AI & Big Data

C Programming (W5)

Welcome!!

Please check attendance individually.

(Mobile App)

miricanvas.com

Things to do today

01 Ch.3, Ch.4 ~ Ch.5

02 Standard Input / Output

Convention of C language:
- Comment /* */, //
- Indentation
- Clean & readable code

Syntax rule of C language
- int a;
- Compile error

Notice – Modify what I mentioned

Build Process (*The build process consists of compilation and linking.)*

- Compilation

. Translating source code (.c files) into object code (.o or .obj files).

. Each source file is compiled independently by the compiler.

- Linking

. Combining object files and libraries into a final executable (.exe on Windows, no extension on Linux).

. Resolves external references (e.g., function calls, global variables).

The entire workflow = Compilation + Linking (and sometimes preprocessing, optimization, packaging).

Problem solving with stdio.h

1. Determine what the input is
   - scanf, fgets, getchar, sscanf

User scenario

```
> 1
> 1 3.14
> name age email
> name age email introduction
> name,age,email,introduction
> Name:abc, Age:20, Email:abc@email.com, Introduction: I am a boy
> word sentence number etc
```

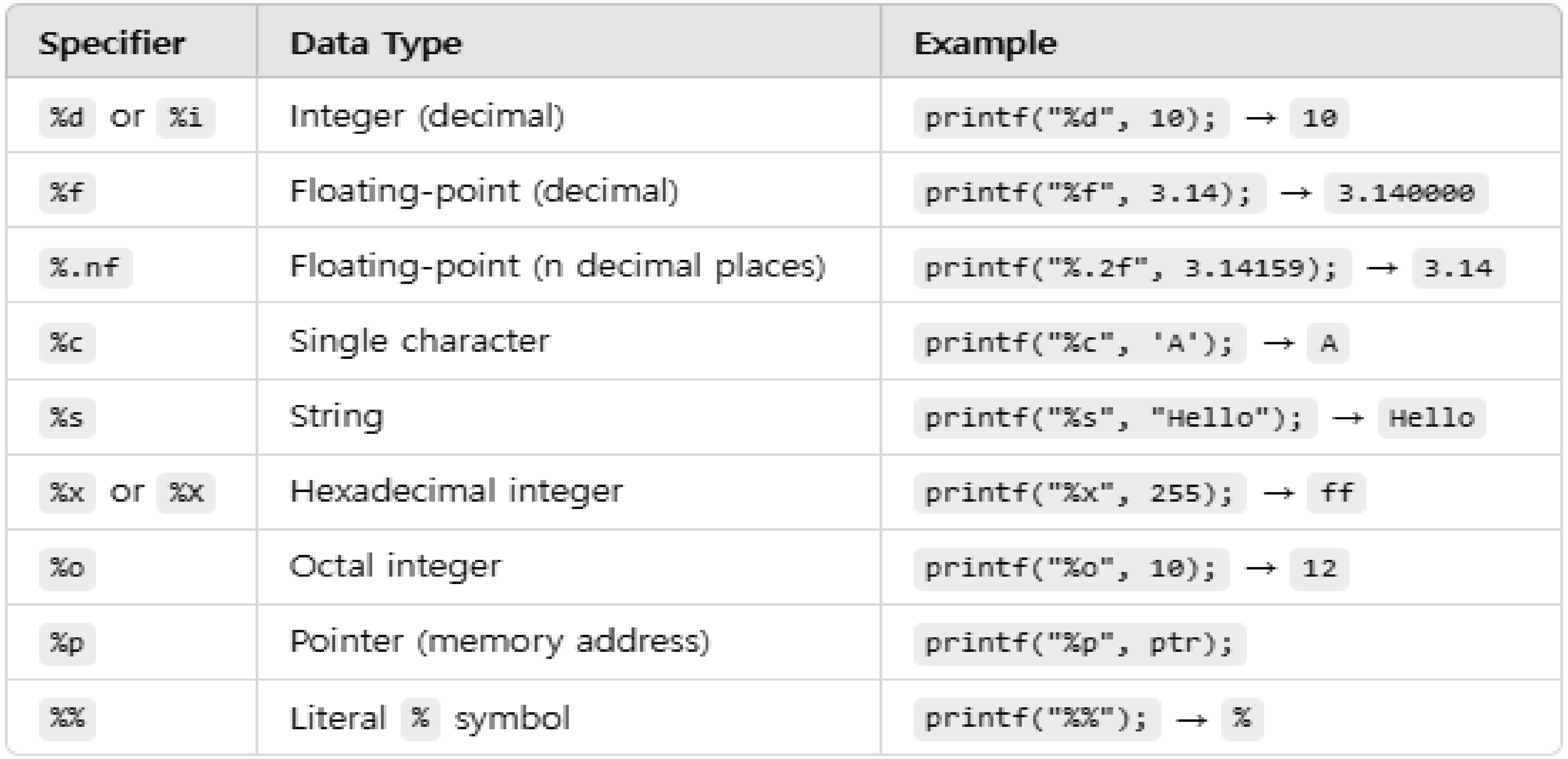
WOOSONG UNIVERSITY AI & Big Data

Standard Output (printf)

1. Syntax of **printf()** : Grammar and Structure of language

```
#include <stdio.h>
int printf(const char *format, ...);
```

- The first argument (format) is a string containing text and format specifiers
- The ellipsis (...) represents a variable number of arguments, which are inserted into the format specifier

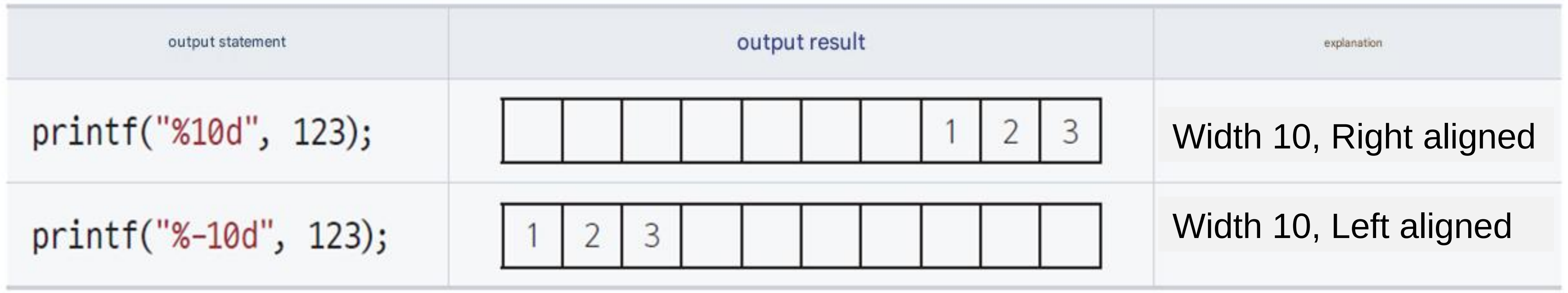


Specifier	Data Type	Example
`%d` or `%i`	Integer (decimal)	`printf("%d", 10);` → `10`
`%f`	Floating-point (decimal)	`printf("%f", 3.14);` → `3.140000`
`%.nf`	Floating-point (n decimal places)	`printf("%.2f", 3.14159);` → `3.14`
`%c`	Single character	`printf("%c", 'A');` → `A`
`%s`	String	`printf("%s", "Hello");` → `Hello`
`%x` or `%X`	Hexadecimal integer	`printf("%x", 255);` → `ff`
`%o`	Octal integer	`printf("%o", 10);` → `12`
`%p`	Pointer (memory address)	`printf("%p", ptr);`
`%%`	Literal `%` symbol	`printf("%%");` → `%`

