

컴퓨터 프로그래밍 및 실습

15주차. 동적 메모리 & 전처리 및 다중소스 파일

실습 안내

■ 실습 제출 안내

- 솔루션 이름은 "Practice week 15"
- 프로젝트 이름과 소스코드 이름은 Problem1, Problem2, ...
 - 실습1의 프로젝트 이름은 Problem1, 소스코드 이름은 problem1.c
 - 실습 2의 프로젝트 이름은 Problem2, 소스코드 이름은 problem2.c ...
- 솔루션 폴더를 압축하여 **Practice_week15_학번_이름.zip** 으로 제출
- 제출기한: 내일 **24시** 까지

1. 동적 메모리

■ 동적 메모리 할당

■ 정적 메모리

- 배열에 데이터가 얼마나 담길지 모르니까 미리 생성해 놓자!
- `int a[100] = { 0 };`
- 컴파일 타임에서 메모리 공간을 만들어 놓기 때문에 입력이 100보다 크면 실행 불가
- 입력이 100보다 적게 들어온다면 메모리 낭비

■ 동적 메모리

- 실행 시간(런타임)에 메모리를 할당 받을 수 있다.
- 메모리 공간을 미리 만들어 놓지 않고, 메모리 필요양에 따라서 유동적으로 공간 할당 가능
- `int* a = (int*) malloc(100 * sizeof(int))`

1. 동적 메모리

❖ 동적 메모리 할당

Syntax: 동적메모리할당

예 `int *p;`
`p = (int *)malloc(100*sizeof(int));` // 100개의 정수 할당

동적 메모리의 주소

필요한 바이트 수

```
int *score;  
score = (int *)malloc(100*sizeof(int));  
  
if( score == NULL ){  
    ... // 오류 처리  
}
```

1. 동적 메모리

■ 동적 메모리 할당

■ malloc(Memory ALLOcation)

Syntax: 동적메모리할당

```
예 int *p;  
p = (int *)malloc(100*sizeof(int)); // 100개의 정수 할당
```


동적 메모리의 주소 필요한 바이트 수

- [타입]* [변수명] = ([타입] *) malloc([바이트]);
- malloc 호출 시, 필요한 바이트만큼의 빈 공간의 메모리 시작주소가 반환됨

Malloc()

```
int* ptr = ( int* ) malloc ( 5* sizeof ( int ));
```

4 bytes

ptr = 

← 20 bytes of memory →

A large 20 bytes memory block is dynamically allocated to ptr



1. 동적 메모리

■ 동적 메모리 할당

■ malloc(Memory ALLOcation)

■ 길이 5의 double 배열이 필요하다면?

- (정적 메모리) `double arr[5];`
- (동적 메모리) `double* arr = (double*) malloc(5*sizeof(double));`

■ 길이 10의 구조체 Student 배열이 필요하다면?

- `struct student { char name[20], ... }`
- (정적 메모리) `struct student arr[10];`
- (동적 메모리) `struct student* arr = (struct student*) malloc(10*sizeof(struct student));`

1. 동적 메모리

■ 동적 메모리 할당

- 할당 받은 메모리의 사용이 끝난 경우
 - 메모리를 다 썼으면 반납해야 한다.
 - 그렇지 않으면 -> 메모리 누수

❖ 동적 메모리 반납

Syntax: 동적메모리해제

예 `score = (int *)malloc(100*sizeof(int));`

...

`free(score);`

score가 가리키는 동적 메모리를 반납한다.

1. 동적 메모리

❖ 예제 2

- 사용자로부터 학생이 몇 명인지 물어보고 적절한 동적 메모리 할당
- 사용자로부터 성적을 받아서 동적 메모리에 저장하였다가 다시 출력

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

int main(void)
{
    int *list;
    int i, students;

    printf("학생의 수: ");
    scanf("%d", &students);

    list = (int *)malloc(students * sizeof(int));

