일일 업무 사항 정리

작성자	제품팀 이민성 인턴
업무 일시	20221128~20221202

세부 사항

1. 업무 내역 요약 정리

목표 내역	Done & Plan
	10주차 **- linux c programming 할때 자주 사용되는 tools** **1. vi** **2. gcc** **3. make** **4. gdb**

2. 내용 세부 (업무 세부 내역 정리 및 기타 사항 정리)

linux c programming 할 때 자주 사용되는 tools

1. vi

- 명령모드: vi 명령어를 통해 vi를 시작할 경우 실행되는 모드. 방향키를 통해 커서를 이동할 수 있다.

- 입력모드: 명령 모드에서 i 또는 a 키를 눌러 입력 모드로 넘어갈 수 있다. 입력 모드에서는 자유롭게 코드나 글을 작성할 수 있으며, 명령 모드로 돌아갈 때에는 ESC를 누르면 된다.

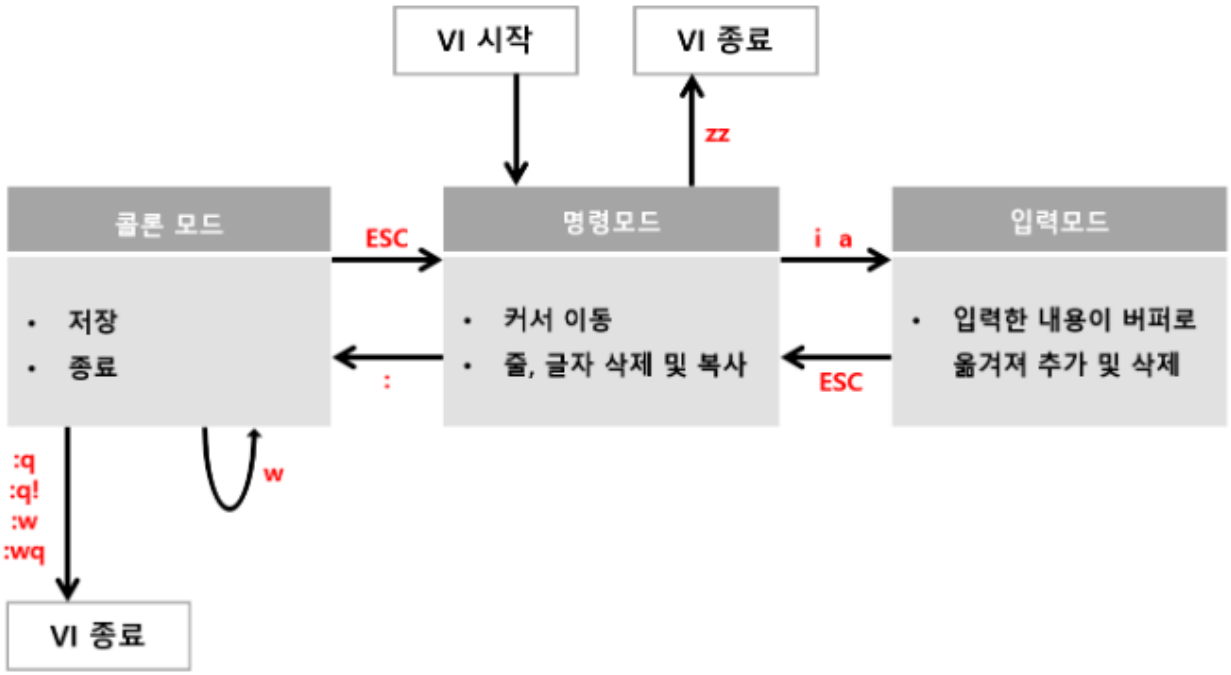
i : 커서가 현재 위치한 부분에서부터 시작

a : 커서 바로 다음 부분부터 시작

shift+spacebar : 영/한 변환

- 콜론모드: 명령 모드에서 : (콜론)을 입력하면 화면 맨 아랫줄에 입력 가능한 공간이 출력된다. 여기서 vi를 종료할 수 있다.

