CSE3013-02 / AIE3091-02
소프트웨어 개발도구 및 환경실습

2025년 1학기, 4주차 4/1 화요일

조교(강의자) 소개 및 연락처

- 박한나(메인 조교)
  - 인공지능학과 석사과정
  - **채점** 문의: crescent.check@gmail.com

- 이강호
  - 컴퓨터공학과 석사과정
  - **출결** 문의: lee7roho@sogang.ac.kr

>예비보고서/결과보고서/과제.. 관련 질문은 '사이버캠퍼스(eclass.sogang.ac.kr) - 질의응답'<

빠른 채점을 위해 도움주셨음 하는 점 (저희 분반 한정)

1. 보고서 ONLY `.pdf`
    a. 한글, docs 다 pdf 파일로 변환해서 제출해주세요
2. **압축 X**
    a. 예비보고서 : 0주차_예비보고서_학번_이름.확장자
    b. 결과보고서 : 0주차_결과보고서_학번_이름.확장자
    c. 실습 : 0주차_실습_학번_이름.확장자
    d. 과제 : 0주차_과제_학번_이름.확장자

[오늘의 실습]

shell programming

목차

이번 실습의 효능

- 보통의 서버들은 Unix/Linux 기반입니다.
  - GPU가 돌아갈 딥러닝 서버
  - AWS(Amazon Web Service) / GCP(Google Cloud Platform)에서의 인스턴스
  - ...
  - Shell Script로 서버 세팅 편하게 한번에 할 수 있음 (컴퓨터 시스템 업데이트, 와이파이 설정 변경, 자동 백업, 하드디스크 공간 확인 후 조치 취하기)

- Makefile -> 보통 컴파일 언어(C계열, …) 빌드할 때 주로 사용
  - 일종의 스크립트 모음집
  - Makefile속 명령어를 target을 정해서 넣을 수 있음. 근데 그러면서 발생되는 의존성 문제를 해결해줌

매우 도움이 될 다른 학교 수업 자료

https://missing.csail.mit.edu/
여러분의 CS 교육에서 누락된 학기

Schedule

- **1/13/20**: Course overview + the shell
- **1/14/20**: Shell Tools and Scripting
- **1/15/20**: Editors (Vim)
- **1/16/20**: Data Wrangling
- **1/21/20**: Command-line Environment
- **1/22/20**: Version Control (Git)
- **1/23/20**: Debugging and Profiling
- **1/27/20**: Metaprogramming
- **1/28/20**: Security and Cryptography
- **1/29/20**: Potpourri
- **1/30/20**: Q&A

The Missing Semester of Your CS Education

Classes teach you all about advanced topics within CS, from operating systems to machine learning, but there's one critical subject that's rarely covered, and is instead left to students to figure out on their own: proficiency with their tools. We'll teach you how to master the command-line, use a powerful text editor, use fancy features of version control systems, and much more!

Students spend hundreds of hours using these tools over the course of their education (and thousands over their career), so it makes sense to make the experience as fluid and frictionless as possible. Mastering these tools not only enables you to spend less time on figuring out how to bend your tools to your will, but it also lets you solve problems that would previously seem impossibly complex.

Read about the motivation behind this class.

