6. C Programming on Linux

Hyunchan, Park

http://oslab.jbnu.ac.kr

Division of Computer Science and Engineering

Jeonbuk National University

학습 내용

- VI 사용법: 추가
- GCC: GNU Compiler Collection
- 표준 입출력
- 명령행 인자

개인 과제 6: 실습

- 실습 과제
  - 실습 내용에서 다루는 명령어를 모두 입력하고, 그 결과를 확인할 것
    - 동영상에서 수행한 내용
    - 슬라이드에 캡처된 결과물들은 모두 포함되어야 함
  - 제출 방법
    - Old LMS, 과제 6
    - Xshell 로그 파일 1개 제출
    - 파일 명: 학번.txt
- 제출 기한
  - 10/19 (월) 23:59 (지각 감점: 5%p / 12H, 1주 이후 제출 불가)

Vi 사용법: 추가

라인 복사 및 삭제

- 모든 명령어는 insert mode 에서 esc 키를 눌러 일반 모드로 나온 후, 수행
- 라인 복사 명령 : yy
  - 앞에 숫자를 입력하면, 현재 커서가 위치한 라인을 포함한 아래의 다수 라인을 한번에 “레지스터”로 복사함
- 라인 삭제 명령 : dd
  - 앞에 숫자를 입력하면, 현재 커서가 위치한 라인을 포함한 아래의 다수 라인을 한번에 “레지스터” 로 복사하고, 제거함
- 레지스터의 붙여넣기 : p
  - 현재 커서가 있는 곳에서부터 레지스터의 내용을 삽입함
- VI Register
  - VI에서 복사한 내용이 임시로 보관되는 공간.
  - VI 프로그램 간에 공유됨.
    - 따라서 VI가 종료되어도, 다시 VI를 수행하면 레지스터의 내용을 이용할 수 있음
    - 이 기능은 한 파일의 내용을 복사해서 다른 파일에 붙여넣을 때 유용함

라인 이동 및 관련 명령

- 라인 이동
  - 사용법 1: “:” 입력 후, 이동할 라인 숫자 입력
  - 사용법 2: 라인 숫자를 입력하고 Shift + g

- 관련 명령
  - 라인의 맨 앞으로 이동하기: 0 (숫자 영)
  - 맨 위로 이동하기: gg
  - 맨 밑으로 이동하기: “:$” or “(입력없이) shift + g”
  - 줄 번호 표시 하기 : “: set number”