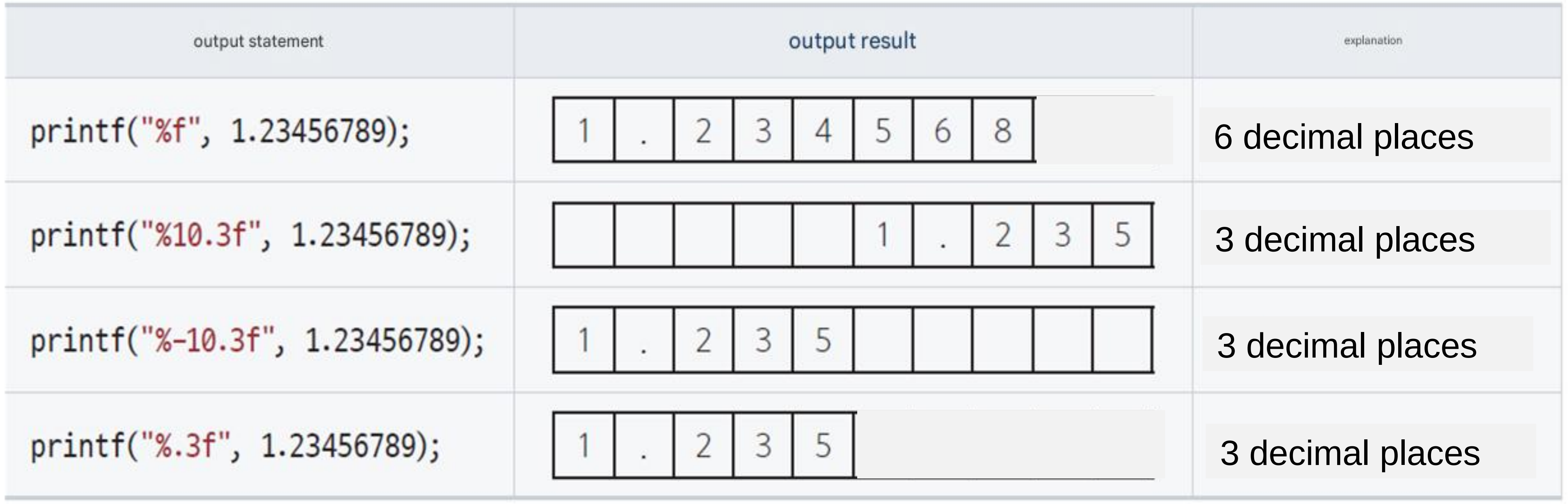
Standard Output (printf)

Field width and precision

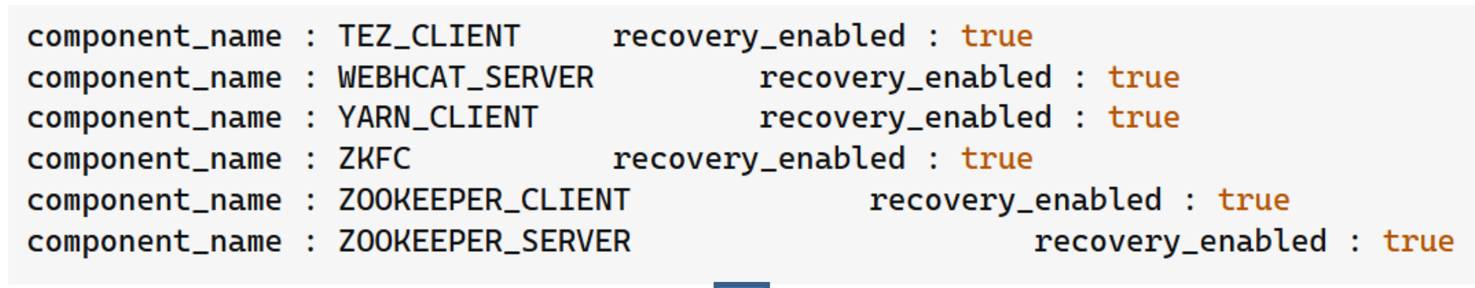
When printing using printf() , you can specify the size of the field in which data is printed .



