출**
  - fclose()

fopen(const char *path, const char *mode)

fopen 함수
기능
파일을 연다.
기본형
FILE *fopen(const char *path, const char *mode);
path : 열고자 하는 파일 이름
mode : 사용 형태
반환값
파일 포인터를 반환한다. 오류면 NULL을 반환한다.
헤더 파일
<stdio.h>

mode	의미
"r"	읽기 전용. 파일이 존재해야 함
"w"	쓰기 전용. 파일이 없어도 되고 만약 이미 존재한다면 기존 내용은 지워짐
"a"	추가용. 파일이 없어도 되고 만약 이미 존재한다면 기존 내용 뒤에 추가됨
"rb"	바이너리 파일의 읽기 전용
"wb"	바이너리 파일의 쓰기 전용
"ab"	바이너리 파일의 추가용
"r+"	읽기와 쓰기용
"w+"	쓰기와 읽기용

fclose(FILE *stream)

fclose 함수
기능
파일을 닫는다.
기본형
int fclose(FILE *stream);
stream : 닫고자 하는 파일 포인터
반환값
0을 반환한다. 오류면 EOF를 반환한다.
헤더 파일
<stdio.h>

파일 열기 – file_open.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    FILE *fp;
    char str[200];

    fp = fopen(argv[1],"r");

    if (fp == NULL) {
        printf("Can't open %s₩n", argv[1]);
        return -1;
    }
    else
        printf("%s is open₩n", argv[1]);

    fscanf(fp, "%s", str);
    printf("%s₩n", str);

    fclose(fp);
}
```

fgetc(FILE *fp), fputc(int c, FILE *fp)

fgetc 함수
기능
　　파일을 통해 한 문자를 입력받는다.
기본형
　　int fgetc(FILE *stream);
　　stream : 파일 포인터
반환값
　　입력받은 문자를 반환한다. 파일의 끝이면 EOF를 반환한다.
헤더 파일
　　<stdio.h>

fputc 함수
기능
　　파일에 한 문자를 출력한다.
기본형
　　int fputc(int c, FILE *stream);
　　c : 출력할 문자
　　stream : 파일 포인터
반환값
　　출력한 문자를 반환한다. 오류면 EOF를 반환한다.
헤더 파일
　　<stdio.h>

File Contents Display Program – mycat.c

```
#include <stdio.h>

int main(int ac, char *av[])
{
    FILE *fp;
    int ch;

    if( (fp = fopen(av[1], "r")) == NULL) {
        printf("Can't open %s₩n", av[1]);
        return -2;
    }

    while((ch=fgetc(fp)) != EOF)        /* fp 파일로부터 한 문자 읽기 */
        fputc(ch, stdout);               /* 읽은 문자 표준 출력으로 보내기 */

    fclose(fp);                         /* 파일 닫기
    return 0;
}
```

File Copy Program – file_copy.c

```
#include <stdio.h>

int main(int ac, char *av[])
{
        FILE *ifp, *ofp;
        int ch;

        if (ac != 3) {
                printf("Usage: %s   src_filename dst_filename₩n", av[0]);
                return -1;
        }

        if( (ifp = fopen(av[1], "r")) == NULL) {
                printf("Can't open %s₩n", av[1]);
                return -2;
        }

        if( (ofp = fopen(av[2], "w")) == NULL) {
                printf("Can't open %s₩n", av[2]);
                return -3;
        }

        while((ch=fgetc(ifp)) != EOF)                   /* ifp 파일로부터 한 문자 읽기 */
                fputc(ch, ofp);                          /* 읽은 문자 ofp 파일에 쓰기 */

        fclose(ifp);  fclose(ofp);                      /* 파일 닫기
        return 0;
}
```

디렉토리 다루기

- 디렉토리 생성: mkdir(const char *path, mode_t mode)
- 디렉토리 삭제: rmdir(const char *path)
- 디렉토리 이동: chdir(const char *path)
- 현재 작업 디렉토리 얻기: getcwd(char *buf, size_t size)

- 디렉토리 열기 -> 디렉토리 관련 동작 -> 디렉토리 닫기
- 열기: DIR *opendir(const char *path)
- 디렉토리 관련 동작: readdir(DIR *dp), rewinddir(DIR *dp)
- 닫기: closedir(DIR *dp)

디렉토리 생성과 삭제