- vi 구성



- vi 명령어

명령 모드에서 입력 모드로 전환

i: 커서 앞(왼쪽)에 입력

a: 커서 다음(오른쪽)에 입력

입력 모드에서 명령 모드로 전환

ESC

저장, 종료하기

명령 모드에서 콜론:을 이용하여 다양한 작업이 가능하다. (콜론 모드)

:q 종료

:q! 저장하지 않고 강제로 종료

:w 저장

:wq 저장 후 종료

ZZ 저장 후 종료 (:wq와 동일)

:wq 파일 이름: 저장 후 파일 이름 지정

** vi 에디터에서는 비정상 종료 시 .swp 파일이 생성된다. 필요하지 않은 경우 삭제한다.

커서 이동

h, j, k, l: 좌,하,상,우 커서 이동 (방향 키가 없는 키보드에서 사용)

w: 다음 단어의 첫 글자로 이동

b: 이전 단어의 첫 글자로 이동

G: 마지막 행으로 가기

:숫자: 지정한 숫자 행으로 이동 ex) :5

삭제

x: 커서에 있는 글자 삭제

X : 커서 앞에 있는 글자 삭제

dw : 커서를 기준으로 뒤에 있는 단어 글자 삭제 (커서 포함)

db : 커서를 기준으로 앞에 있는 단어 글자 삭제

dd : 커서가 있는 라인(줄) 삭제

** dw, db, dd 명령 앞에 삭제할 숫자를 지정 가능 ex) 3dw, 2db, 4dd

** 삭제 된 내용은 버퍼에 저장되어 붙여넣기가 가능

복사

yw : 커서를 기준으로 뒤에 있는 단어 글자 복사 (커서 포함)

yb : 커서를 기준으로 앞에 있는 단어 글자 복사

yy : 커서가 있는 라인(줄) 복사

** yw, yb, yy 명령 앞에 복사할 숫자를 지정 가능 ex) 3yw, 2yb, 4yy

붙여넣기 (복사, 삭제된 내용을 붙여넣는다.)

p : 커서 다음에 붙여넣기

P : 커서 이전에 붙여넣기

찾기

/문자열 : 앞에서 부터 문자열을 찾는다.

?문자열 : 뒤에서 부터 문자열을 찾는다.

n : 뒤로 검색

N : 앞으로 검색

바꾸기

:%s/old/new: 각 행의 처음 나오는 old를 찾아 new로 바꾼다.

:%s/old/new/g: 모든 old를 찾아 new로 바꾼다.

:%s/old/new/gc: 모든 old를 찾아 new로 바꾸기 전에 물어본다

되돌리기(Undo), 다시실행(Redo)

u : 이전으로 되돌리기 (Undo)

Ctrl + r : 되돌리기한 것을 다시 실행 (Redo)

자주 사용하는 기능들

:set number: 행번호를 출력 (간단하게 :set nu)

:set nonumber: 행번호를 숨긴다. (간단하게 :set nonu)

:cd: 현재 디렉토리를 출력

2. GCC

- GCC란?

GCC는 GNU 컴파일러 모음 (GNU Compiler Collection)의 약자이다. GNU 프로젝트의 일환으로 개발되어 널리 쓰이고 있는 컴파일러이다.

- GNU란?

GNU는 GNU's not UNIX의 재귀약자로, 리처드 스톨먼이 각종 자유 소프트웨어들이 돌아가고 번영할 수 있는 기반 생태계를 구축하기 위해 시작한 프로젝트이다.

- 컴파일러란?

