

C Programming (W3)



Welcome!!

Please check attendance individually.
(Mobile App)

Things to do today

01 | Language

02 | Programming

03 | Variables

04 | Operators

Unicode

C was originally designed for **ASCII (1-byte characters)**, but modern applications require **Unicode** support to handle characters from multiple languages. Unicode works through different encoding formats such as **UTF-8, UTF-16, and UTF-32**.

Unicode is a universal character set that assigns a unique number to every character in every language.

ex) 'A' → U+0041 (Just number)

'가' → U+AC00

'あ' → U+3042

Unicode defines the meaning of characters but not how they are stored in memory or files.

Range: U+0000 ~ U+10FFFF → a total of 1,114,112 possible code points.

Within this range, Unicode is designed to represent all the world's scripts, emojis, symbols, and even ancient characters.

Method of expression using UTF-8:

- U+0000 ~ U+007F → 1 byte
- U+0080 ~ U+07FF → 2 bytes
- U+0800 ~ U+FFFF → 3 bytes
- U+10000 ~ U+10FFFF → 4 bytes

Unicode Encoding in C

Encoding: a way to encode Unicode characters as bytes, so they can be stored or transmitted.

Unicode characters can be represented using:

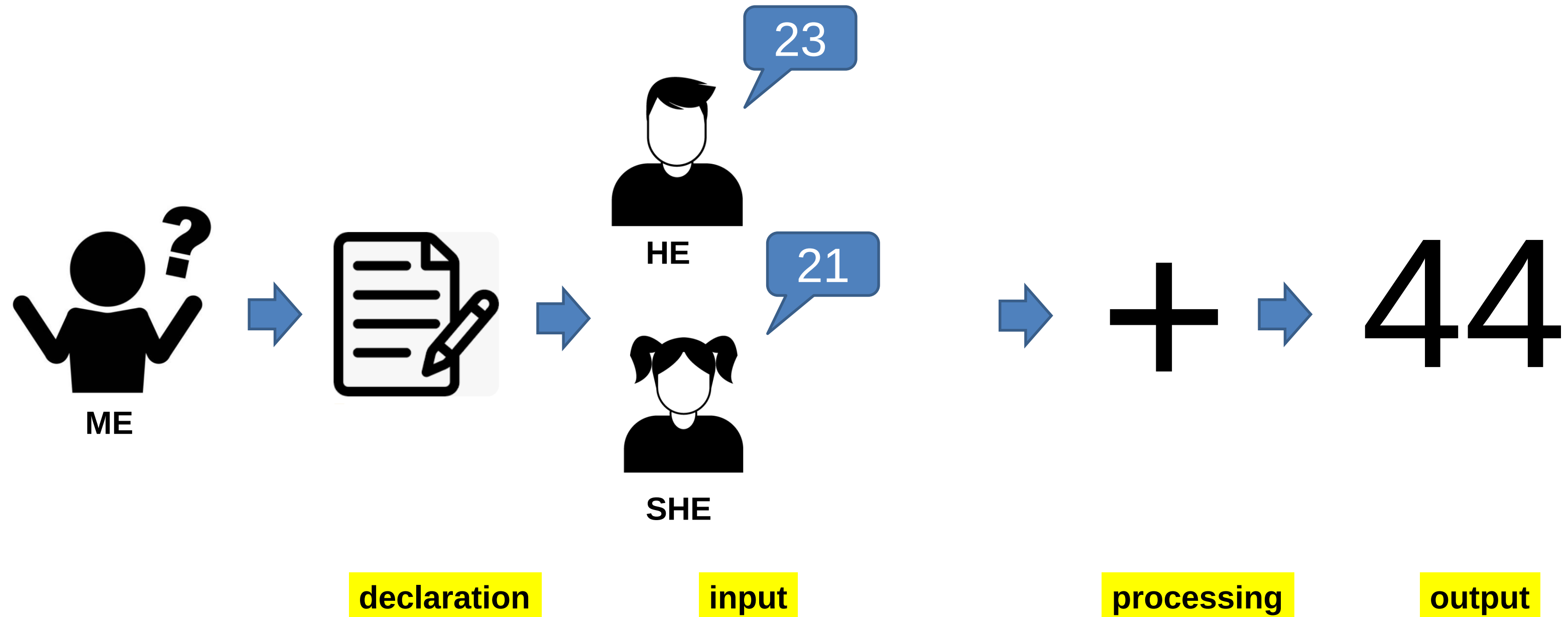
- **UTF-8** (variable-length, 1–4 bytes) → Most widely used (default in Linux/macOS)
 - ASCII character: 1 byte
 - European character: 2 bytes
 - Korean/Chinese etc: 3 ~ 4 bytes
- 1 byte: Basic ASCII characters (e.g. A, B, C, 0-9) → 0xxxxxxx
- 2 bytes: European languages, Korean initial consonants (e.g. é, ñ, ü, 한) → 110xxxxx 10xxxxxx
- 3 bytes: Most Korean characters, Hanja, some emojis → 1110xxxx 10xxxxxx 10xxxxxx
- 4 bytes: Extended characters and emojis (e.g. 😊, 🏆) → 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