Shell Script

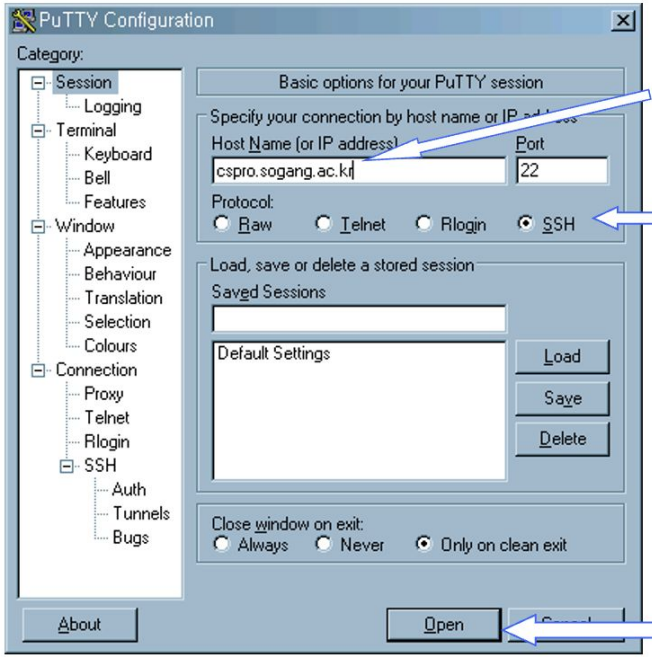
Putty 기준으로 채점한다 공지했지만,,,,

- Windows
  - WSL(Windows Subsystem for Linux) 깔려있다면 그거 ok
  - 윈도우 터미널 (Powershell)을 쓰는법도 방법
- MacOS
  - iterm
  - 기타 프로그램,,,(warp, ...),,,,

Unix System 기초

■ Putty를 이용한 접속 Session Save 기능 활용하기

▼ Unix 서버에 접속 할 때 주로 이 프로그램을 많이 이용한다.



1. 주소창에 cspro.sogang.ac.kr 을 입력한다.

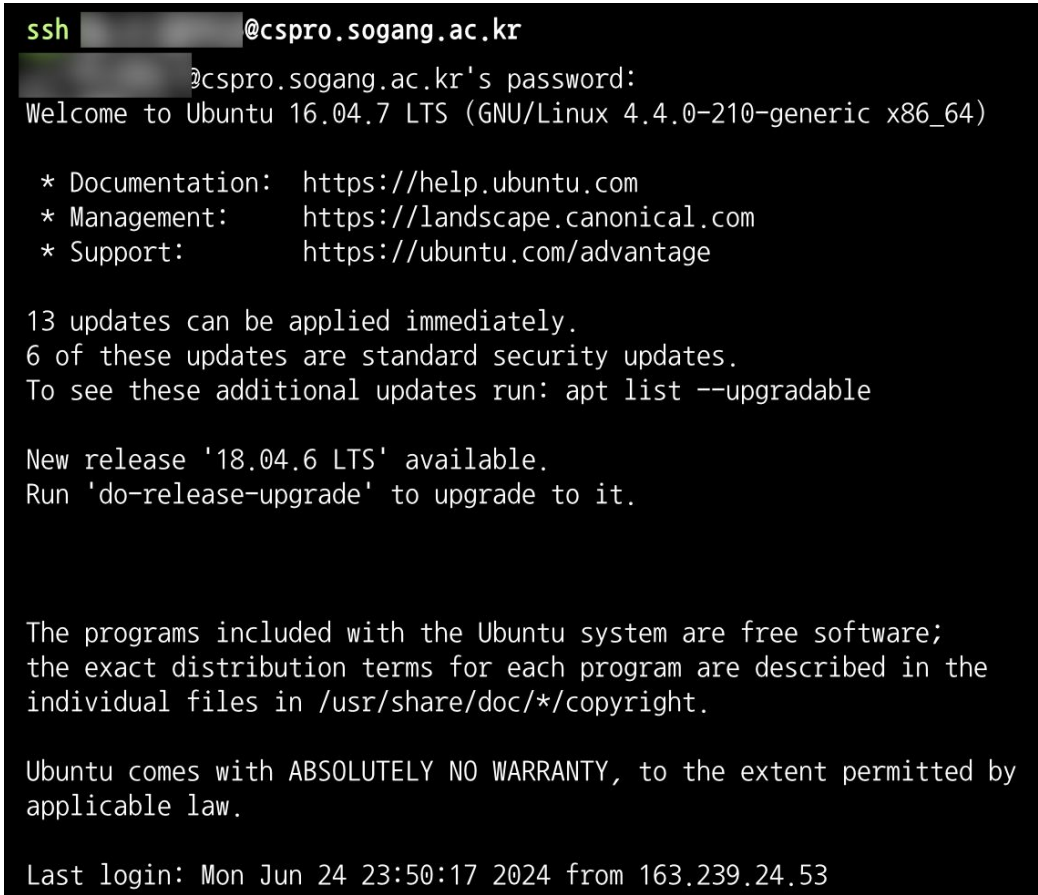
2. ssh 프로토콜을 선택한다.