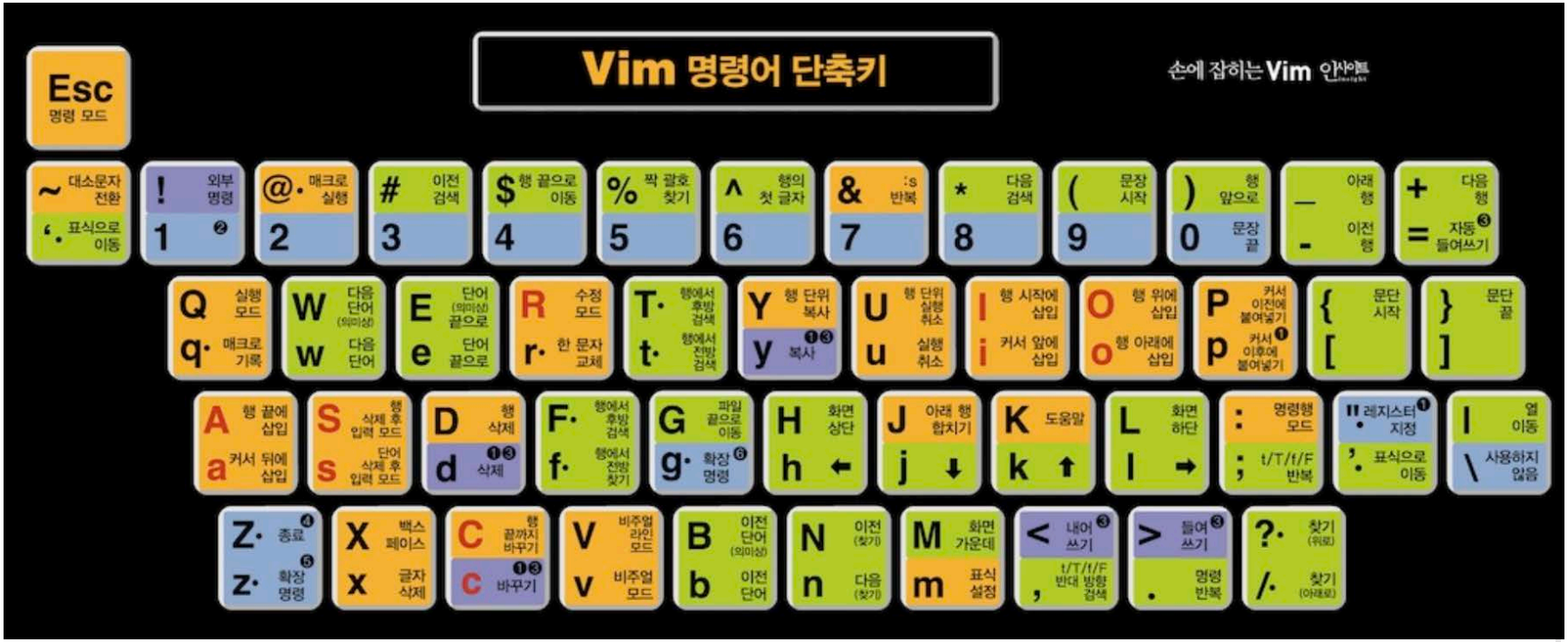
문자열 찾기 / 바꾸기

- 문자열 찾기
  - “/”를 입력하고 찾을 문자열 입력
  - Enter 입력 후, 다음 단어, 이전 단어 검색
    - 소문자 n: 다음 단어
    - 대문자 N: 이전 단어
  - 이전에 찾아본 문자열 불러오기
    - “/”를 입력한 상태에서 위 아래 화살표 사용
- 문자열 바꾸기
  - :[범위]s/찾을문자열/바꿀문자열/[option]
  - 범위: comma 를 이용해 범위 표현. % 는 전체 영역
    - 예) 1,10: 첫 번째부터 10번째 라인 내에서 수행.
  - 찾을 문자열에는 정규 표현식 사용 가능 (regular expression 으로 검색)
  - Options
    - g: 범위 내에서 바꾸기 수행
    - c: 한 항목씩 물어보면서 수행
    - i: 대소문자 무시
  - 예) :%s/Protocol/protocol/gc

기타

- Undo/Redo (취소하기, 되돌리기)
  - u: undo
  - ^r: Redo (CTRL + r)
- 세로 및 가로 블록 선택, 편집
  - ^v: Visual block mode (CTRL + v)
  - 모드 진입 후, 화살표로 선택 후, 편집 명령
  - 예) 여러 라인에 있는 주석을 한번에 제거
    - 세로 모드로 여러 주석 문자를 선택 후 delete
  - 예) 여러 라인에 주석 한번에 넣기
    - 세로 모드로 영역 선택 후, shift + i 로 입력 모드 진입
    - 텍스트 입력 후, esc 를 두 번 누름
- Read-only 파일의 저장
  - :w 혹은 :wq 뒤에 ! 를 붙임 (force)
- 외부 텍스트 붙여넣기 모드
  - :set paste

VI 사용법은 이만하면 충분함!



* https://gmlwjd9405.github.io/2019/05/14/vim-shortkey.html

- “에디터와 같은 도구에 얽매이지 말고, 얼마나 빠르게 결과물을 낼 수 있을지만 고민할 것”
- VI는 기본 유닉스 에디터로, 간단한 에디팅 정도만 수행할 수 있으면 충분함
- 실제 개발은 대부분 리눅스 시스템을 원격으로 연결한 윈도우 환경에서 이루어질 것