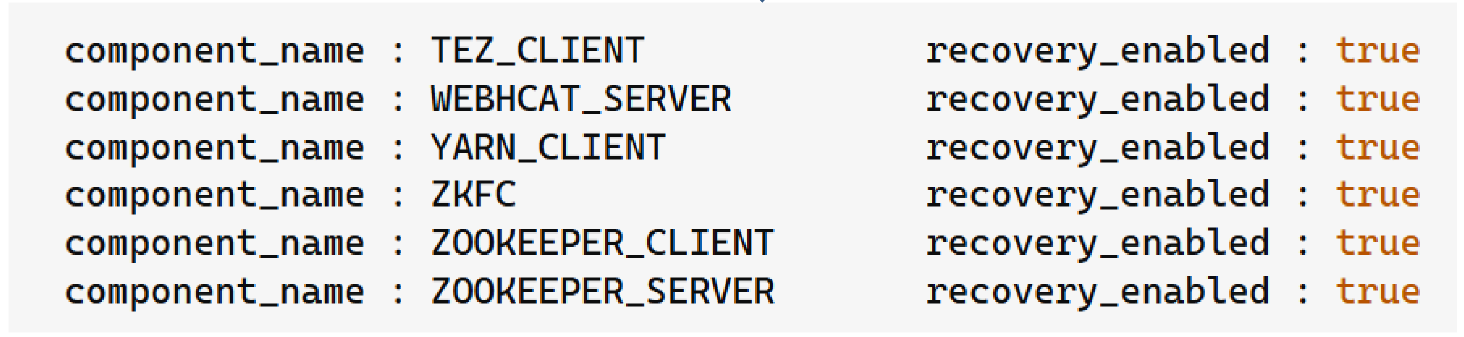
output statement	output result	explanation
`printf("%10d", 123);`	`       123`	Width 10, Right aligned
`printf("%-10d", 123);`	`123       `	Width 10, Left aligned



output statement	output result	explanation
`printf("%f", 1.23456789);`	`1.234568`	6 decimal places
`printf("%10.3f", 1.23456789);`	`     1.235`	3 decimal places
`printf("%-10.3f", 1.23456789);`	`1.235     `	3 decimal places
`printf("%.3f", 1.23456789);`	`1.235`	3 decimal places



```
component_name : TEZ_CLIENT     recovery_enabled : true
component_name : WEBHCAT_SERVER         recovery_enabled : true
component_name : YARN_CLIENT            recovery_enabled : true
component_name : ZKFC           recovery_enabled : true
component_name : ZOOKEEPER_CLIENT               recovery_enabled : true
component_name : ZOOKEEPER_SERVER                       recovery_enabled : true
```



```
component_name : TEZ_CLIENT             recovery_enabled : true
component_name : WEBHCAT_SERVER         recovery_enabled : true
component_name : YARN_CLIENT            recovery_enabled : true
component_name : ZKFC                   recovery_enabled : true
component_name : ZOOKEEPER_CLIENT       recovery_enabled : true
component_name : ZOOKEEPER_SERVER       recovery_enabled : true
```

Standard Output (printf)

AI & Big Data

3. Format of printf()

- Width and Alignment
  You can specify a **minimum width** for the output using numbers.
  Default is right-aligned; to left-align, use -

```
printf("%10d\n", 123);  // Right-aligned, width 10
printf("%-10d\n", 123); // Left-aligned, width 10
```

Practice

- Precision for Floating-Point Numbers

```
printf("%.2f\n", 3.14159);  // Prints with 2 decimal places
```