3. Open 버튼을 누른다.

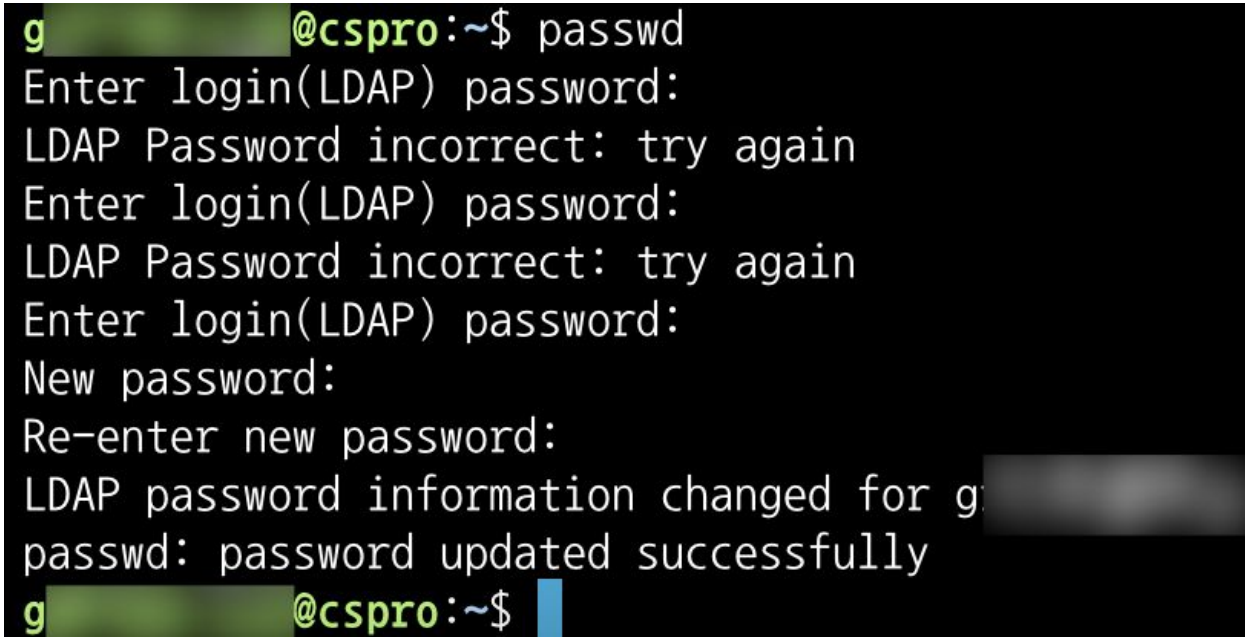
터미널에서 ssh로 서버 연결

`ssh 학번@cspro.sogang.ac.kr`

비밀번호 입력

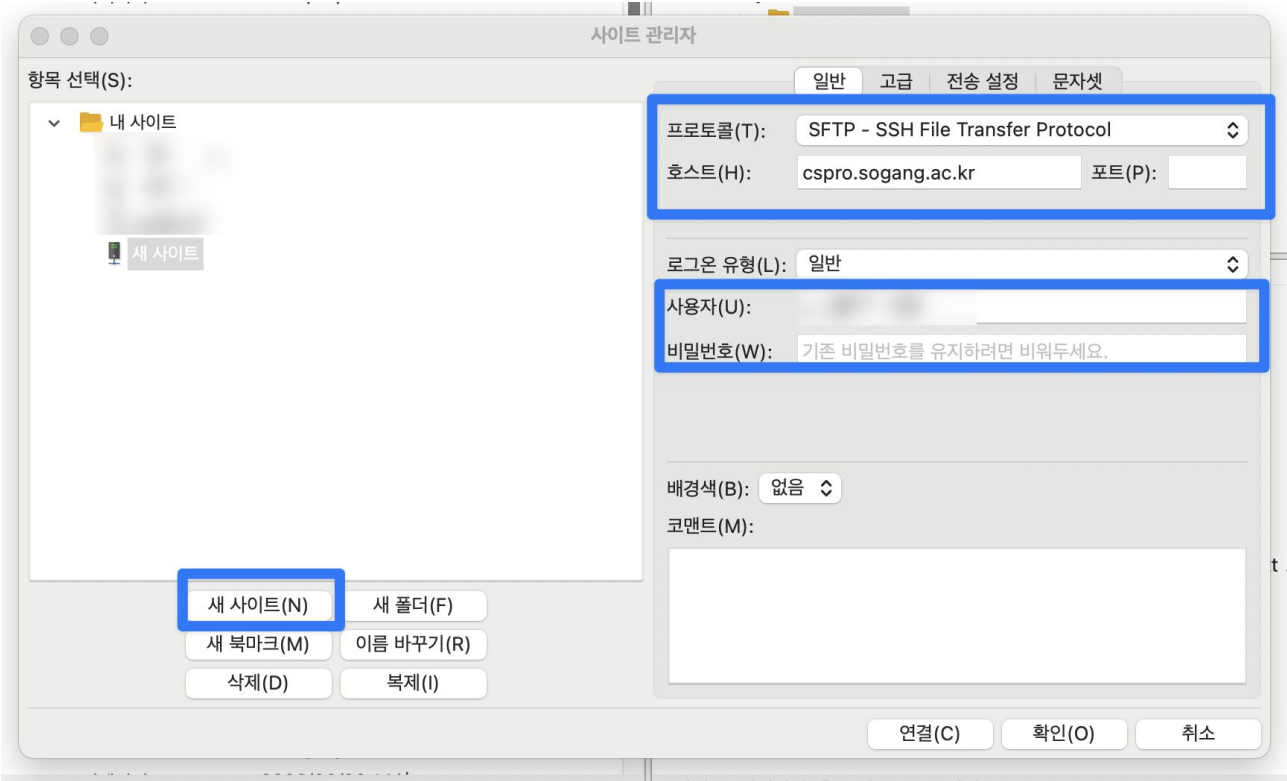


비밀번호 바꾸는 법 -> `passwd`



서버에 파일 쉽게 옮기는 법 - filezilla

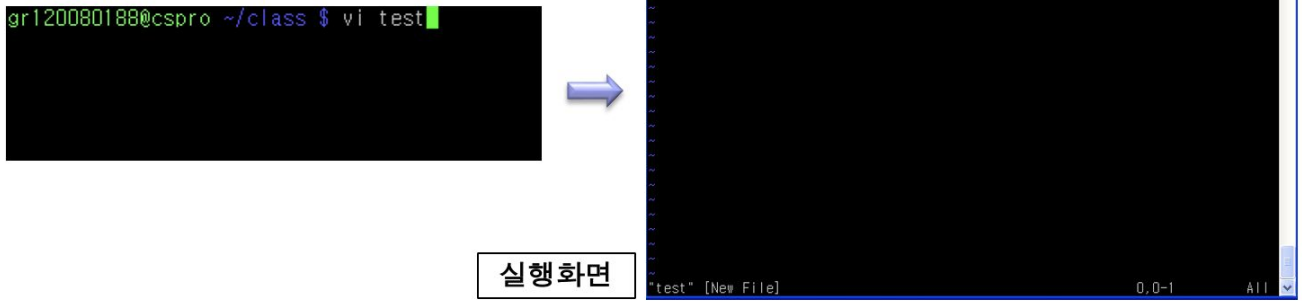
https://filezilla-project.org/



vi 설정 및 사용법

■ vi 기본 사용법

- ▼ vi는 Visual display editor로 유닉스 상에서 가장 널리 쓰이는 텍스트 편집기임.
- ▼ Unix의 vi가 소스코드가 공개되지 않은 관계로 Linux 상에서는 vi 역할을 하는 Vim (Vi Improved)이라는 텍스트 편집기를 사용. linux에서 vi 명령어를 치면 vim이 실행됨.
- ▼ 보통 "vi 파일이름"으로 실행시킨 후 저장 후 종료하면 해당 파일이 생성됨.