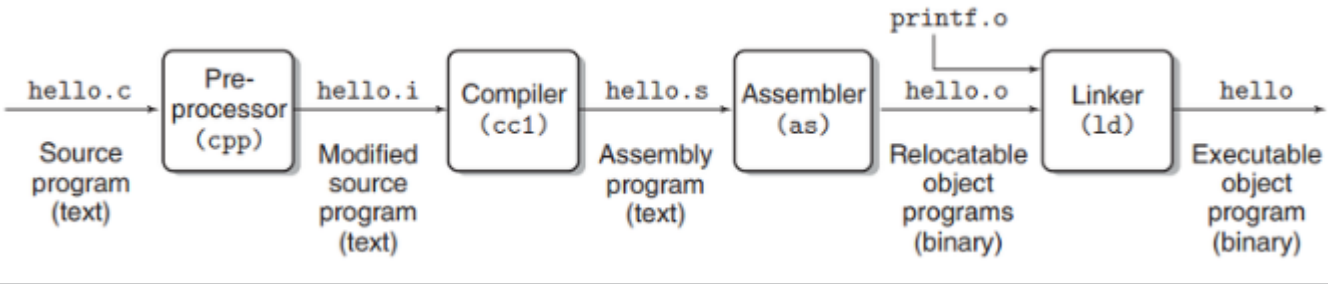
컴파일(Compile)은 어떤 언어의 코드를 다른 언어로 바꿔주는 과정을 말한다. 예를 들어, 사람이 인식하고 이해할 수 있는 C언어 코드를 컴퓨터가 이해할 수 있는 기계어로 바꿔주는 것이다.

즉, 컴파일러(Compiler)는 어떤 프로그래밍 언어로 쓰여진 소스 파일을 다른 언어로 바꾸어 주는 번역기이다.

- 소스 코드가 실행 파일이 되는 과정

소스 코드를 실행 파일로 만들기 위해 전처리, 컴파일, 어셈블, 링크 단계를 거친다.

gcc hello.c 명령어를 입력하면 네 가지 단계를 거쳐서 실행 파일이 생성되지만, 각 단계의 파일들은 임시 파일로 생성되었다가 사라진다.



(1) 전처리 단계

전처리기가 소스 파일 내의 전처리기 지시자를 처리한다.

전처리기 지시자란?

#으로 시작하고 세미콜론 없이 개행문자로 종료되는 라인을 의미한다.

* #include: 지정된 특정 파일의 내용을 해당 지시자가 있는 위치에 삽입

* #define: 매크로 함수 및 상수 정의에 사용한다. 코드 내의 해당 상수를 프로그래머가 정의한 문자열로 대체한다.

전처리 단계를 거치면 소스파일 hello.c에서 확장 소스 파일인 hello.i가 생성된다.

```
# 전처리 과정 실행
gcc -E main.c -o main.i
```

(2) 컴파일 단계

전처리된 파일인 hello.i로부터 어셈블리어로 된 파일인 hello.s 파일을 생성한다.

```
# 컴파일 과정 실행 [ *.c -> (*.i) -> *.s ]
gcc -S main.c
```

어셈블리어란?

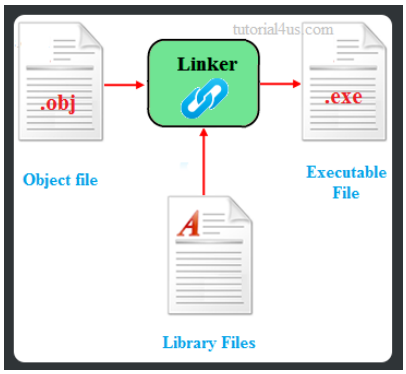
기계어보다 한 단계 위에 있는 언어이며, 기계어와 함께 단 두 가지 뿐인 저급 언어에 속한다. 기계어는 컴퓨터 관점에서 바로 읽을 수 있지만, 인간이 사용하기 불편한 언어이기 때문에 이를 보완하기 위해 등장한 것이 어셈블리어이다.

