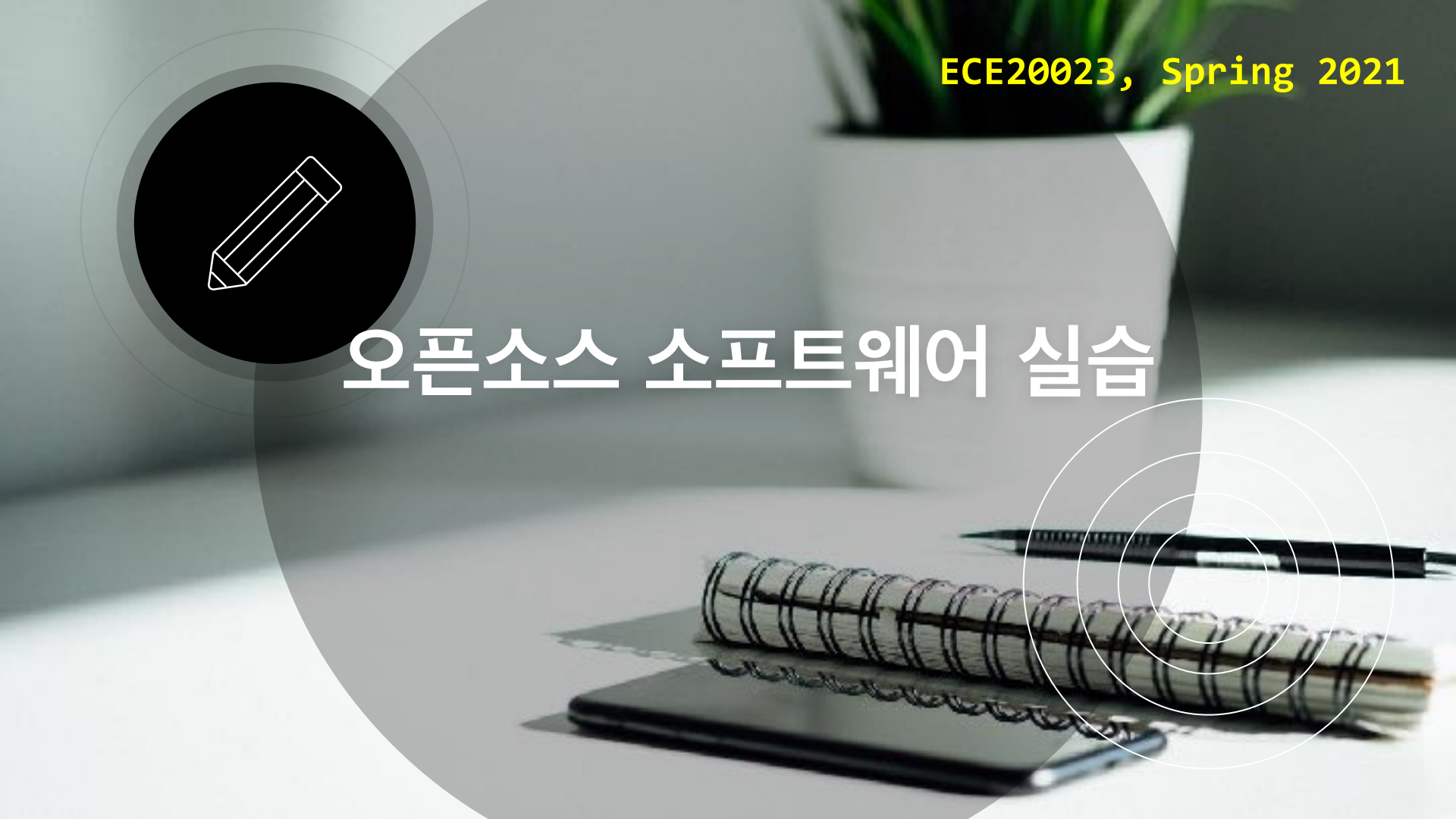
ECE20023, Spring 2021



오픈소스 소프트웨어 실습

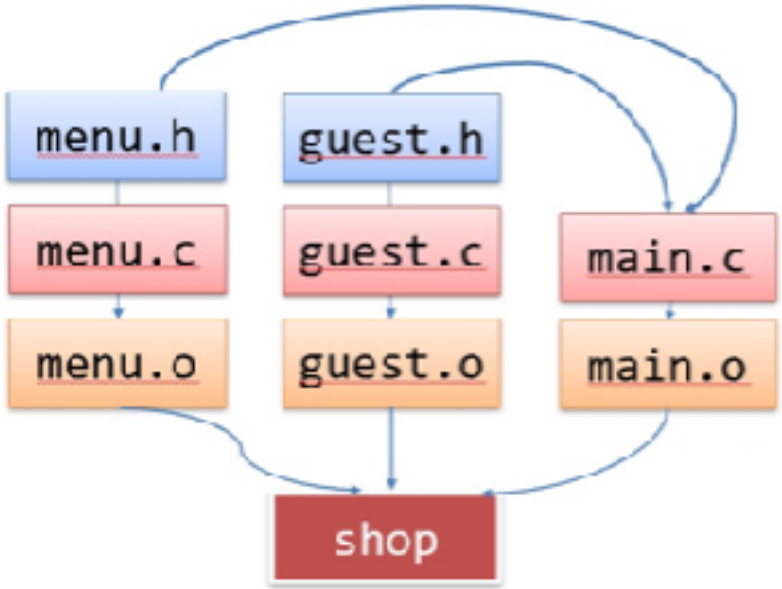
9

make Utility

make utility

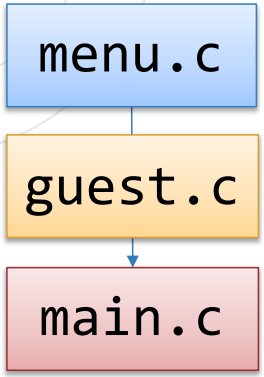
- 소프트웨어 개발을 위해 유닉스 계열 운영 체제에서 주로 사용 되는 프로그램 빌드 도구
- 여러 파일들의 의존성과 필요한 명령을 정의할 수 있음
- 컴파일 & 실행파일을 생성할 수 있는 표준문법이 있음
- makefile을 실행함
- 장점
  - 파일에 대한 반복적 명령 자동화로 시간 절약
  - 프로그램의 종속 구조를 빠르게 파악하고 관리가 용이
  - 단순 반복 작업 및 재작성을 최소화

