

C Programming (W9)



Welcome!!

Please check attendance individually.
(Mobile App)

Things to do today

- 01** | Lecture Notes (Ch.8 ~ 9)
- Functions & Variables
- 02** |
- 03** |

Notice

Invite all of students to our much anticipated 2025 Endicott College Job Fair which is scheduled to be held per scheduled below:

Event: 2025 Endicott College Job Fair

When: November 5, 2025 (Wednesday), 1PM - 5PM

Where: Front Yard of the Endicott (W19) Building

Come visit booths to meet and interview with prospective employers who are actively seeking qualified applicants for employment and to take advantage of a program packed full of various free career development services such as, Job & Career consulting, Interview Image Makeup/Snapshot, Special Lectures (Understanding Korean Companies, Business Manners & Smart Newcomer Tips), and much more.

Notice

Place

Booth — Front Yard of the W19 Endicott Building

Special Lecture — Lecture Room 111, W19



Program

Career Connect Booth Zone (12 Hiring Companies)

Job & Career Consulting Booth Zone

Daejeon Work Experience Center/ Visa Consulting Booth

Aroma Strength Booth

Interview Image Makeup Booth

Career Profile Studio Booth

Pause & Pose Zone

Snapshot Booth

Refresh & Gift Lounge

Career Insight Stage (Special Lecture in English)

- Special Lecture 1 – Understanding Korean Companies (12:00~13:00)
- Special Lecture 2 – Business Manners & Smart Newcomer Tips (16:00~17:00)

For more details and inquiries,
or to pre-register,
scan the QR code

Pre-Register Now!

Pre-register through the QR code!

On-site registration available.

Pre-registered participants can skip the interview queue!

Cooking Analogy for Functions, Parameters, and Variables

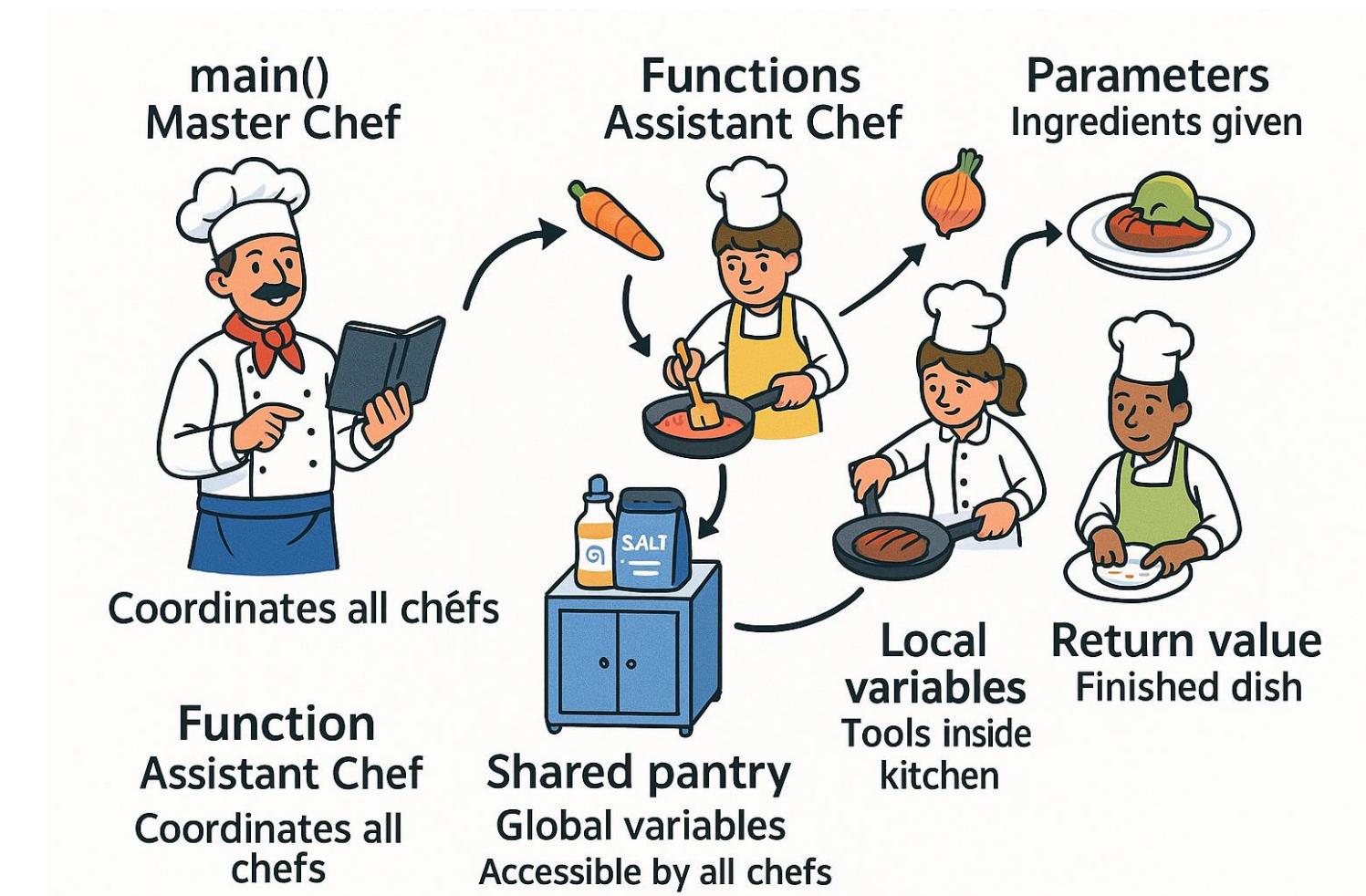
In programming, think of the `main()` function as the **Master Chef** of your restaurant.