(3) 어셈블 단계

어셈블리어 파일 hello.s를 기계어로 된 오브젝트 파일 hello.o 파일로 변환한다. 즉, 컴퓨터가 읽을 수 있는 0 과 1 로 이루어진 2 진수 코드로 변환하는 것이다.

```
# 어셈블 과정 실행 [ *.c -> (*.i) -> (*.s) -> *.o ]
gcc -c main.c
```

(4) 링크 단계



링크 단계는 작성한 프로그램이 사용하는 다른 프로그램이나 라이브러리를 가져와서 연결하는 과정이다. 그 결과로 실행 가능한 파일을 생성한다. (hello.o → hello)

```
# 링크 단계 [ *.c -> (*.i) -> (*.s) -> (*.o) -> excutable ]
gcc -o main main.c func.c
```

- GCC 컴파일 옵션

-o [파일명] [*.c] : 지정한 파일명으로 실행 파일을 저장한다.

ex) gcc -o result.out main.c

-E : 전처리 단계를 수행한 후, 컴파일 과정을 거치지 않는다.
실행 결과는 standard output에 출력된다.

-S : 컴파일 단계를 수행한 후, 어셈블 과정을 거치지 않는다.
실행 결과로 어셈블리어로 변환된 *.s 파일이 생성된다.

-c [파일명] [*.c] : 소스 코드를 컴파일 또는 어셈블하며, 링크를 하지 않는다.
파일명으로 오브젝트 파일을 생성한다.
ex) gcc -c ft_isalnum.c

-I [디렉토리명] : 디렉토리명에서 헤더 파일을 검색한다.

-l [라이브러리] : 라이브러리 파일과 링크한다. 접미사나 확장자(.a/.o)가 없어도 링크한다.
ex) 라이브러리 파일이 libmath.a 일때 다음과 같이 작성
gcc myfile.c -lmath -o myfile

-L [디렉토리명] : 디렉토리 내에서 라이브러리 파일을 찾는다.

-D [매크로상수명]=[값] : 매크로 상수를 정의하기 위한 옵션이다.

ex) gcc -D BUFFER_SIZE=42 : BUFFER_SIZE 라는 매크로 상수의 값을 42 로 설정한다.

-c : 옵션은 링크를 하지 않고 컴파일만 진행한다. 이 옵션을 생략하면 main 함수를 찾을 수 없다는 오류가 출력된다.

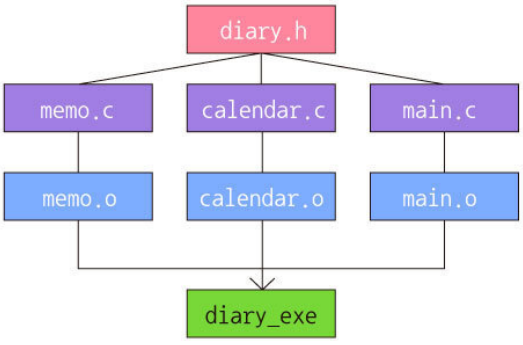
3. make

- make란?

make는 파일 관리 유틸리티로 파일 간의 종속관계를 파악하여 Makefile(기술파일)에 적힌 대로 컴파일러에 명령하여 SHELL 명령이 순차적으로 실행될 수 있게 합니다.

- make를 쓰는 이유

1. 각 파일에 대한 반복적 명령의 자동화로 인한 시간 절약
2. 프로그램의 종속 구조를 빠르게 파악할 수 있으며 관리가 용이
3. 단순 반복 작업 및 재작성을 최소화



위의 종속관계 표를 보며 diary_exe라는 실행 파일을 만들어 보자.