```
mkdir 함수
기능
        디렉토리를 만든다.
기본형
        int mkdir(const char *pathname, mode_t mode);
        pathname : 만들고자 하는 디렉토리 이름
        mode : 만들고자 하는 디렉토리의 접근 권한
반환값
        성공 : 0
        실패 : -1
헤더 파일
        <sys/stat.h>
        <sys/types.h>
        <fcntl.h>
        <unistd.h>
```

```
rmdir 함수
기능
        디렉토리를 삭제한다.
기본형
        int rmdir(const char *pathname);
        pathname : 삭제하고자 하는 디렉토리 이름
반환값
        성공 : 0
        실패 : -1
헤더 파일
        <unistd.h>
```

디렉토리 이동

chdir, fchdir 함수
기능
　　작업 디렉토리를 이동한다.
기본형
　　int chdir(const char *path);
　　int fchdir(int fd);
　　path : 이동하고자 하는 디렉토리 이름
　　fd : 이동하고자 하는 디렉토리를 의미하는 식별자
반환값
　　성공 : 0
　　실패 : -1
헤더 파일
　　<unistd.h>

getcwd 함수
기능
　　현재 작업 디렉토리 이름을 얻어온다.
기본형
　　char *getcwd(char *buf, size_t size);
　　buf : 현재 작업 디렉토리 이름을 저장하는 공간
　　size : 디렉토리 이름의 예상되는 길이
반환값
　　성공 : 현재 작업 디렉토리 이름
　　실패 : NULL
헤더 파일
　　<unistd.h>

디렉토리 다루기 예제 (dir.c)

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

int main(int ac, char *av[])
{
    char dname[80];                                    // getcwd()에서  현재 디렉토리 경로 저장

    if (ac != 2 {                                      // 인수가 1개가 아닐 경우  에러처리
        printf("Usage: %s  dir_name\n", av[0]);
        return -1;
    }

    if( ( mkdir(av[1], 0755) {                         // 755 모드로 디렉토리 생성
        printf("Failed to make directory %s\n", av[1]);
        return -2;
    }

    if( ( chdir(av[1]) {                               // 만든 디렉토리로 이동
        printf("Failed to change directory %s\n", av[1]);
        return -3;
    }

    getcwd(dname, 80);                                 // 현재 디렉토리 경로 얻기
    printf("Current working directory is %s.\n", dname);

    return 0;
}
```

디렉토리 열기/닫기

opendir 함수
기능
　　디렉토리를 연다.
기본형
　　DIR *opendir(const char *name);
　　name : 열고자 하는 디렉토리 이름
반환값
　　성공 : DIR 구조체 포인터
　　실패 : NULL
헤더 파일
　　<sys/types.h>
　　<dirent.h>

closedir 함수
기능
　　디렉토리를 닫는다.
기본형
　　int closedir(DIR *dir);
　　dir : 닫고자 하는 디렉토리 정보에 대한 포인터
반환값
　　성공 : 0
　　실패 : -1
헤더 파일
　　<sys/types.h>
　　<dirent.h>

디렉토리 읽기

readdir 함수
기능
　　디렉토리를 읽는다.
기본형
　　struct dirent *readdir(DIR *dir);
　　dir : 읽고자 하는 디렉토리
반환값
　　성공 : struct dirent의 포인터
　　실패 : NULL
헤더 파일
　　<sys/types.h>
　　<dirent.h>