The Master Chef doesn't cook every dish alone - instead, they give orders to other chefs (functions).

Each chef has their own ingredients (parameters) and tools (local variables).

When their part is done, they hand back a finished dish (return value).

Together, they create the full course meal — the complete program.



Cooking Analogy for Functions, Parameters, and Variables

main() Function = Master Chef (Head Chef)

the one who manages the entire cooking process and coordinates all the other chefs.

Explanation:

The Master Chef doesn't cook everything alone.

Instead, they **organize the workflow**, decide **which chef (function)** to call, and in what order, to complete the full meal.

```
int main(void) {  
    prepareIngredients();  
    makeSauce();  
    grillSteak();  
    serveDish();  
    return 0;  
}
```

In this example,
the **Master Chef (main)** gives orders to other chefs (prepareIngredients, makeSauce, grillSteak, serveDish) : each of whom performs a specific task independently.

Cooking Analogy for Functions, Parameters, and Variables

Functions = Assistant Chefs

Each function represents an **assistant chef** who handles one specific part of the recipe.

Example:

- The *sauce chef* makes the sauce.
- The *grill chef* cooks the steak.
- The *plating chef* arranges the food on the dish.

They all report back to the **Master Chef (main())** once their part is done.

Cooking Analogy for Functions, Parameters, and Variables

Parameters = Ingredients Given to the Chef

Parameters are like the **ingredients** you give to an assistant chef when you assign them a task.

makeSauce(tomatoes, onions, salt);

You provide ingredients; the chef uses them inside their own kitchen.

When the cooking ends, those ingredients (local variables) are gone.

Return Value = Finished Dish

The **return value** is the **dish** the assistant chef gives back to the Master Chef.

sauce = makeSauce(tomatoes, onions, salt);

The Master Chef can then use the sauce for another dish or combine it with other results.

Cooking Analogy for Functions, Parameters, and Variables

Local Variables = Tools Inside Each Kitchen

Local variables are like the **pans, bowls, and utensils** inside each chef's workspace. Only that chef can use them — others can't even see them.

```
void makeSauce() {  
    int temperature = 90; // private tool for this chef only  
}
```

When cooking ends, the tools are cleaned up — just like local variables are destroyed when the function exits.