1. diary.h 헤더파일 만들기

세 개의 c파일이 include 할 헤더파일을 생성해 보자

```
//diary.h

#include <stdio.h>
void memo();
void calendar();
```

2. 재료로 사용될 c파일 만들기

```
//memo.c

#include "diary.h"
void memo(){
        printf("I'm function Memo! \n");
}
```

```
//calendar.c

#include "diary.h"
void calendar(){
        printf("I'm function Calendar() \n");
}
```

```
//main.c

#include "diary.h"

int main(void){
        memo();
        calendar();
        return 0;
}
```

3. 생성된 파일 확인하기

```
[lms@localhost c]$ ls
calendar.c  diary.h  main.c  memo.c
```

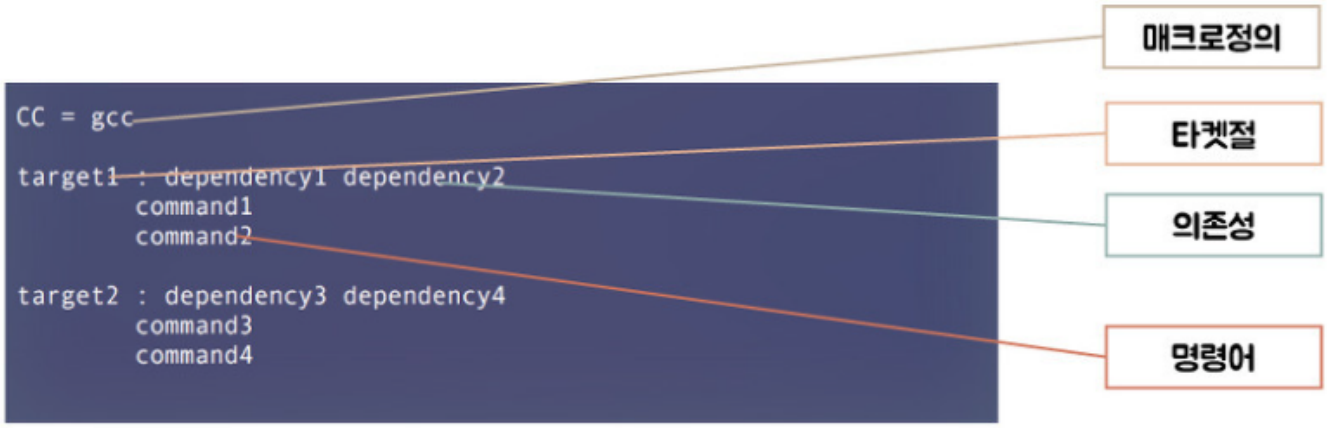
- make를 이용한 컴파일 과정

- makefile의 구성

makefile은 다음과 같은 구조를 가진다.

* 목적파일(Target) : 명령어가 수행되어 나온 결과를 저장할 파일
* 의존성(Dependency) : 목적파일을 만들기 위해 필요한 재료
* 명령어(Command) : 실행 되어야 할 명령어들
* 매크로(macro) : 코드를 단순화 시키기 위한 방법