Think over

I am trying to write a C program that takes two people's ages and calculates their sum, following the input instructions.

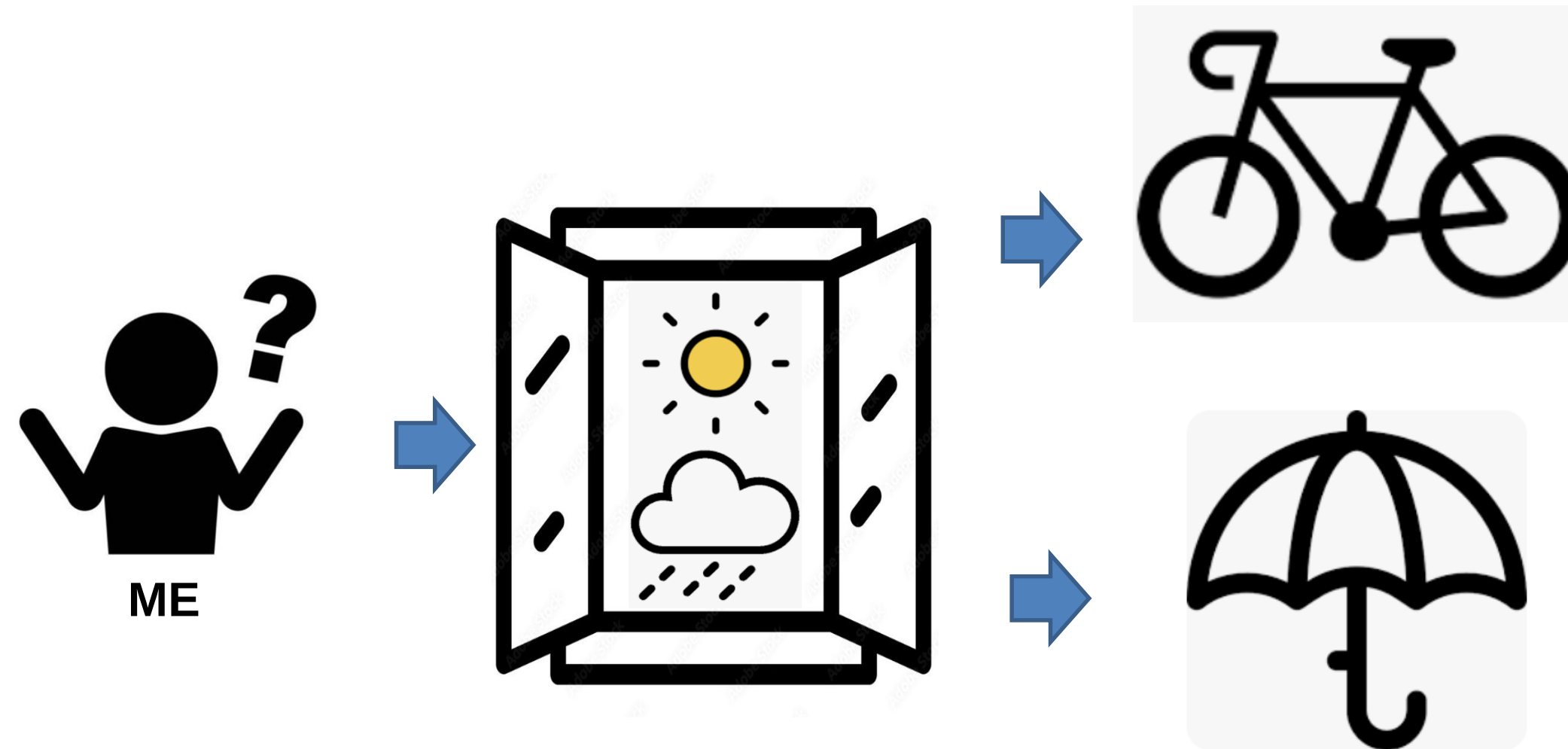
What is Programing?