Global Variables = Shared Pantry

Global variables are like a **shared pantry or refrigerator** that all chefs can access.

Example:

The salt and oil stored in the shared pantry can be used by any chef, but if one chef finishes all the salt, others are affected — which is why overusing global variables can cause problems.

```
int salt = 100; // shared pantry
```

Why Do Programs Need Functions?

1. Divide and Conquer — Breaking Big Problems into Smaller Tasks

- A function allows you to split a big task into smaller, independent pieces.

2. Reusability — Write Once, Use Many Times

- Once you've written a function, you can call it **anytime, anywhere** without rewriting the same code.

3. Readability and Organization

- Functions make your code look like a **step-by-step story** instead of a wall of logic

4. Testing and Debugging

- Functions help isolate problems.
- If something breaks, you only need to test or fix **one small part**, not the whole program.

5. Collaboration

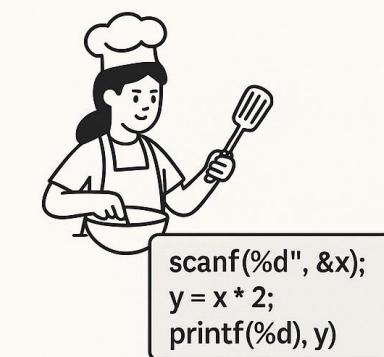
- In large projects, different programmers can work on **different functions** at the same time.

6. Recursion and Abstraction

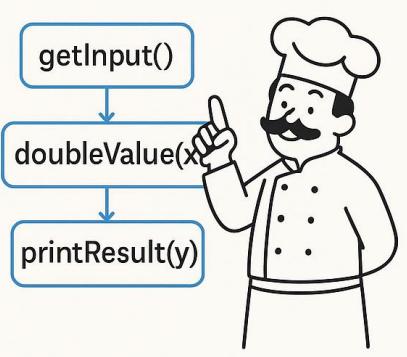
- Functions allow you to think at a **higher level** of abstraction.
- Focus on *what* a function does, not *how* it does it