- makefile의 기본구조



- makefile 작성규칙

목표파일 : 목표파일을 만드는데 필요한 구성요소들

(tab)목표를 달성하기 위한 명령 1

(tab)목표를 달성하기 위한 명령 2

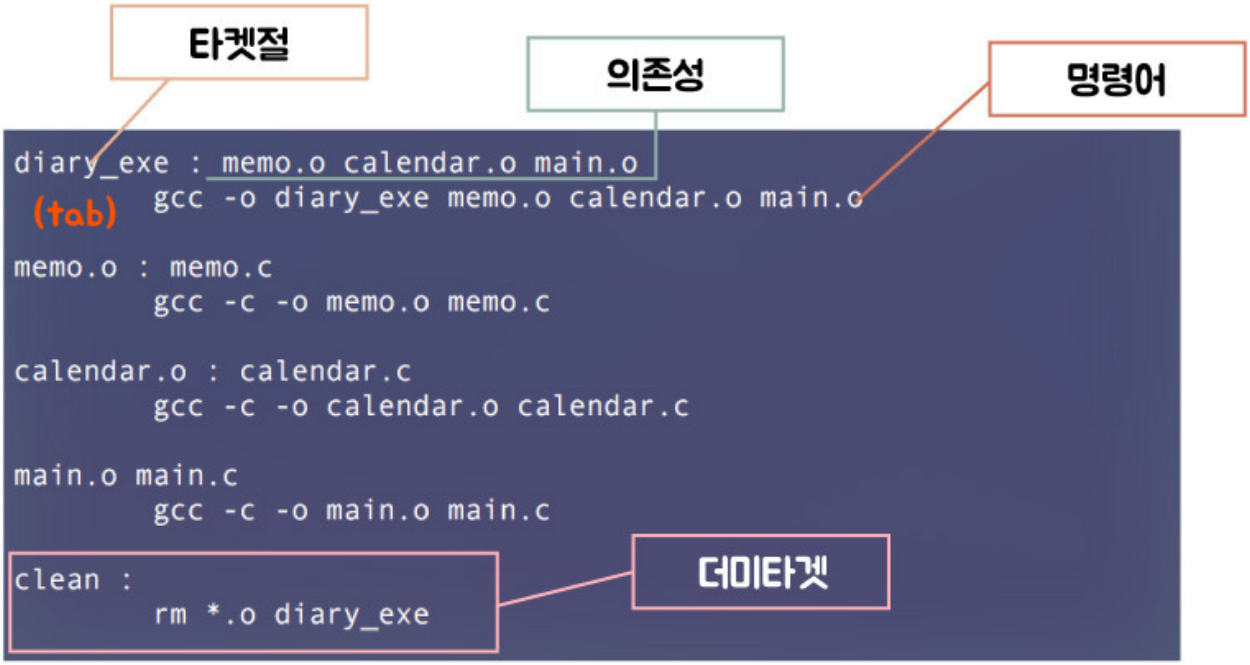
// 매크로 정의 : Makefile에 정의한 string 으로 치환한다.

// 명령어의 시작은 반드시 탭으로 시작한다.

// Dependency가없는 target도 사용 가능하다.

- make 예제 따라해보기

vi makefile



여기서 더미타겟은 파일을 생성하지 않는 개념적인 타겟으로 make clean 라 명령하면 현재 디렉토리의 모든 object 파일들과 생성된 실행파일인 diary_exe를 rm 명령어로 제거해 준다.

이제 make로 makefile을 실행해준다.

```
ubuntu@ip-172-31-10-167:~/fsLab/exMake$ ls
calendar.c  diary.h  main.c  Makefile  memo.c
ubuntu@ip-172-31-10-167:~/fsLab/exMake$ make
gcc    -c -o memo.o memo.c
gcc    -c -o main.o main.c
gcc    -c -o calendar.o calendar.c
gcc  -o diary_exe memo.o main.o calendar.o
ubuntu@ip-172-31-10-167:~/fsLab/exMake$ ls
calendar.c  diary_exe  main.c  Makefile  memo.o
calendar.o  diary.h    main.o  memo.c
ubuntu@ip-172-31-10-167:~/fsLab/exMake$ ./diary_exe
Function Memo .
Function Calendar()
ubuntu@ip-172-31-10-167:~/fsLab/exMake$
```

명령어들이 실행되면서 타겟파일이였던 diary_exe가 만들어졌다. 실행결과는 기본적인 컴파일 과정에서 본 결과와 동일함을 알 수 있다.

4. gdb

- 디버깅: 버그를 제거하는 과정

- gdb란?

보통은 GDB라고 부르는 GNU 디버거(GNU Debugger)는 GNU 소프트웨어 시스템을 위한 기본 디버거이다. GDB는 다양한 유닉스 기반의 시스템에서 동작하는 이식성있는 디버거로, 에이다, C, C++, 포트란 등의 여러 프로그래밍 언어를 지원한다.

- 명령어 사용 예

```
gdb prog.out      prog.out 를 디버깅..
gdb > run         프로그램 실행
```