■ struct dirent 데이터형

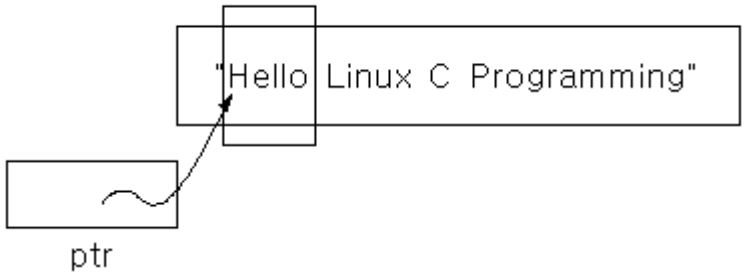
```
struct dirent {
   ino_t d_ino;  /* inode 번호 */
   off_t d_off;  /* 디렉토리 내에서의 오프셋 */
   unsigned short d_reclen;  /* 항목의 레코드 길이 */
   char d_name[NAME_MAX+1];  /* 파일 이름 */
};
```

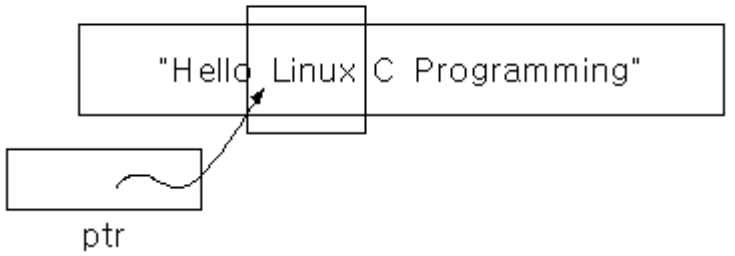
디렉토리 내의 항목 출력하기 (readdir.c)

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

int main(int argc, char *argv[])
{
    DIR *dp;                    /* 디렉토리 정보에 대한 포인터 */
    struct dirent *dirp;        /* 디렉토리의 한 항목의 정보에 대한 포인터 */

    if((dp=opendir(argv[1])) == NULL) {                    /*  디렉토리 열기 */
        printf("opendir failed");
        return 1;
    }
        /* dp 디렉토리의 항목들을 하나씩 읽어서 항목 이름을 출력한다.
        더 이상 읽을 항목이 없으면 NULL이 반환된다. */
    while (dirp=readdir(dp))
        printf("%s \t", dirp->d_name);
    printf("\n");

    closedir(dp);   /* dp 디렉토리 닫기 */
    return 0;
}
```

strtok() 함수: 문자열을 자르는 함수

strtok 함수
기능
문자열에서 구분자와 일치하는 문자가 나오면 단어로 자른다.
기본형
char *strtok(char *s, const char *delim);
s : 문자열
delim : 구분자 집합
반환값
단어의 첫 문자의 포인터를 반환한다. 더 이상 단어가 없으면 NULL을 반환한다.
헤더 파일
<string.h>





strtok() 사용예 (strtok.c)

```
#include <stdio.h>
#include <string.h>   /* strtok 함수가 정의된 헤더 파일 */

main()
{
   char str[]="Hello Linux C Programming";
   /* 어떤 구분자를 기준으로 단어로 자를지를 지정하는 데 이 프로그램에서는
     공백 문자(" ")를 이용하고 만약 구분자로 ;, : 을 이용하려면 ";:"을 지정하면 됨 */
   char delim[]=" ";
   char *ptr;

   /* 문자열 str에서 delim 구분자가 나오면 단어로 자르고 단어의 첫 글자에 대한
      포인터 반환 */
   ptr = strtok(str, delim);
   printf("%s\n", ptr);
   /* 두 번째 호출부터는 NULL로 설정해야 함 */
   while(ptr=strtok(NULL, delim))
      printf("%s\n", ptr);
}
```

쉘의 기본 구조 (simple_sh.c)

```
#include <stdio.h>
int main()
{
        char str[200];

        while(1){
                printf("prompt$ ");        // 프롬프트 출력
                gets(str);                 // 사용자 입력 1라인 읽기
                if ( !strcmp(str,"exit") )// exit이 입력되면 종료
                        return 0;
                if (strlen(str) == 0)      // 입력된 문자열이 없을 경우
                        continue;
                else     // 쉘에서는 입력된 명령을 수행하기 위한 코드 삽입
                         // 본 예제에서는 입력된 문자열을 단순히 출력함
                        puts(str);
        }
}
```