    if (list == NULL) { // 반환값이 NULL인지 검사
        printf("동적 메모리 할당 오류\n");
        exit(1);
    }
}
```


1. 동적 메모리

■ 실습 1

- 여러 사람의 이름을 저장하고 출력하는 프로그램을 개발하고자 한다.
- 이 때, 사람의 이름을 저장하는 배열의 공간을 최대한 작게 쓰려고 한다.
- 아래 예시 참조. 0은 'w0'을 의미

R	i	c	h	0		
D	o	n	a	l	d	0
T	o	m	0			
g	r	e	g	0		

<<일반적인 2차원 배열>>

R	i	c	h	0		
D	o	n	a	l	d	0
T	o	m	0			
g	r	e	g	0		

<<실습1에서 구현할 배열>>

- 첫번째 줄에 저장할 이름의 개수가 입력된다.
- 두번째 줄부터 사람의 이름이 차례대로 입력된다. 이름은 최대 10글자
- 배열에 대해 정적 할당을 절대로 하지 않아야 한다.
- 저장한 이름을 차례대로 출력한다.
- 프로그램의 종료 전에 free를 통해 할당 받은 모든 메모리를 반환한다.
 - 길이 11의 버퍼배열로 이름을 입력받고, 이름의 길이를 토대로 메모리 할당하면 쉽다.

1. 동적 메모리

■ 실습 1

■ free의 예시

- free는 할당 받은 메모리마다 모두 호출해야 정상적으로 메모리가 반환된다.
- 아래 배열은 5번의 동적 할당이 이루어졌다.

R	i	c	h	0		
D	o	n	a	l	d	0
T	o	m	0			
g	r	e	g	0		

<<실습1에서 구현할 배열>>

- char**을 담는 메모리 1개, char*을 담는 메모리 4개
- 위 5개 메모리에 대해서 모두 free를 호출해주어야 함.

1. 동적 메모리

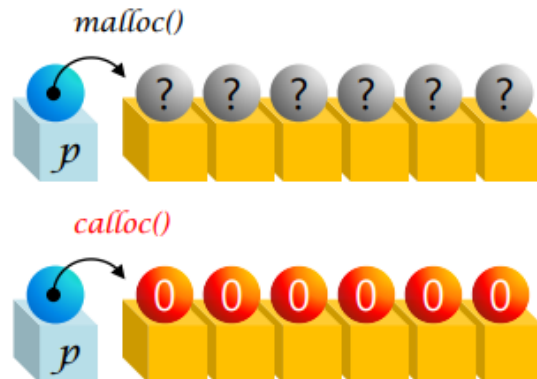
■ Calloc(Contiguous ALLOcation)

❖ calloc()

- calloc()은 0으로 초기화된 메모리 할당
- calloc()은 항목 단위로 메모리를 할당
- (예)

```
int *p;
```

```
p = (int *)calloc(5, sizeof(int));
```



1. 동적 메모리

■ realloc (RE-ALLOcation)

❖ realloc()

- realloc() 함수는 할당하였던 메모리 블록의 크기를 변경
- (예)

```
int *p;  
p = (int *)malloc(5 * sizeof(int));  
p = realloc(p, 7 * sizeof(int));
```



1. 동적 메모리

■ 실습 2

- 자바 언어의 ArrayList라는 배열 역할을 하는 컬렉션은 메모리 공간 절약을 위해 아래와 같은 행동을 한다.
 - 처음 생성시 길이 10의 배열 공간을 할당받는다.
 - 배열이 꽉 찼을 경우 다음 데이터 저장을 위해서 길이를 2배로 늘린다.
 - 만약 또 꽉 찼을 경우, 길이를 다시 2배 늘린다.
 - 길이: 10 -> 20 -> 40 -> ...
- 자바의 ArrayList를 차용하여, c에서 메모리 절약을 위해 int를 담을 수 있는 ArrayList 기능을 개발해보자.
 - 아래 함수를 개발해보자
 - `void add(int num)` -> ArrayList의 배열에 데이터 num을 저장한다.
 - `int get(int index)` -> ArrayList의 index번 인덱스에 저장된 값을 반환한다.
 - `int len()` -> ArrayList에 데이터가 몇 개 담겼는지 반환한다.
- 데이터를 15개 정도 담아보고, 차례대로 출력해보자
- 메모리 할당과 재할당은 각각 `calloc`과 `realloc`을 활용한다.
- 다음장 참조

1. 동적 메모리

■ 실습 2

■ 예시

■ 처음 상태



■ add(3)

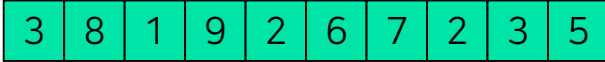


■ add(8)



■ ...

■ add(5)



0 1 2 3 4

■ get(4)

- 2를 반환

■ get(2)

- 1을 반환

2. 전처리기

- 단순 매크로
 - 컴파일 시간에 치환되는 구문