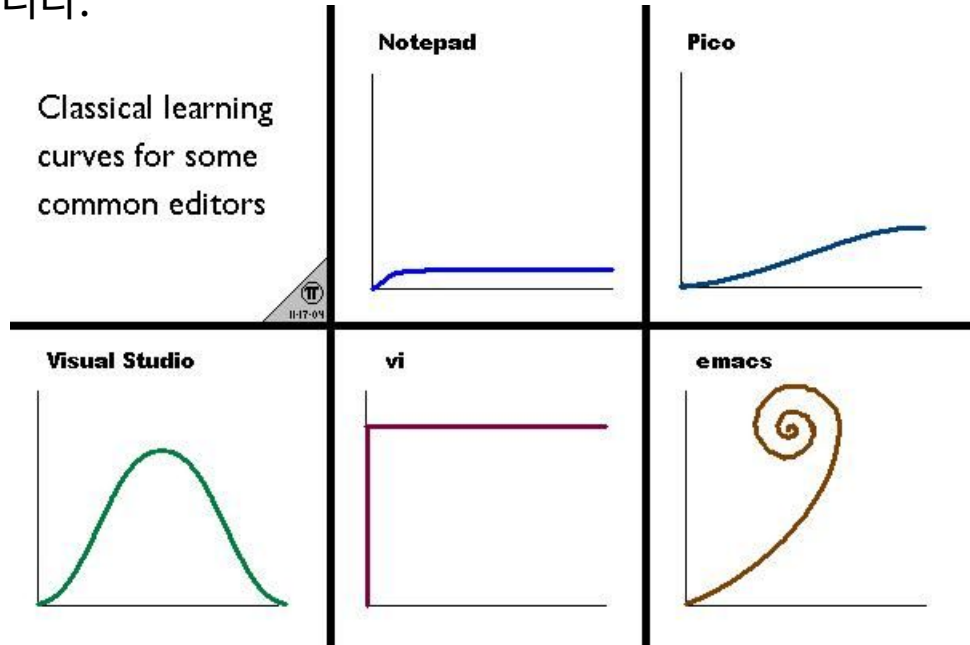


실행화면

vi

vi 대신 vim(Vi IMproved) 쓰세요.

- https://www.vim.org/
- 따로 안깔아도 됩니다. 학교 서버 vim 지원합니다.
  - nano, emacs도 깔려있어요.
-



vi/vim 어려워요

- [esc] + [i] → insert mode (현재 커서 위치에서 입력 모드)
- [esc] + [:wq] → 저장 후 나가기
  - [esc] + [ZZ]
  - [esc] + [:x]
- [esc] + [:q!] → 저장 안하고 **강제로** 나가기

[esc] 눌러 왼쪽 밑에 `-- INSERT --`가 안보이는 상황에서

- [gg] → 파일 맨 위로 이동
  - [G] → 파일 맨 끝으로 이동
- [dd] → 현재 줄 삭제
- [u] → undo

Makefile

Use Makefile

- 만약 관리해야 하는 소스코드가 수십, 수백개라면?

```
gcc –c main.c
gcc –c fun1.c
gcc –c fun2.c
…
gcc –c fun100.c
gcc –o program main.o fun1.o fun2.o … fun100.o
```