Practice

- Padding with Zeros

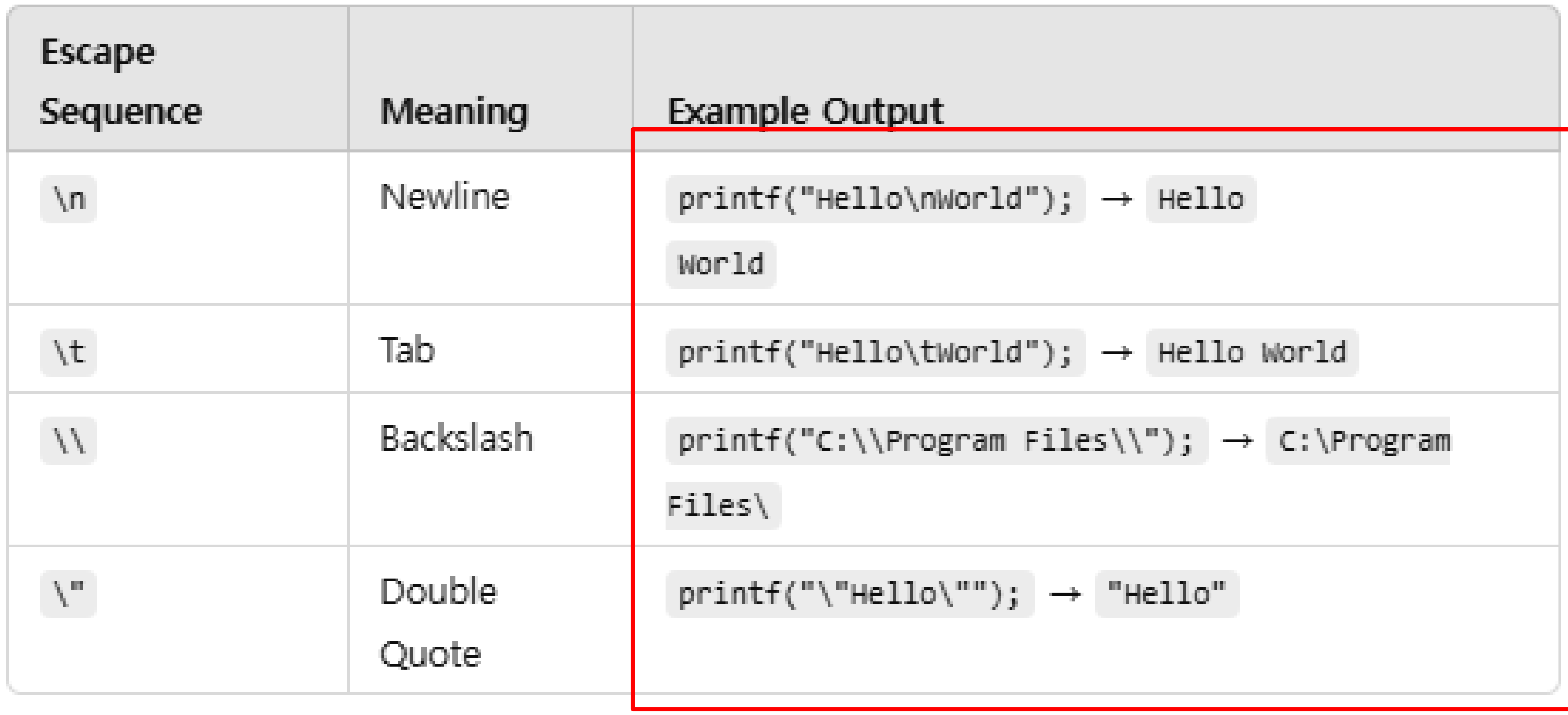
```
printf("%05d\n", 42);  // Pads with zeros up to 5 digits
```

Practice

Standard Output (printf)

4. Using Escape Sequences of printf()

- printf supports escape sequences to control output formatting



Escape Sequence	Meaning	Example Output
`\n`	Newline	`printf("Hello\nWorld");` → `Hello World`
`\t`	Tab	`printf("Hello\tWorld");` → `Hello World`
`\\`	Backslash	`printf("C:\\Program Files\\");` → `C:\Program Files\`
`\"`	Double Quote	`printf("\"Hello\"");` → `"Hello"`