Pizza Ordering System




```
*****************
1. Pizza   : 20000
2. Chicken : 12000
*****************
원 하 는  메 뉴 ?  1
Pizza 선 택
```

Pizza Ordering System



- Compiling libraries

```
walab-HGU:~/2021OSS/lab5:> gcc -c menu.c -o menu.o
walab-HGU:~/2021OSS/lab5:> gcc -c guest.c -o guest.o
```

- Build(main)

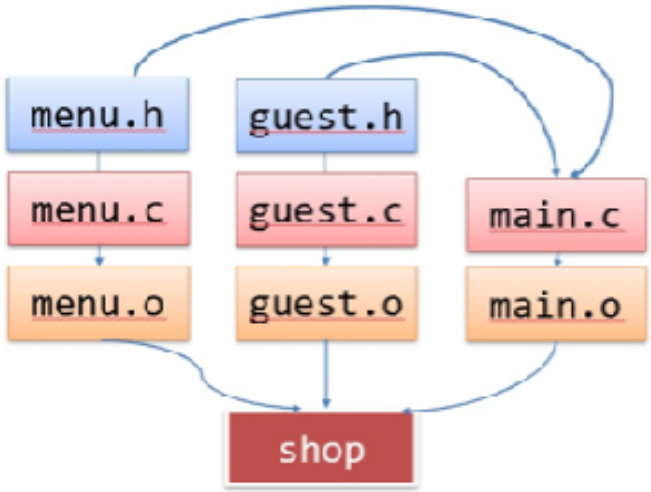
```
walab-HGU:~/2021OSS/lab5:> gcc -o pizzashop main.c menu.o guest.o
```