```
#define PI 3.141592           // 원주율
#define EOF      (-1)         // 파일의 끝표시
#define EPS      1.0e-9       // 실수의 계산 한계
#define DIGITS    "0123456789" // 문자 상수 정의
#define BRACKET   "{}[]"      // 문자 상수 정의
#define getchar() getc(stdin) // stdio.h에 정의
#define putchar() putc(stdout) // stdio.h에 정의
```

2. 전처리기

■ 함수 매크로

- 마찬가지로 컴파일 시간에 치환되는 구문
- 단, 매개변수를 가지고 있다.
- 실제 함수를 호출하는 것은 함수 매크로 보다 속도가 느림
- 따라서 간단한 연산은 함수 매크로를 이용하자!

❖ 함수 매크로(function-like macro)란 매크로가 함수처럼 매개 변수를 가지는 것

Syntax: 함수매크로

예 `#define SQUARE(x) ((x) * (x))`

매크로 인수 SQUARE(x)는 이것과 같다.

❖ 함수 매크로 예

```
#define SUM(x, y)      ((x) + (y))
#define AVERAGE(x, y, z) (( (x) + (y) + (z) ) / 3 )
#define MAX(x,y)      ( (x) > (y) ) ? (x) : (y)
#define MIN(x,y)      ( (x) < (y) ) ? (x) : (y)
```


2. 전처리기

■ 내장 매크로

❖ 내장 매크로: 미리 정의된 매크로

내장 매크로	설명
__DATE__	이 매크로를 만나면 현재의 날짜(월 일 년)로 치환된다.
__TIME__	이 매크로를 만나면 현재의 시간(시:분:초)으로 치환된다.
__LINE__	이 매크로를 만나면 소스 파일에서의 현재의 라인 번호로 치환된다.
__FILE__	이 매크로를 만나면 소스 파일 이름으로 치환된다.

```
printf("컴파일 날짜=%s\n", __DATE__);  
printf("치명적 에러 발생 파일 이름=%s 라인 번호= %d\n", __FILE__, __LINE__);
```