Why Programs Need Functions

Without Functions



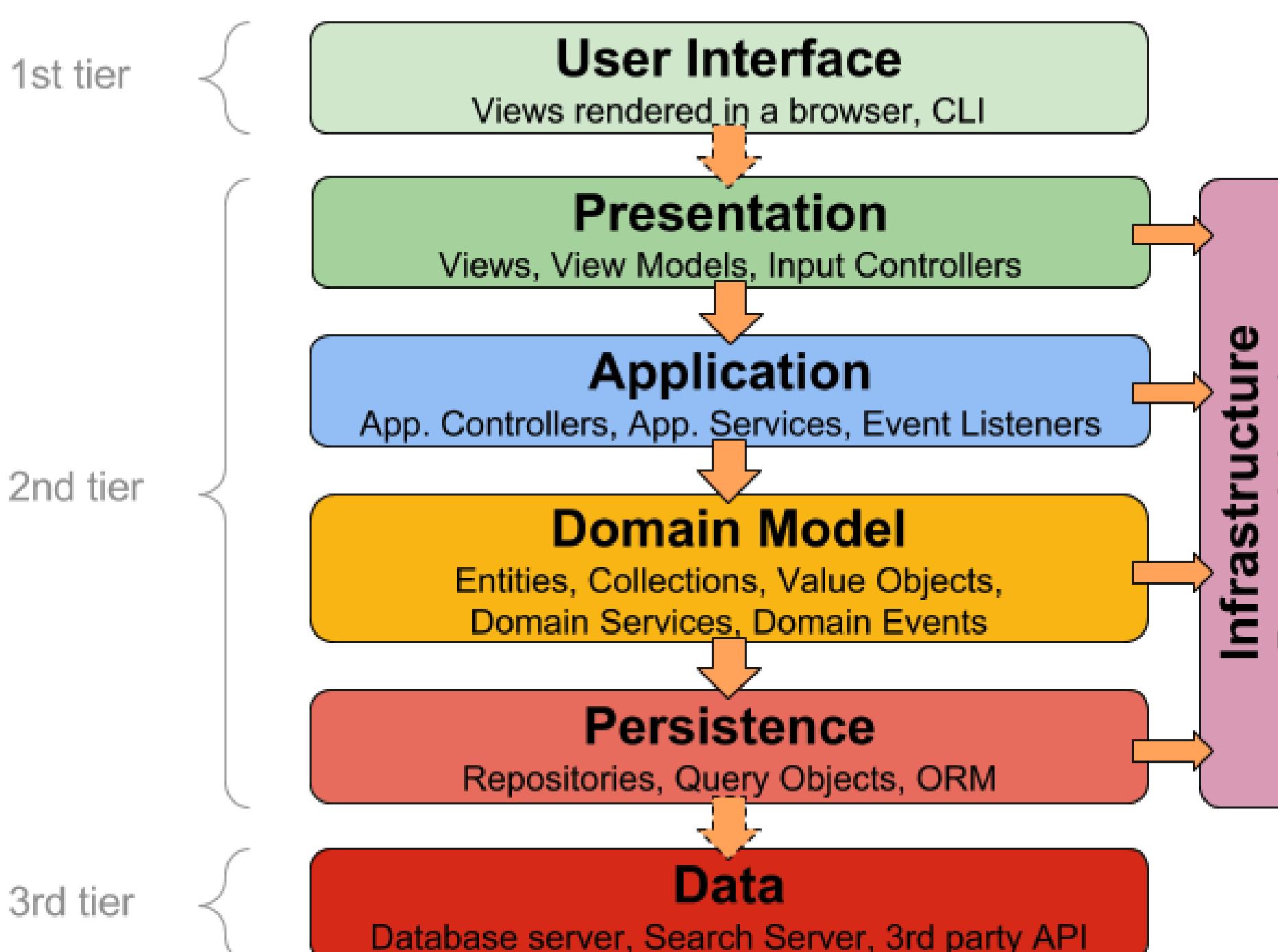
With Functions