- Run

```
walab-HGU:~/2021OSS/lab5:> ./shop
```

Pizza Ordering System

- Create header files : menu.h, guest.h
- Create C files : menu.c, guest.c, main.c
- Compile & build
- Run



Pizza Ordering System

- Create header files : menu.h, guest.h

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

void displayMenu();
```

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

int addGuest();
void displayGuest(int menu);
```

Pizza Ordering System

- Create C files : menu.c, guest.c, main.c

```
**********************
1. Pizza : 20000
2. Spaghetti: 12000
**********************
원하는 메뉴는? 1
Pizza 선택
```

menu,c

```c
//menu.c
#include "menu.h"

void displayMenu(){
        printf("**********************\n");
        printf("1. Pizza : 20000 \n");
        printf("2. Spaghetti: 12000 \n");
        printf("**********************\n");
}
```

guest,c

```c
//guest.c
#include "guest.h"

int addGuest(){
        int menu;
        printf("원하는 메뉴는? ");
        scanf("%d", &menu);
        return menu;
}

void displayGuest(int menu){
        if(menu == 1)
                printf("Pizza 선택");
        else
                printf("Spaghetti 선택");
        printf("\n");
}
```

main,c

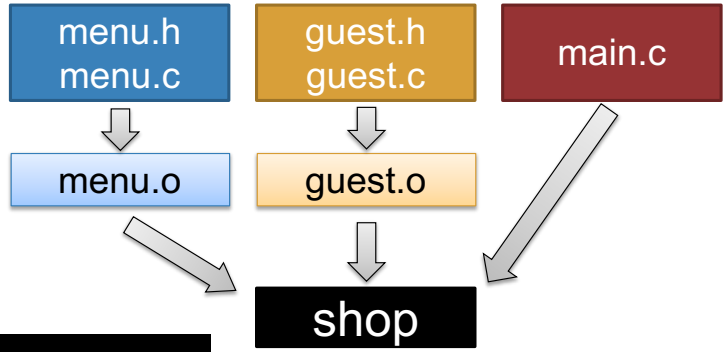
```c
#include "menu.h"
#include "guest.h"

int main(){
        int menu;

        displayMenu();
        menu = addGuest();
        displayGuest(menu);
        return 0;
}
```

Pizza Ordering System

- Compile & build
  - menu.c / guest.c
  - main.c




```
walab-HGU:~/lab5:> ls
guest.c  guest.h  main.c  menu.c  menu.h
walab-HGU:~/lab5:> gcc -c menu.c
walab-HGU:~/lab5:> gcc -c guest.c -o guest.o
walab-HGU:~/lab5:> gcc main.c guest.o menu.o -o shop
walab-HGU:~/lab5:> ls
guest.c  guest.h  guest.o  main.c  menu.c  menu.h  menu.o  shop
walab-HGU:~/lab5:> ./shop
```

Makefile

- make utility를 실행하면 Makefile을 읽어들임
- 구조
  - Target : 명령어가 수행된 결과 파일명
    - target / dummy target(결과 파일 없음)
  - Dependency : 대상 파일 생성을 위해 필요한 파일
    - 파일 수정 날짜 체크
  - Command : 실행할 명령 문장
  - Macro : 코드 단순화

```
target1 : dependency1 dependency2 ...
<tab>command1
<tab>command2
```