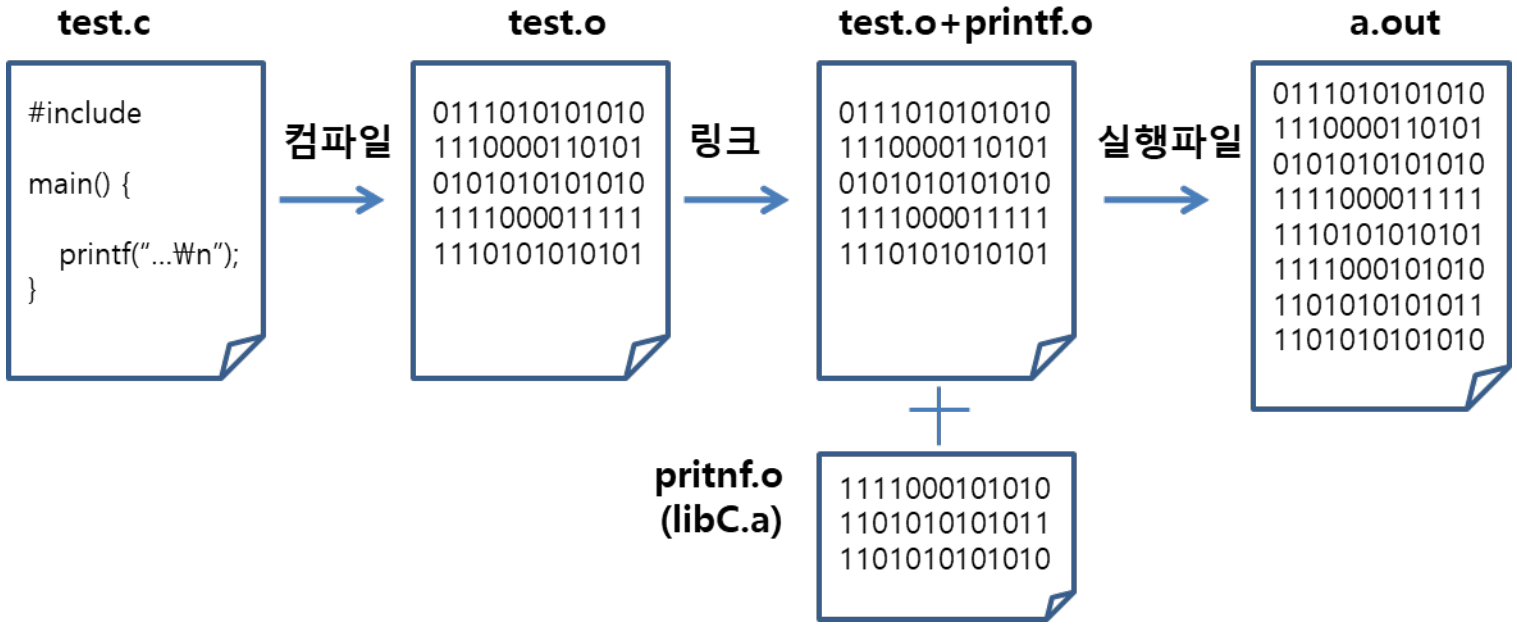
nano

- nano - Nano's ANOther editor, inspired by Pico
  - 전통적으로 메일 클라이언트에서 사용하던 Pico 라는 편집기를 기반으로,
  - Vi와 같이 여러 리눅스 배포판에서 기본 프로그램으로 사용함
  - Vi와 비교: “사용하기 쉽다.”
    - 대부분 단축키가 아래에 나열되어 있어 모르는 기능도 빠르게 활용할 수 있다.
    - “검색하여 교체하기” (replace) 기능이 보다 편리하게 사용 가능 하다.
    - 자동 들여쓰기 기능이 더 편리하다.
- Comments
  - Vi, nano 모두 기본 에디터이므로 간단한 사용 방법은 익혀두는 것이 좋다.
  - Vi, nano 모두 간단한 편집 기능은 큰 차이 없으므로, 익숙한 도구를 쓰면 된다.
  - 초심자가 처음 배운다면?
    - Nano: 접근성이 높다
    - Vi: 처음 배울 때 어려울 수 있지만, 보다 다양하고 강력한 기능들을 제공한다.
  - (!) 결국 다양하고 많은 편집을 필요로 할 때는 윈도우 환경이 훨씬 편리함.
    - 실제 파일은 리눅스에, 편집은 윈도우(or Mac)에서 수행하는 환경을 구축하는 것이 일반적

GCC: GNU Compiler Collection

컴파일 환경

- 컴파일이란
  - 텍스트로 작성한 프로그램을 시스템이 이해할 수 있는 기계어로 변환하는 과정
  - 보통 컴파일 과정과 라이브러리 링크 과정을 묶어서 수행하는 것을 의미



GCC: GNU Compiler Collection

- GNU
  - 유닉스 환경에서 필수적인 다양한 시스템 소프트웨어를 공개 SW 형태로 제작, 배포하는 그룹
  - 1983년 부터 활동하며 다수의 SW를 배포하였고, 대다수 SW가 유닉스 환경에서 de facto standard 로 활용되고 있음
    - De facto standard: 사실상의 표준. 관습, 관례, 제품이나 체계가 시장이나 일반 대중에게 독점적 지위를 가진 것
  - GNU 패키지 목록
    - 일상적으로 사용하는 다양한 명령어들이 포함되어 있음 (bash, grep, gzip, tar, …)