```
컴파일 날짜=Aug 23 2021  
치명적 에러 발생 파일 이름=C:\Users\Wkim\source\repos\Project14\Project14\소스.c  
라인 번호= 6
```

2. 전처리기

■ 실습 3

- 퍼센트를 구하는 매크로 PER를 정의해보자
 - `PER(5/20) -> 25`
- 세제곱을 구하는 매크로 QUBE를 정의해보자
 - `QUBE(3) -> 27`
- 문자열을 출력하고 개행하는 매크로 PRINTLN을 정의해보자
 - `PRINTLN("Hello") -> "HelloWn"을 출력`
- 위 세 매크로를 정의하고, 사용 및 출력하는 코드를 작성

3. 다중 소스 파일

■ 소스 파일이 하나였다면?

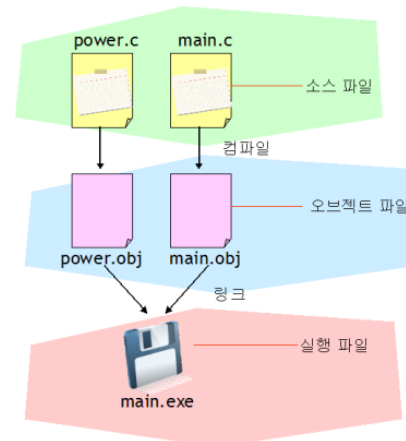
- printf는 stdio.h에 정의되어 있음
- stdio.h도 마찬가지로 소스파일
- 우리는 printf를 쓰기 위해서 매번 100줄이 넘는 printf코드를 우리 소스파일에 복붙해야 한다.

❖ 단일 소스 파일

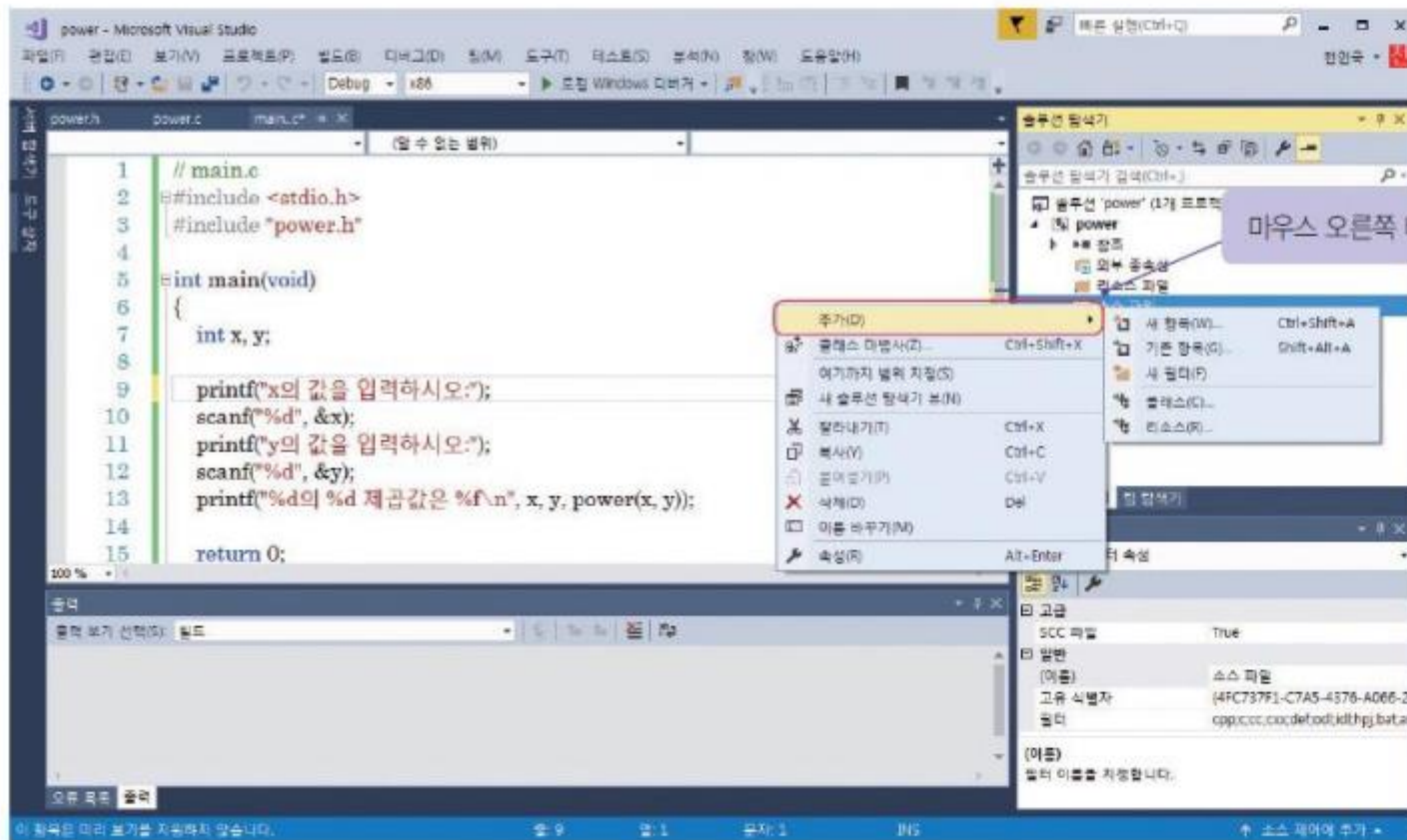
- 파일의 크기가 너무 커진다.