- Program operation process



What is Programing?

- Program operation process



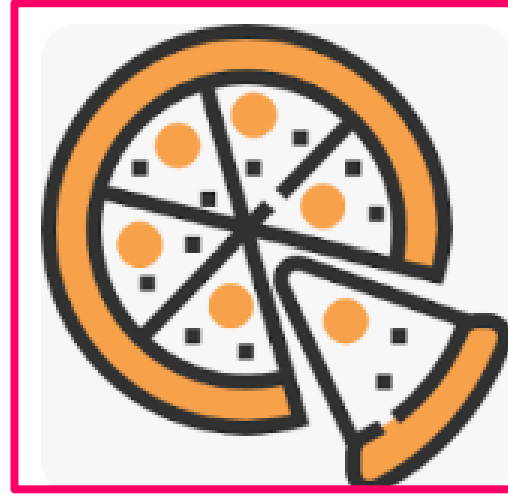
```
int Bicycle, Umbrella;
```

```
If(weather == "rainy")
{
    printf("Umbrella");
}
else
{
    printf("Bicycle");
}
```

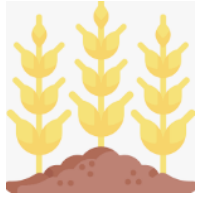
Think over

You are on an island with nothing. You want to cook pizza or spaghetti. Think about what you need to prepare. Imagine a list of things you need.

Let's cook



Making pizza base



Cooking pizza



The real appearance of printf function

- Source code : <https://github.com/lattera/glibc/tree/master/stdio-common>

Let's make pizza



Potato



Beef



Pepperoni



What should I consider?

1. Select dough(가루 반죽)
* Crust / Base

Thin / Thick

What kinds of wheat flour(밀가루)

2. Select sauce / cheese / topping

Tomato, Olive oil, Garlic, Herb, BBQ Sauce

Mozzarella, Gorgonzola, ...

Potato, Pepperoni, Beef,

3. Baking condition

Oven – 15 mins

Pan - 30 mins

Programming Process

- The programming process is similar to the cooking process.



1. Problem Analysis & Requirements Definition (Preparation)



2. Selecting Data & Tools (Gathering Ingredients)



3. Coding, Implementation (Cooking)



4. Debugging & Testing (Tasting & Adjusting)



5. Deployment & Release (Serving the Dish)



6. Maintenance (Feedback & Improvement)



Programming Process



1. Problem Analysis & Requirements Definition

(Programming) Understanding the problem and defining requirements

- Decide what program to create
- Define the necessary features and functionalities

(Cooking) Deciding what dish to make before cooking

- Choose a menu and find a recipe
- Identify the ingredients needed

Example:

Program: "Create a user login system"

Dish: "Make pasta"

Programming Process



2. Selecting Data & Tools (Gathering Ingredients)

(Programming) Choosing the programming language, tools, and data

- Decide on the programming language (Python, Java, C++, etc.)
- Select necessary libraries and frameworks, IDE

(Cooking) Preparing ingredients before cooking

- Gather pasta, olive oil, garlic, tomato sauce, and noodles
- Prepare kitchen tools like a frying pan and pot

Example:

Program: C + Standard Lib + VSCode + Github

Dish: Using spaghetti with tomato sauce

Programming Process



3. Coding, Implementation (Cooking)

(Programming) Writing the actual code to build the program

- Define variables, write functions, and implement algorithms
- Be mindful of errors and bugs

(Cooking) Preparing and cooking the dish

- Chop the ingredients (garlic), boil the pasta
- Control the heat while cooking the sauce and mixing everything

1. Design (Define requirement)
2. Write source code
3. Compile & Link
4. Execute a program
5. Debugging
6. Store & Maintaining

Example:

Program: Writing a void main(): function and implementing “hello world!”