GCC: GNU Compiler Collection

- GCC: GNU Compiler Collection
  - GNU SW 중 가장 유명한 SW의 하나로,
    다양한 Architecture (CPU) 환경에서 다양한 언어를 지원함
  - C, C++, Objective-C, Fortran, Ada, Go, and D
  - 위 언어를 위한 라이브러리도 포함
  - 상용 컴파일러와 비교해 성능이 낮다는 인식이 있었으나,
    최근에는 많은 상용 레벨 SW를 위한 컴파일러로 널리 활용하고 있음
    - Linux, MySQL, Apache 등등
  - https://gcc.gnu.org/
  - Git repository: official, github
  - 설치 방법 (j-cloud 인스턴스에서는 수행할 필요 없음)
    - 패키지 업데이트 후, SW 빌드를 위한 필수 패키지 설치. 개발을 위한 manpage 추가
    - $ sudo apt update && sudo apt install build-essential
    - $ sudo apt-get install manpages-dev

사용 방법

- $ gcc <source file>
  - Output: 컴파일 성공 시, “a.out” executable file (실행 파일) 생성
- Options
  - “-o” : 생성된 실행 파일의 이름을 지정
  - “-Wall” : 모든 레벨의 warning messages 출력
  - “-O” : optimization 수행. “-O1”, “-O2”, “-O3” 와 같이 최적화 레벨을 지정할 수 있음
    - https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html
  - “-l” : (소문자 L) 라이브러리 링크. Math, pthread 와 같이 명시적 링크가 필요한 경우

```
$ gcc test.c
$ ls
a.out test.c
```

기본 실행파일명은
a.out

```
$ gcc -o test test.c
$ ls
test   test.c
```

실행파일명 지정은
-o 옵션

1st program: hello world

```
ubuntu@41983:~$ cd hw1
ubuntu@41983:~/hw1$ vi hw1.c
ubuntu@41983:~/hw1$ gcc hw1.c
ubuntu@41983:~/hw1$ ls -al a.out
-rwxrwxr-x 1 ubuntu ubuntu 16688 Sep 22 15:34 a.out
ubuntu@41983:~/hw1$ a.out
a.out: command not found
ubuntu@41983:~/hw1$ ./a.out
Hello World
ubuntu@41983:~/hw1$
```

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

int main (void) {

        printf("Hello World\n");
        return 0;
}
```

- 참고: 실행 파일의 실행 시에는 앞에 디렉토리 위치를 지정해주어야 함
  - 명령어 이름이 중복될 수 있으니, 수행하려는 명령을 명확하게 하기 위함
  - 기존에 사용했던 명령어들은?
    실행파일들이 위치한 경로명이 환경 변수 PATH에 지정되어 있어, 따로 지정할 필요가 없음
  - 환경 변수? 차후에 더 자세히 다룰 것

```
ubuntu@41983:~/hw1$ env | grep PATH
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
ubuntu@41983:~/hw1$
```

-l option

```
ubuntu@41983:~/hw1$ vi hw1.c
ubuntu@41983:~/hw1$ gcc -o hello hw1.c
/usr/bin/ld: /tmp/ccSGxQz9.o: in function `main':
hw1.c:(.text+0x54): undefined reference to `pow'
collect2: error: ld returned 1 exit status
ubuntu@41983:~/hw1$ gcc -o hello hw1.c -lm
ubuntu@41983:~/hw1$ ./hello
Please input an integer: 9
9^2 = 81.000000
ubuntu@41983:~/hw1$
```

```
SYNOPSIS
       #include <math.h>

       double pow(double x, double y);
       float powf(float x, float y);
       long double powl(long double x, long double y);

       Link with -lm.
```

```c
#include <stdio.h>
#include <math.h>

int main (void) {
        int num;

        printf("Please input an integer: ");

        scanf("%d", &num);

        printf("%d^2 = %f\n", num, pow(num,2));

        return 0;
}
```

- Man page 에 나오는대로 include 도 했는데, undefined reference?
- 적절한 library 를 –l 옵션을 이용해 링크해주어야 함
  - 예) math (-lm) , pthread (-lpthread)

Standard Input and Outputs

stdout and stderr

```
ubuntu@41983:~/hw1$ vi hw1.c
ubuntu@41983:~/hw1$ gcc -o hello hw1.c && ./hello
Hello World (stdout)
Hello World (stderr)
ubuntu@41983:~/hw1$ ./hello > stdout.txt
Hello World (stderr)
ubuntu@41983:~/hw1$ ./hello 2> stderr.txt
Hello World (stdout)
ubuntu@41983:~/hw1$ cat stdout.txt
Hello World (stdout)
ubuntu@41983:~/hw1$ cat stderr.txt
Hello World (stderr)
ubuntu@41983:~/hw1$
```