- 소스 파일을 다시 사용하기가 어려움

❖ 다중 소스 파일

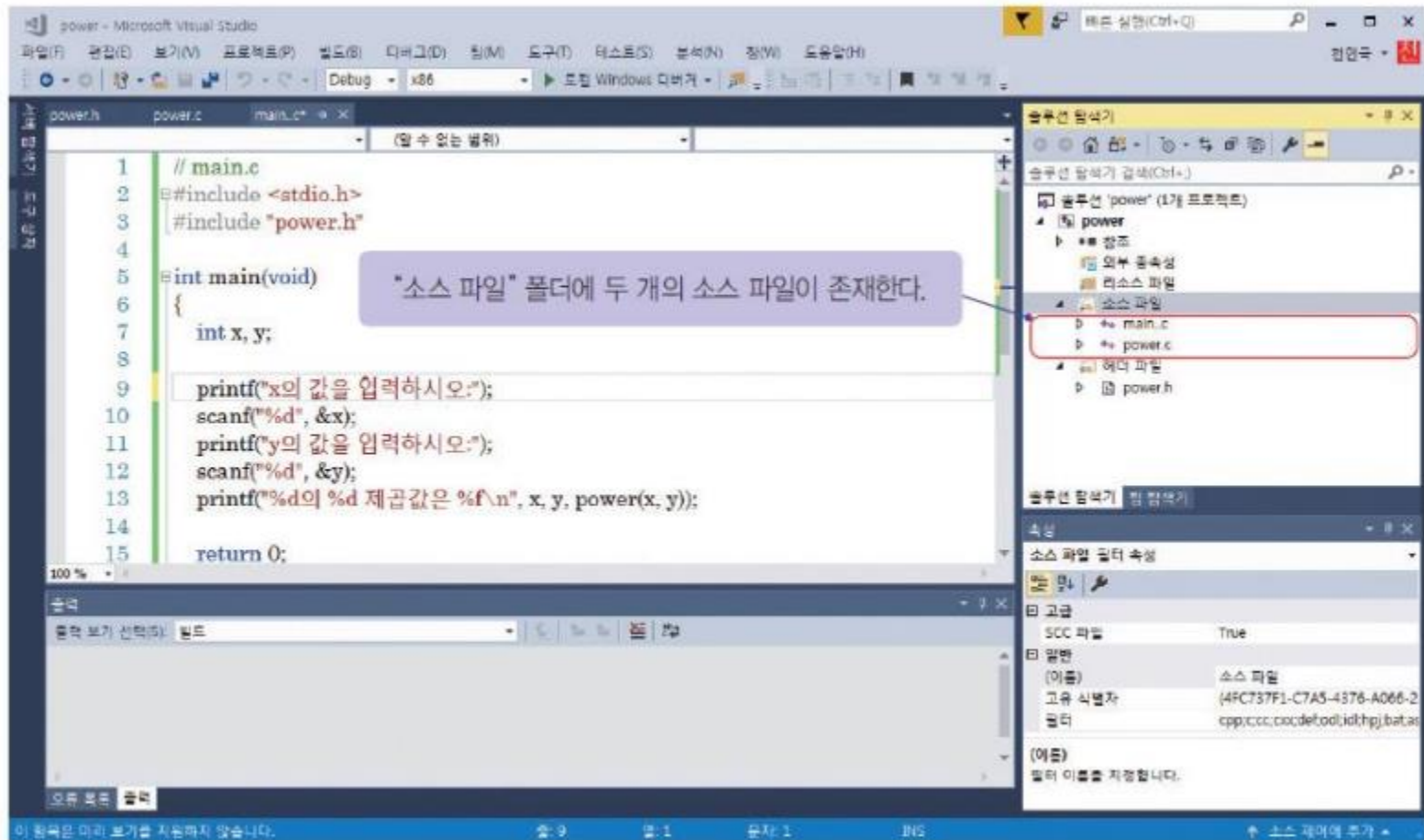
- 서로 관련된 코드만을 모아서 하나의 소스 파일로 할 수 있음
- 소스 파일을 재사용하기가 간편함



3. 다중 소스 파일

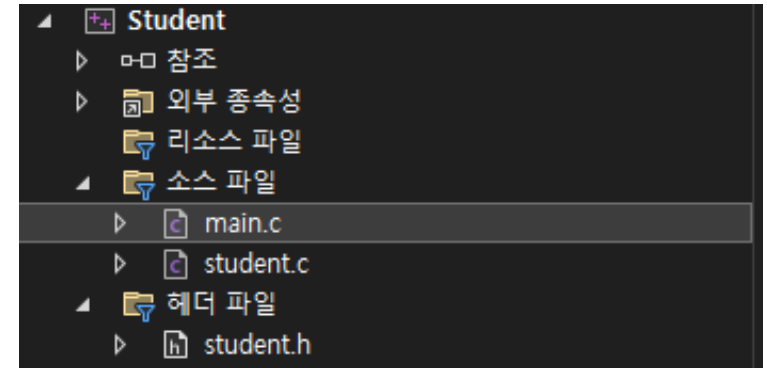


3. 다중 소스 파일



3. 다중 소스 파일

■ 다중 소스 파일의 예시



■ main.c

- <https://github.com/lani009/Ajou-c-programming/blob/main/materials/week%2015/Practice%20Week%2015/Practice%20Week%2015/main.c>

■ student.c

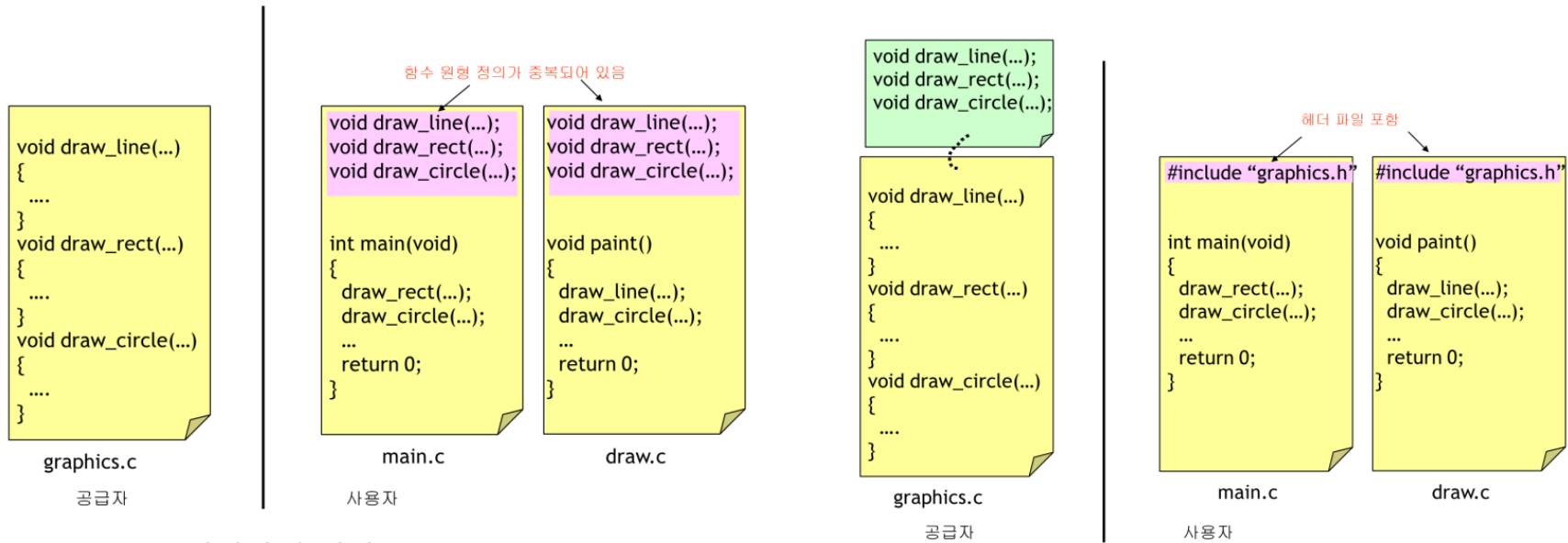
- <https://github.com/lani009/Ajou-c-programming/blob/main/materials/week%2015/Practice%20Week%2015/Practice%20Week%2015/student.c>

■ student.h

- <https://github.com/lani009/Ajou-c-programming/blob/main/materials/week%2015/Practice%20Week%2015/Practice%20Week%2015/student.h>

3. 다중 소스 파일

■ 헤더파일의 사용



<<헤더파일 미사용>>

<<헤더파일 사용>>


3. 다중 소스 파일

■ 헤더파일의 중복 막기

- ❖ 구조체 정의가 들어 있는 헤더 파일을 소스 파일에 2번 포함시키면 컴파일 오류가 발생한다.
- ❖ 이것을 막기 위하여 `#ifndef` 지시어를 사용할 수 있음

```
#ifndef STUDENT_H
#define STUDENT_H

struct STUDENT {
    int number;
    char name[10];
};
#endif
```



소스 파일에서 여러 번
포함시켜도 컴파일 오류가
발생하지 않음

3. 다중 소스 파일

- 실습 4

- 실습 2에서 개발한 ArrayList를 다중 소스 파일로 구현해보자
- add, get, len 세 함수를 ArrayList.h 헤더파일에 담으면 된다.