현실적으로 어려움

- Makefile을 사용

자주 나오는 에러

1. Makefile:17: *** missing separator. Stop.
*Makefile을 작성할 때 명령어(command)부분은 모두 TAB 문자로 시작해야 한다*고 첫 번째 장부터 강조하였다. 위의 에러는 *TAB* 문자를 쓰지 않았기 때문에 make가 명령어인지 아닌지를 구별 못하는 경우이다.
대처: 17번째 줄(근처)에서 명령어가 *TAB* 문자로 시작하게 바꾼다.
2. make: *** No rule to make target `io.h', needed by `read.o'. Stop.
위의 에러는 의존 관계에서 문제가 발생했기 때문이다. 즉 read.c가 io.h에 의존한다고 정의되어 있는데, io.h를 찾을 수 없다는 에러이다.
대처: 의존 관계에서 정의된 io.h가 실제로 존재하는지 조사해 본다. 없다면 그 이유를 한번 생각해 본다. *make dep*를 다시 실행시켜서 의존 관계를 다시 생성시켜 주는 것도 하나의 방법이다.
3. Makefile:10: *** commands commence before first target. Stop.
위의 에러는 '첫 번째 타겟이 나오기 전에 명령어가 시작되었다'는 애매한 에러 메시지이다. 필자가 경험한 이 에러의 원인은 주로 긴 문장을 여러 라인에 표시를 하기 위해서 '₩'를 사용할 때, 이를 잘못 사용했기 때문인 것 같다. 즉 '₩'부분은 라인의 가장 끝문자가 되어야 하는데 실수로 '₩'뒤에 스페이스를 몇 개 집어넣으면 여지없이 위의 에러가 발생한다.
대처: 10번째 줄(근처)에서 '₩'문자가 있거든 이 문자가 라인의 가장 끝문자가 되도록 한다. 즉 '₩'문자 다음에 나오는 글자(스페이스가 대부분) 는 모조리 없애 버린다.
4. make를 수행시키면 의도했던 실행 파일은 안생기고 이상한 행동만 한다. 가령 make clean 했을 때와 같은 행동을 보인다.
make는 천재가 아니라는 점을 생각해야 한다. *make는 Makefile의 내용을 읽다가 첫 번째 타겟으로 보이는 것을 자신이 생성시켜야 할 결과 파일이라고 생각한다.* 따라서 clean 부분을 Makefile의 첫번째 타겟으로 정해 버리면 위와 같은 결과가 나타나게 된다.
대처: 예제 7.1에서 all 이라는 필요 없는 타겟을 하나 만들어 두었다. 이것은 make가 all 을 첫 번째 타겟으로 인식시키기 위함이었다. 따라서 자신이 생성시키고 싶은 결과 파일을 첫 번째 타겟이 되게 하던지, 아니면 예제 7.1처럼 all과 같은 더미 타겟(dummy target)을 하나 만들어 둔다. 그리고 make clean, make dep 같은 부분은 Makefile의 끝부분에 만들어 두는 것이 안전하다.
5. 이미 컴파일했던 파일을 고치지 않았는데도 다시 컴파일한다.
이 행동은 make가 의존 관계를 모르기 때문이다. 즉 사용자가 의존 관계를 설정해 주지 않았다는 말이 된다. 따라서 make는 무조건 모든 파일을 컴파일해서 실행 파일을 만드는 일이 자신이 할 일이라고 생각하게 된다.
대처: 목적 파일, 소스 파일, 헤더 파일들의 의존 관계를 설정해 주어야 한다. *gccmakedep *.c 라고 하면 Makefile의 뒷부분에 자동적으로 의존 관계를 만들어 준다.* 그외의 다른 파일들에 대해서는 사용자가 적절하게 의존 관계를 설정해 주어야 한다.
main.o : main.c io.h read.o : read.c io.h write.o : write.c io.h

gcc 사용시 발생할 수 있는 에러 메세지들
https://sfixer.tistory.com/entry/GCC-Error-Message-List

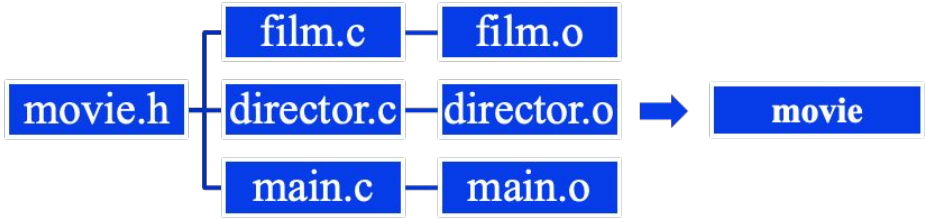
Makefile

일종의 빌드용 스크립트

순서관계 세팅 가능 -> 의존성이 있는 스크립트를 작성할 때 필요

📚

- GNU make
- Chapter 18. Managing More Code with Make | Red Hat Product Documentation



4주차 실습

1. 온라인 전화/주소록

전화번호와 주소록이 기록되어 있는 데이터 파일이 있다. 사람의 이름, 주소, 전화번호의 일부분이 입력으로 주어지면 데이터 파일에서 입력과 맞는 데이터를 검색하여 포맷에 맞게 출력한다. 스크립트 파일명은 “phone”으로 한다.