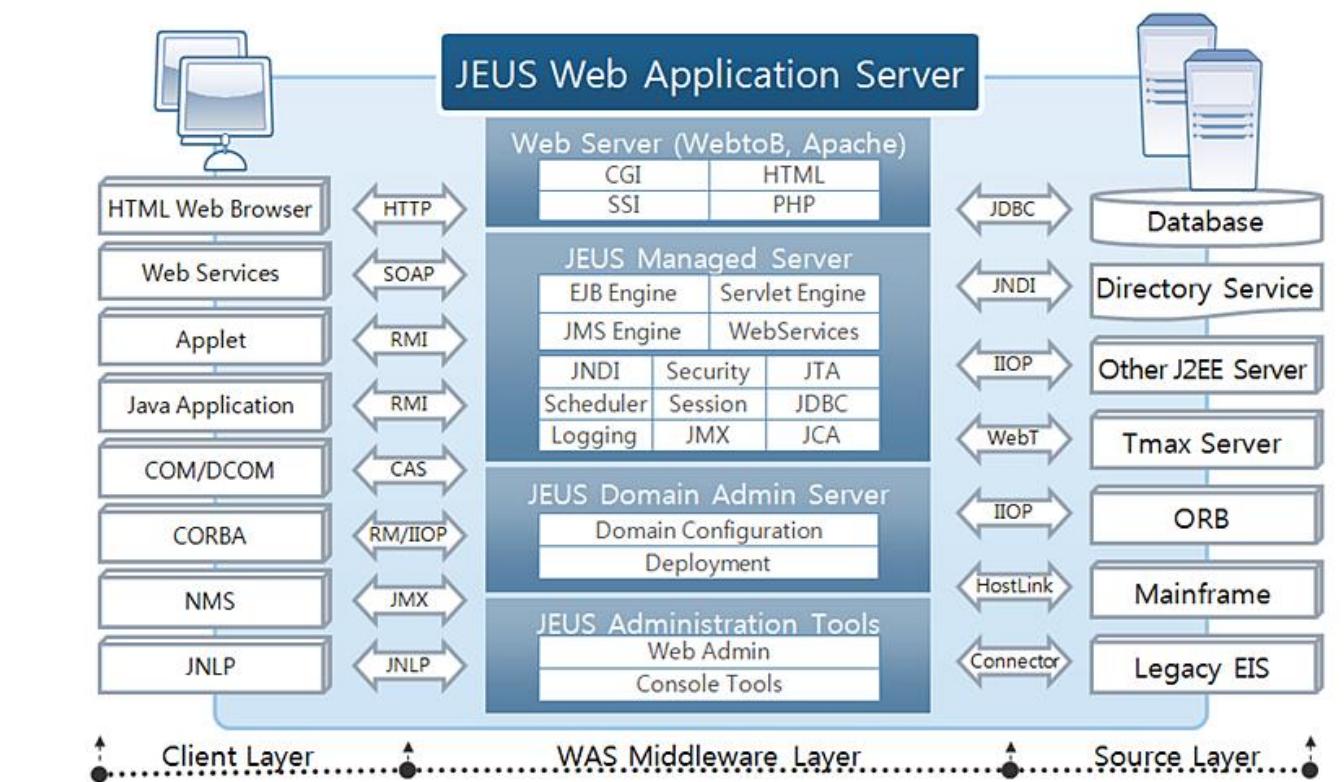
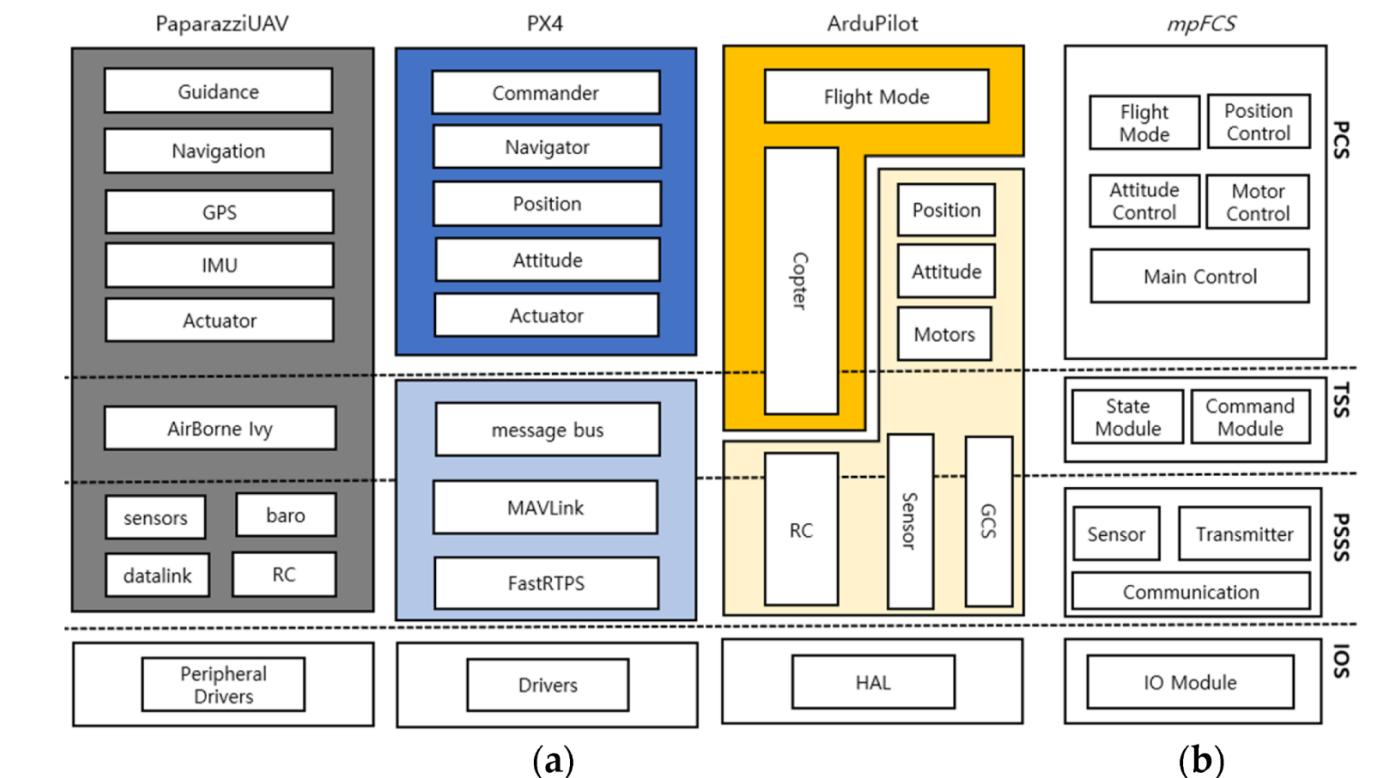
- Break up complex problems
- Avoid repetitive code
- Improve readability and organization
- Isolate functionality for testing