Dish: Fry the garlic, boil the sauce and cook with the noodles

Programming Process



4. Debugging & Testing (Tasting & Adjusting)

(Programming) Checking if the program works correctly and fixing errors

- Find & fix bugs
- Test with different inputs to ensure reliability

(Cooking) Tasting the dish and adjusting flavors

- If it's too bland, add salt; if it's too salty, add water
- Adjust seasoning for the best taste

Example:

Program: Fixing a bug where login credentials are not verified correctly

Dish: Adjusting the seasoning by adding salt or pepper

Programming Process



5. Deployment & Release (Serving the Dish)

(Programming) Deploying the program for users

- Storing at github or publishing an app
- Sharing it with users

(Cooking) Serving the finished dish

- Plating the food in an appealing way
- Serving it to family or customers

Example:

Program: Deploying the website on AWS or Github

Dish: Serving the pasta to guests

Programming Process



6. Maintenance (Feedback & Improvement)

(Programming) Updating and improving the program based on user feedback

- Adding new features and security updates
- Continuous maintenance and bug fixes

(Cooking) Improving the dish based on feedback

- If guests say the dish is too salty, adjust it next time
- Experiment with new recipes to enhance flavors

Example:

Program: Optimizing login speed if it's slow

Dish: "The pasta is overcooked" → Reduce boiling time

Operators

1. Arithmetic Operators

Used to perform basic mathematical operations.

Operator	Description	Example
+	Addition	$a + b$
-	Subtraction	$a - b$
*	Multiplication	$a * b$
/	Division	a / b
%	Modulus (remainder)	$a \% b$

2. Relational (Comparison) Operators

Used to compare two values.

Operator	Description	Example
==	Equal to	$a == b$
!=	Not equal to	$a != b$
>	Greater than	$a > b$
<	Less than	$a < b$
>=	Greater or equal	$a >= b$
<=	Less or equal	$a <= b$

Operators

3. Logical Operators

Used to combine or invert boolean expressions.

Operator	Description	Example
&&	Logical AND	a && b
!	Logical NOT	!a

4. Assignment Operators

Used to assign values to variables.

Operator	Description	Example
=	Assign	a = b
+=	Add and assign	a += b
-=	Subtract and assign	a -= b
*=	Multiply and assign	a *= b
/=	Divide and assign	a /= b
%=	Modulus and assign	a %= b

Special Characters

- { } (Curly braces - Used for code blocks)
- [] (Square brackets - Used for arrays)
- () (Parentheses - Used for functions and expressions)
- ; (Semicolon - Statement terminator)
- : (Colon - Used in labels and ternary operator)
- # (Hash - Preprocessor directive)
- " (Double quotes - String literals)
- ' (Single quotes - Character literals)
- \ (Backslash - Escape sequences)
- // (Single-line comment)
- /* */ (Multi-line comment)

Operator

- + (Addition)
- (Subtraction)
- * (Multiplication, Asterisk)
- / (Division)
- % (Modulus / Remainder)
- = (Assignment)
- == (Equal to, Comparison)
- != (Not equal to)
- > (Greater than)
- < (Less than)
- >= (Greater than or equal to)
- <= (Less than or equal to)
- && (Logical AND)
- || (Logical OR)
- ! (Logical NOT)

- & (Bitwise AND / Address-of operator, Ampersand)
- | (Bitwise OR)
- ^ (Bitwise XOR)
- ~ (Bitwise Complement)
- << (Left shift)
- >> (Right shift)
- += (Addition assignment)
- = (Subtraction assignment)
- *= (Multiplication assignment)
- /= (Division assignment)
- %= (Modulus assignment)
- &= (Bitwise AND assignment)
- |= (Bitwise OR assignment)
- ^= (Bitwise XOR assignment)
- <<= (Left shift assignment)
- >>= (Right shift assignment)
- ++ (Increment)
- (Decrement)
- > (Structure pointer access)
- . (Structure member access)
- ? : (Ternary conditional operator)
- , (Comma operator)

Homework 2: Basic operations

- //

Debugging

- Practice with debugger
- VSC (launch.json, tasks.json)

See you next week!

DO NOT miss the classes