```c
#include <stdio.h>

int main (void) {

        fprintf(stdout, "Hello World (stdout)\n");
        fprintf(stderr, "Hello World (stderr)\n");
        return 0;
}
```

- fprintf()
  - printf() 와 유사하게 형식이 지정된 문자열 (formatted string)을 출력하되,
  - 맨 앞의 인자로 출력 방향을 지정할 수 있음
    - printf()는 fprintf()의 simple version. 실제로 fprintf(stdout, …) 으로 구현됨

```
SYNOPSIS
       #include <stdio.h>

       int printf(const char *format, ...);
       int fprintf(FILE *stream, const char *format, ...);
```

stdin

```
ubuntu@41983:~/hw1$ vi hw1.c
ubuntu@41983:~/hw1$ vi input.txt
ubuntu@41983:~/hw1$ cat input.txt
HELLO
ubuntu@41983:~/hw1$ gcc -o hello hw1.c
ubuntu@41983:~/hw1$ ./hello
Please input some characters: abcdefg

Received: abcdefg
ubuntu@41983:~/hw1$ cat input.txt | ./hello
Please input some characters:
Received: HELLO
ubuntu@41983:~/hw1$
```

```c
#include <stdio.h>

int main (void) {
        char buff[80];

        fprintf(stdout, "Please input some characters: ");

        scanf("%s", buff);

        fprintf(stdout, "\nReceived: %s\n", buff);
        return 0;
}
```

- Pipe 를 이용한 stdin 입력
  - Scanf()는 본래 stdin 으로 부터 입력을 받는 함수
  - Stdin 은 기본으로 console 을 통한 키보드 입력으로 연결되어 있음
  - Pipe를 이용해 cat 의 수행 결과를 stdin 으로 입력받은 것
- Stdin, stdout, stderr 의 redirection 을 이용해,
  여러 프로그램 간의 편리한 연동이 가능함

명령행 인자

명령행 인자

- 명령행 : 사용자가 명령을 입력하는 행 (command line)
- 명령행 인자 : 명령을 입력할 때 함께 지정한 인자(옵션, 옵션인자, 매개변수 등)
  - 명령행 인자는 main 함수로 전달됨.
  - Main 함수의 첫 번째 인자: 인자의 개수 (보통 int argc 로 선언함. Argument count)
  - Main 함수의 두 번째 인자: 문자열로 된 인자들이 저장된 포인터 배열
    - 보통 char *argv[] 또는 char **argv 로 선언함. Argument vector
    - 명령어는 항상 첫 번째 인자

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

```
ubuntu@41983:~/hw2$ vi arg.c
ubuntu@41983:~/hw2$ gcc -o arg arg.c
ubuntu@41983:~/hw2$ ./arg 123 456
argc = 3
argv[0] = ./arg
argv[1] = 123
argv[2] = 456
ubuntu@41983:~/hw2$
```

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int n;
    printf("argc = %d\n", argc);
    for (n = 0; n < argc; n++)
        printf("argv[%d] = %s\n", n, argv[n]);
    return 0;
}
```

명령행 인자

- 포인터 배열?
  - 다양한 길이의 문자열이 임의의 개수만큼 저장되는 경우,
  - 포인터 배열로 다루는 것이 적합함

```
ubuntu@41983:~/hw2$ ./arg 12345 abc 6789
argc = 4
argv[0] = ./arg
argv[1] = 12345
argv[2] = abc
argv[3] = 6789
```

```
ubuntu@41983:~/hw2$ ./arg "how are you" "12345 678"
argc = 3
argv[0] = ./arg
argv[1] = how are you
argv[2] = 12345 678
```

```
ubuntu@41983:~/hw2$ ./arg 1234567 abc "ha ha ha"
argc = 4
argv[0] = ./arg
argv[1] = 1234567
argv[2] = abc
argv[3] = ha ha ha
```

```
ubuntu@41983:~/hw2$ ./arg 1234 567 ab cdefg hi j klm nop
argc = 9
argv[0] = ./arg
argv[1] = 1234
argv[2] = 567
argv[3] = ab
argv[4] = cdefg
argv[5] = hi
argv[6] = j
argv[7] = klm
argv[8] = nop
```