- Enable teamwork and collaboration
- Abstract away details

SW Structure (Big picture)


www.herbertograca.com

SW Area: UI, UX, App, Middleware, Driver, DB, Network
(how to communication? API, interface)



Header files: Std Lib

C:\MinGW64\include

C:\MingW64\x86_64-w64-mingw32\include

C:\my_project_folder\include

Object file : gcc -c main.c -o main.o

Executable file : gcc main.o -o main

C:\MingW64\x86_64-w64-mingw32\lib

1. Student characteristics get/set functions (student.h, student.o)

int setStudentInfo(char* name, char* ID, char* gender);

char* getName(int id); / char* getID(char*); / char* getGender(char*);

2. Grade-related get/set functions (grade.h, grade.o)

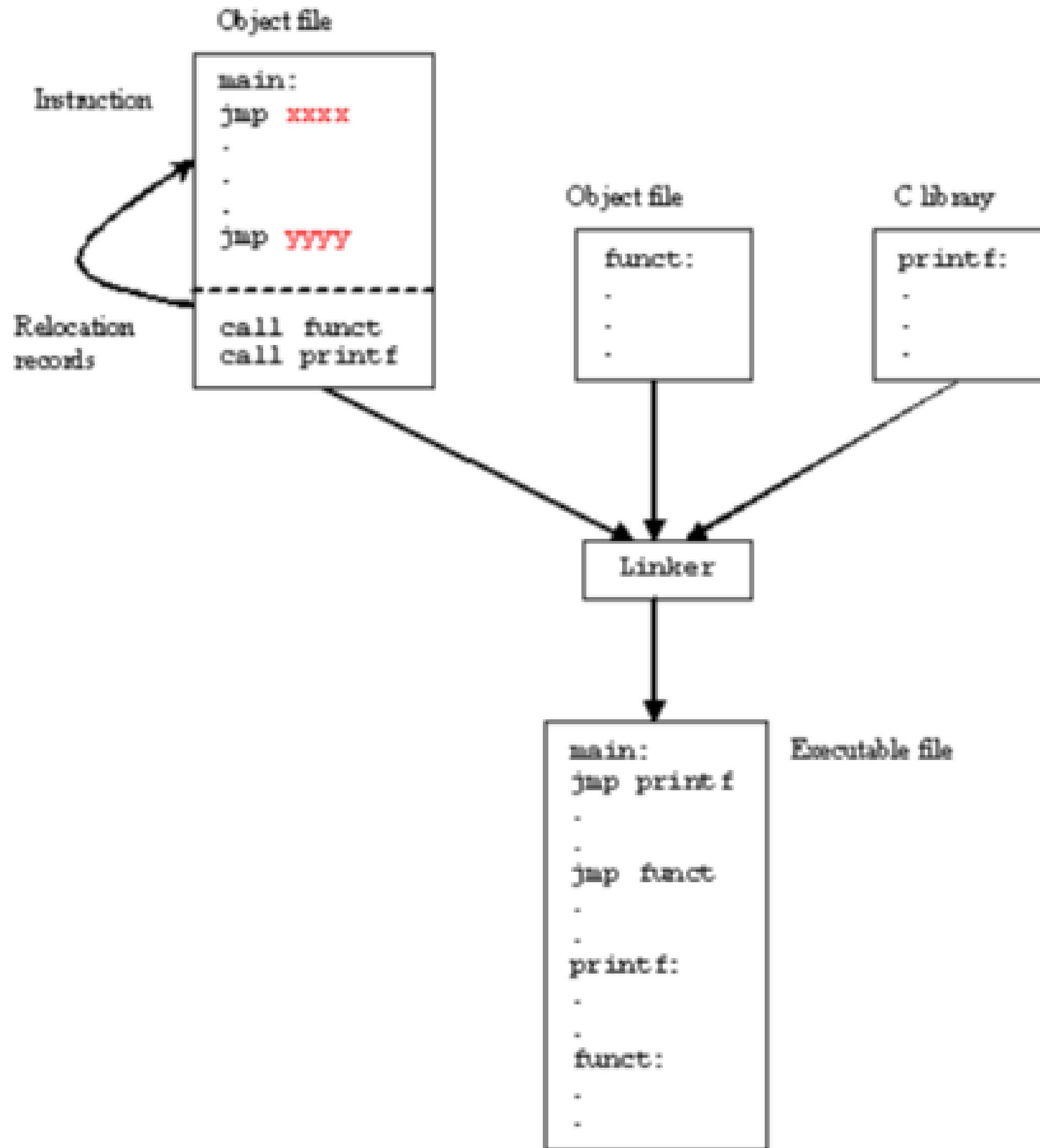
int setScore(char* name, char* class, int score); / int getScore(char* name, char* class);

3. Statistics-related functions (statistics.h, statistics.o)

int getTotalScore(char* name);

int getAverage(char* name);

int getGradeRanking(char* name);



Library

| File Type | Extension | Purpose | Linking Type | Usage |
|-----------------|-----------------------------|--|-------------------|--|
| Object File | .o | Contains compiled code for a single source file. | Static or Dynamic | Intermediate file created during compilation, linked to create executable. |
| Static Library | .lib (Windows) .a (Unix) | Collection of object files packaged together. | Static | Linked into executable at compile time; no external dependencies at runtime. |
| Shared Library | .so (Unix) | Dynamic library loaded at runtime. | Dynamic | Shared among multiple programs, linked at runtime. |
| Dynamic Library | .dll (Windows) | Similar to .so, used on Windows for dynamic linking. | Dynamic | Loaded at runtime by programs that use the library. |

Ex) codespace => nm ch7.o

Function

Material Producer(Farmer) - [Language Developer](#)

Wholesaler - [Library Developer](#)

Retailer - [Framework Developer](#)

Chef - Application Developer (Student - [Program Learner](#)) → C : C Language + Standard Lib (header + object)

Python : Python Language + Standard Lib + Framework

In the case of cooking, it is divided by the degree of completion of the ingredients, and the usage of each ingredient is distributed to the upper user.

In the case of the program, it is divided by the task scope, and the usage of each function is distributed to the upper user.