Makefile 실습

```
target1 : dependency1 dependency2 ...
<tab>command1
<tab>command2
```

```
shop : main.c menu.o guest.o
	gcc -o shop main.c menu.o guest.o
menu.o: menu.c menu.h
	gcc -c menu.c -o menu.o
guest.o: guest.c guest.h
	gcc -c guest.c
clean:
	rm *.o shop
```

Makefile

```
walab-HGU:~/lab5:> make clean
rm *.o shop
walab-HGU:~/lab5:> make
gcc -c menu.c -o menu.o
gcc -c guest.c
gcc -o shop main.c menu.o guest.o
walab-HGU:~/lab5:> make clean
rm *.o shop
walab-HGU:~/lab5:> make shop
gcc -c menu.c -o menu.o
gcc -c guest.c
gcc -o shop main.c menu.o guest.o
walab-HGU:~/lab5:> ./shop
```

Macro

- Makefile을 작성할 때 자주 사용하며 문자열을 간단하게 함
- 변수에 특정 문자열을 정의하고 표현하는 방식
- Pre-defined macro

```
walab-HGU:~:> make -p | grep cc
make: *** No targets specified and no makefile found.  Stop.
COMPILE.cpp = $(COMPILE.cc)
CC = cc
LINK.cc = $(CXX) $(CXXFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH)
YACC = yacc
LINK.C = $(LINK.cc)
LINK.cpp = $(LINK.cc)
COMPILE.cc = $(CXX) $(CXXFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c
COMPILE.C = $(COMPILE.cc)
```

Macro 작성방법

- NAME = 문자열 (관습적으로 NAME은 대문자 사용)
- # 주석문
- \ 여러 행에 걸쳐서 문자열 사용할 때
- $(NAME) 매크로를 참조
- 정의되지 않은 매크로를 사용하는 경우 NULL로 치환됨
- 문자열의 따옴표도 문자열로 인식함

Macro 예제

```
shop : main.c menu.o guest.o
	gcc -o shop main.c menu.o guest.o
menu.o: menu.c menu.h
	gcc -c menu.c -o menu.o
guest.o: guest.c guest.h
	gcc -c guest.c
clean:
	rm *.o shop
```

Makefile

macro를 적용한 Makefile

```
CC = gcc
shop : main.c menu.o guest.o
	$(CC) -o shop main.c menu.o guest.o
menu.o: menu.c menu.h
	$(CC) -c menu.c -o menu.o
guest.o: guest.c guest.h
	$(CC) -c guest.c
clean:
	rm *.o shop
```

Suffix rule

- 파일 확장자를 보고 그에 따라 적절한 명령을 실행함
  - C 소스 파일(*.c), 목적파일(*.o)을 인식
- 목적파일이 없는 경우 같은 이름의 C 소스 파일을 컴파일하여 생성

확장자 규칙을 적용한 Makefile

```
CC = gcc
shop : main.c menu.o guest.o
	$(CC) -o shop main.c menu.o guest.o
clean:
	rm *.o shop
```

Special Macros

- $@ 현재 Target 이름
- $* (확장자가 있는 경우에) 확장자가 없는 현재 Target
- $< 첫번째 dependency 이름
- $^ Dependency 전체
- $? Dependency 중 현재 target보다 최근에 갱신된 파일

```
CC = gcc
shop : main.c menu.o guest.o
	$(CC) -o shop main.c menu.o guest.o
clean:
	rm *.o shop
```

Special macro를 적용한 Makefile

```
CC = gcc
shop : main.c menu.o guest.o
	$(CC) -o $@ $^
clean:
	rm *.o shop
```

Makefile

```
CC = gcc
shop : main.c menu.o guest.o
	$(CC) -o $@ $^
clean:
	rm *.o shop
```

```
CC = gcc
CFLAGS = -W -Wall
TARGET = shop
OBJECTS = main.o menu.o guest.o
all : $(TARGET)
$(TARGET) : $(OBJECTS)
	$(CC) $(CFLAGS) -o $@ $^
clean:
	rm *.o shop
```