입출력 형식

- 데이터 파일 형식: 각 줄마다 하나의 레코드가 “이름|주소|전화번호” 로 입력된다.
- 입력형식: 쉘프롬프트 상에서 한 개 이상의 인자(argument)로 주어진다.
- 출력형식: 검색된 레코드를 아래 예와 같이 출력한다. 만약 검색된 레코드가 없다면 아무것도 출력하지 않는다.

데이터 파일 예 -> mydata.txt / mydata

```
1  홍길동|서울시 마포구 신수동 서강대학교 AS관 301호|02-705-2665
2  박문수|서울시 서대문구 신촌동 연세대학교 연구관 102호|02-708-4678
3  Andrew|경기도 의정부시 호원동 23-12번지|031-827-7842
4  한승현|경기도 고양시 일산동구 장항2동 호수마을 407호 1602호|031-904-5643
5  신봉선|경기도 고양시 일산동구 마두동 206동|031-813-3803
```

1. 온라인 전화/주소록

mydata 파일을 기반으로 저렇게 되어야 함.

입력예

```
./phone  홍길동  신수동
```

출력예

```
------->
name: 홍길동
address: 서울시 마포구 신수동 서강대학교 AS관 301호
phone: 02-705-2665
<-------
```

1주차 실습

- **문제 해결 방법**
  - 에러처리: 반드시 하나이상의 인자를 필요로 하며, 해당 인자가 주어지지 않은 경우 다음과 같이 사용 예(usage)를 출력해 준다.

    입력예 `./phone`

    출력예
    ```
    Usage: phone searchfor [… searchfor]
    (You didn't tell me what you want to search for.)
    ```

  - 입력 인자 변경: egrep 을 사용하기 위해 모든 인자를 다 붙여서 하나의 문자열로 만들어야 한다.

    ```
    egrep –i "(arg1|arg2|…|argn)" datafile
    과 같이 사용되어야 하므로
    ./phone arg1 arg2 … argn
    에서 (arg1|arg2|…|argn) 으로 고쳐줘야 한다
    ```

  - 해당 데이터 추출 및 출력: egrep을 통해 원하는 데이터를 추출하고 포맷에 맞게 출력해야 한다. 포맷을 맞추는 것은 awk 프로그램을 이용할 수 있다. Field의 구분자로 "|"를 사용해야 하며 이것은 BEGIN {} 내에 FS="|"; 를 넣어주면 된다.
  - Stripping off the head of a pathname: basename (e.g. name="`basename $0`")

awk

- Programming language designed to make many common information retrieval and text manipulation tasks easy to state and to perform

- Basic operation: scan a set of input lines in order, searching for lines which match any of a set of patterns which the user has specified

  예 **{print $1, $2, $3}**

- Usage
  - awk 'program' [datafile]
  - awk –f program [datafile]

1. 온라인 전화/주소록

`display.awk`을 활용해서 출력예가 저런 형식으로 나와야 함.

입력예

```
./phone  홍길동  신수동
```

출력예

```
------->
name: 홍길동
address: 서울시 마포구 신수동 서강대학교 AS관 301호
phone: 02-705-2665
<-------
```

1. 온라인 전화/주소록 - 검사

./phone

./phone 경기도

./phone Andrew 호원동

```
[redacted] (0.157s)
./phone

Usage: phone searchfor [·searchfor]
(You didn't tell me what you want to search for.)
```

```
[redacted]
./phone 경기도

---------------->
name : Andrew
address : 경기도 의정부시 호원동 23-12번지
phone : 031-827-7842
<----------------
---------------->
name : 한승현
address : 경기도 고양시 일산동구 장항2동 호수마을 407호 1602호
phone : 031-904-5643
<----------------
---------------->
name : 신봉선
address : 경기도 고양시 일산동구 마두동 206동
phone : 031-813-3803
<----------------
```

```
[redacted]
./phone Andrew 호원동

---------------->
name : Andrew
address : 경기도 의정부시 호원동 23-12번지
phone : 031-827-7842
<----------------
```

2. Makefile

2주차 실습1

- Makefile 간단히 만들기
- variable을 사용하여 Makefile을 간단히 만들어본다
- phony도 사용해본다

```
animal_exe : dog.o blackcow.o turtle.o main.o
    gcc -o animal.exe dog.o blackcow.o turtle.o main.o

dog.o : dog.c
    gcc -c -o dog.o dog.c

blackcow.o : blackcow.c
    gcc -c -o blackcow.o blackcow.c

turtle.o : turtle.c
    gcc -c -o turtle.o turtle.c

main.o : main.c
    gcc -c -o main.o main.c

clean :
    rm *.o animal.exe
```

2. makefile

Use Makefile

.PHONY : clean

...