Structure (Student grading system at school)

Well-Organized functions

(Re-use @ Elementary, middle, and high schools, all schools)

Functions for specific purposes:

1. Statistics-related functions

- get total score
- get average
- get average of the entire class
- get average by school

Common Features
ID, Name, Gender, Score etc



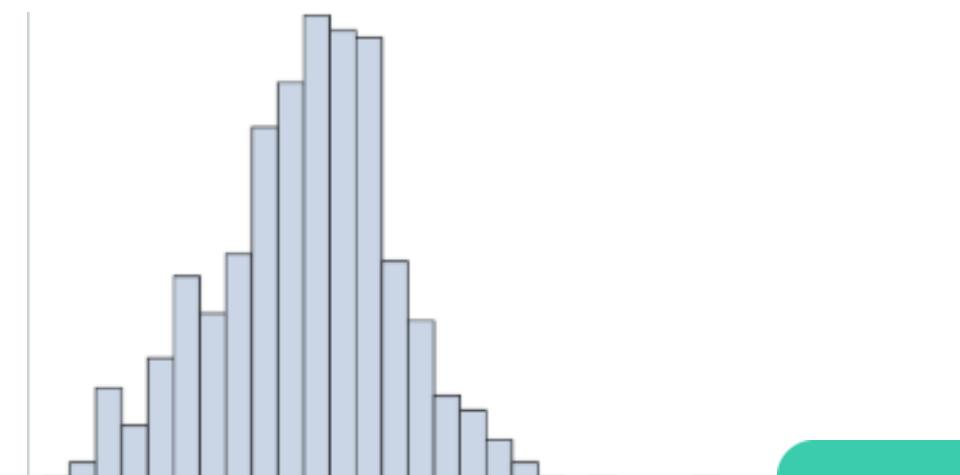
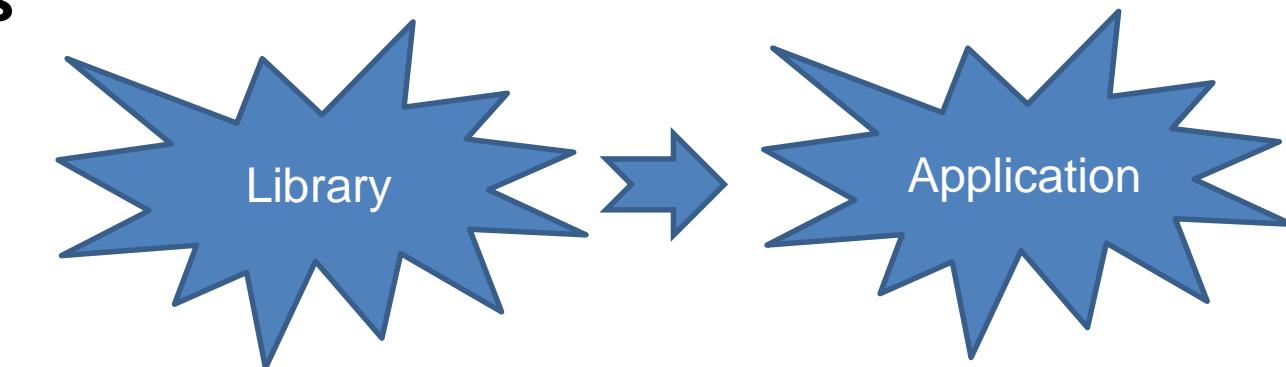
2. Grade-related get/set functions

- get math class score
- get C class score

3. Student characteristics get/set functions

- set / get name
- set / get ID
- set / get gender

| Student ID | Name | Gender | Age | English Grade | Math Grade | Science Grade |
|------------|------------|--------|-----|---------------|------------|---------------|
| 1 | John Doe | Male | 15 | A | B | B+ |
| 2 | Jane Smith | Female | 16 | B+ | A- | A |
| 3 | Tom Brown | Male | 14 | C | C | C+ |
| 4 | Sarah Lee | Female | 15 | A- | B+ | A- |
| 5 | Alex Wang | Male | 16 | A+ | A+ | A+ |



See you next week!
DO NOT miss the classes