Practice

Standard Output (printf)

AI & Big Data

5. Printing Multiple Values

- You can print multiple values in a single printf call by passing multiple

```
int age = 25;
float pi = 3.14;
printf("Age: %d, Pi: %.2f\n", age, pi);
```

Practice

5. Return Value of printf

-printf returns the number of characters printed (excluding \0)

```
int count = printf("Hello");
printf("\nCharacters printed: %d\n", count);
```

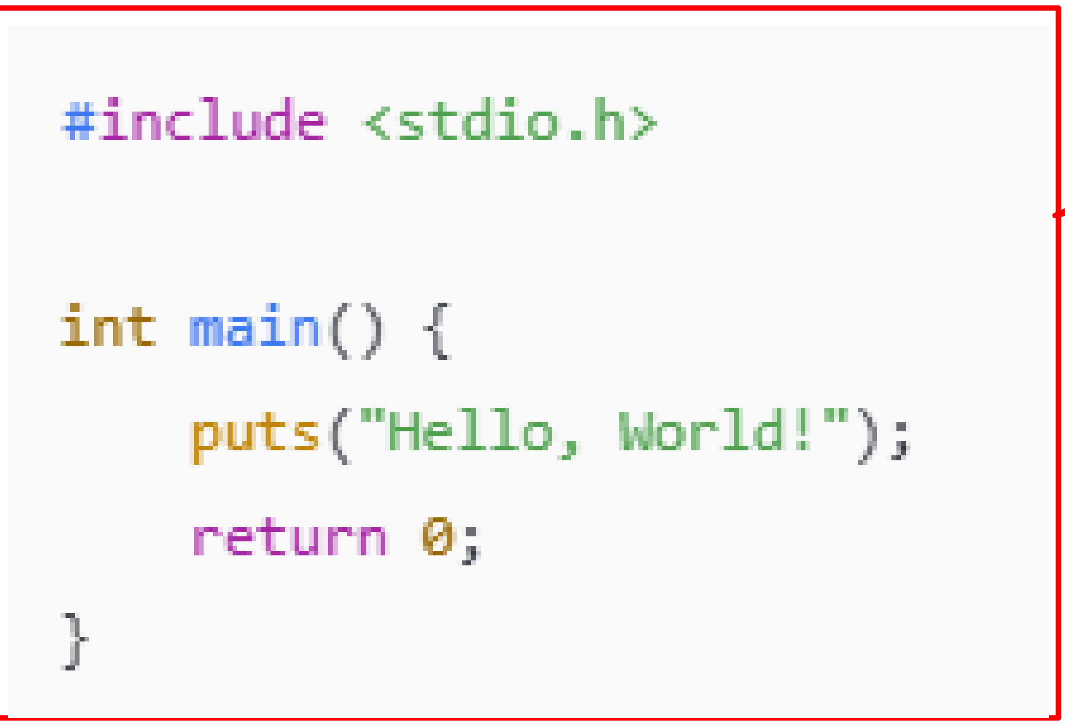
Practice

Standard Output (puts)

1. Syntax of **puts**()

#include <stdio.h>
int puts(const char *str);

- puts prints a string (str) to the console and automatically appends a newline (\n) at the end.
- It is simpler and safer than printf("%s\n", str); because it does not require format specifiers.
- It returns a non-negative integer on success and EOF (-1) on failure.



```c
#include <stdio.h>

int main() {
    puts("Hello, World!");
    return 0;
}
```