Phony target

실제 파일의 이름이 아니라, 단순히 recipe를 대표하는 이름이라고 생각하면 된다.

여기서의 recipe: rm $(target) $(objects)

현재 디렉토리에 있는 target 파일과 모든 object 파일들을 지운다.

```
1  cc = gcc
2  target = movie
3  objects = main.o film.o director.o
4
5  $(target): $(objects)
6  →   $(cc) -o $(target) $(objects)
7
8  $(objects) : movie.h
9
10 .PHONY : clean
11 clean :
12 →   rm $(target) $(objects)
13
```

– PHONY를 사용하는 이유

예상치 못한 상황을 방지하기 위하여..

Ex) 파일 중에 clean이라는 이름을 가진 것이 있다면..?

2번 검사법

```
make clean
rm animal_exe dog.o blackcow.o turtle.o main.o
```

```
make
cc    -c -o dog.o dog.c
cc    -c -o blackcow.o blackcow.c
cc    -c -o turtle.o turtle.c
cc    -c -o main.o main.c
gcc -o animal_exe dog.o blackcow.o turtle.o main.o
```

```
./animal_exe
dog
blackcow
turtle
```

3. GDB

2주차 실습2

- gdb를 이용하여 코드의 문제를 찾고 이유를 설명해본다.
- 코드)

```c
#include <stdio.h>

main(void)
{
    int i;
    double num;

    for (i=0; i<5; i++){
        num=i/2 + i;
        printf("num is %f \n", num);
    }
}
```

- breakpoint를 잡고 run해본다. breakpoint: b 포인트 잡을 줄 번호, run: r
- 한줄 한줄 실행해본다. 한 줄 실행: s
- num값을 확인해본다. 변수에 들어있는 값을 확인하려면: display 변수명, num값 확인: p 변수명

3. GDB

- **`i * 1.5`를 출력하려 했는데 문제가 생겼다.**
- `gcc test.c -o test -g` → -g flag 있어야 gdb로 확인 가능

```c
#include <stdio.h>

main(void)
{
    int i;
    double num;

    for (i=0; i<5; i++){
        num=i/2 + i;
        printf("num is %f \n", num);
    }
}
```

3. GDB

- 컴파일해서 나온 **실행 파일**을 gdb로 실행하면 됩니다.
- 나가는 법: [q] + [Enter]
- gdb 사용시 나오는 ‘$2 = 3’은
  - gdb에 기록된 2번째 값이 3
  - 단순히 GDB가 내부적으로 결과를 추적하기 위해 붙이는 일련번호

3. GDB 채점 (1)

1. 9번 라인을 breakpoint
2. 뒷장 확인

```
 1 #include <stdio.h>
 2
 3 main(void)
 4 {
 5     int i;
 6     double num;
 7
 8     for (i=0; i<5; i++){
 9         num=i/2 + i;
10         printf("num is %f \n", num);
11     }
12 }
```

2. GDB 채점 (2)

- 같은 색은 같은 명령어로 시작
- **요건**
  1. breakpoint가 잘 잡혀있음
  2. i와 num의 값이 보임
  3. 디버깅 현재 위치가 다음으로 움직여짐

정말정말 도움이 필요하면 손 들어주시고 🙋
실습 완료하면 손 흔들어주세요 👋

감사합니다! 🙇

실습 확인: ppt 참조

1번, 2번: 압축 파일(.zip), 3번: 스크린샷(사진파일)

1번, 2번: `4_class_[student id]_[문제번호].zip`안에 실행할 때 사용되는 **모든 파일**
e.g. `4_class_20251234_1.zip` -> mydata, display.awk, phone 같은 파일들
3번: `4_class_[student id]_2.png/jpeg/...` -> `4_class_20251234_2.png`