Practice

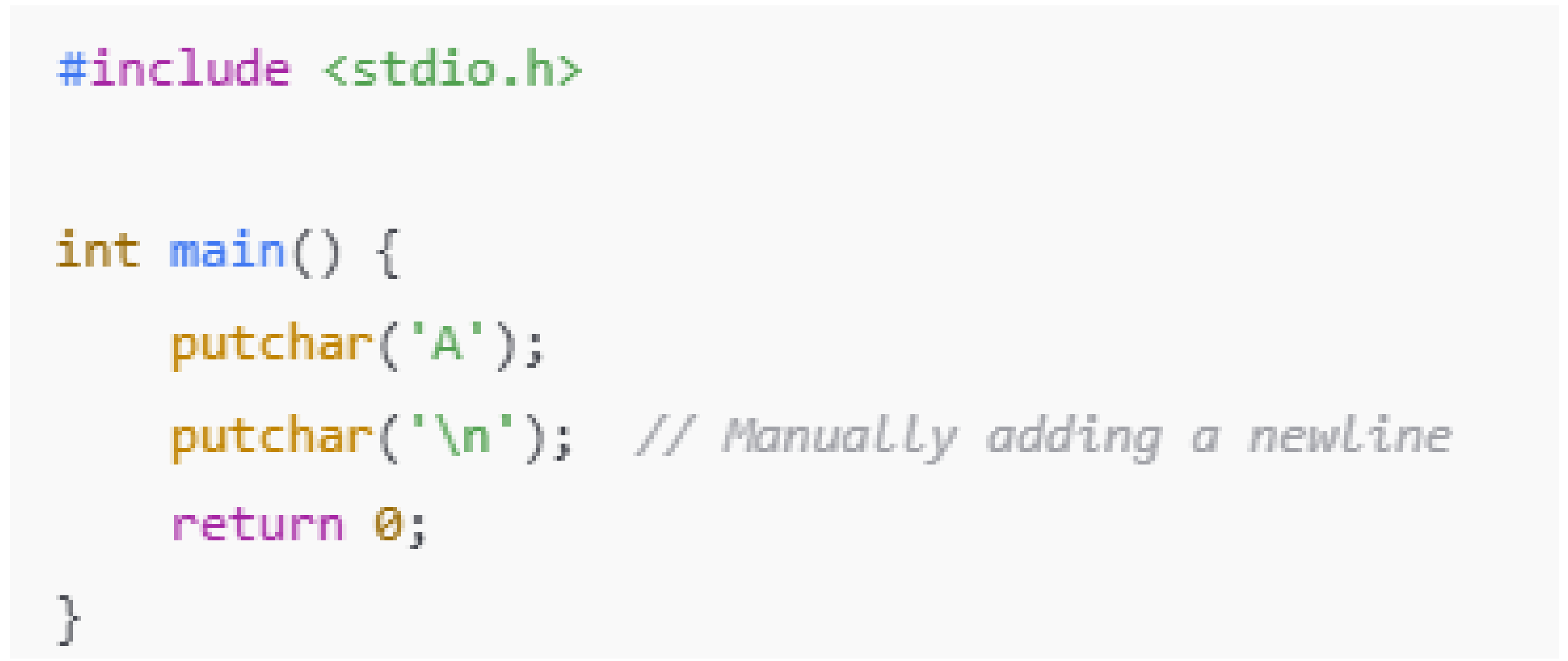
Standard Output (putchar)

AI & Big Data

1. Syntax of **putchar**()

```
#include <stdio.h>
int putchar(int ch);
```

- putchar prints a single character (ch) to the console.
- It is simpler and safer than printf("%s\n", str); because it does not require format specifiers.
- It returns a non-negative integer on success and EOF (-1) on failure.



```
#include <stdio.h>

int main() {
    putchar('A');
    putchar('\n');  // Manually adding a newline
    return 0;
}
```

Practice

Standard Input

Standard input reads data from an input device, typically the keyboard.

It uses functions like scanf(), getchar(), and fgets() to read user input

```
[Keyboard typing]
      ↓
[OS input buffer (stdin)]
      ↓ (flush when Enter pressed)
[Program function: scanf/getchar/fgets...]
      ↓
[Stored in variable]
```

```
User input:   hello⏎
stdin buffer: 'h' 'e' 'l' 'l' 'o' '\n' '\0'
```

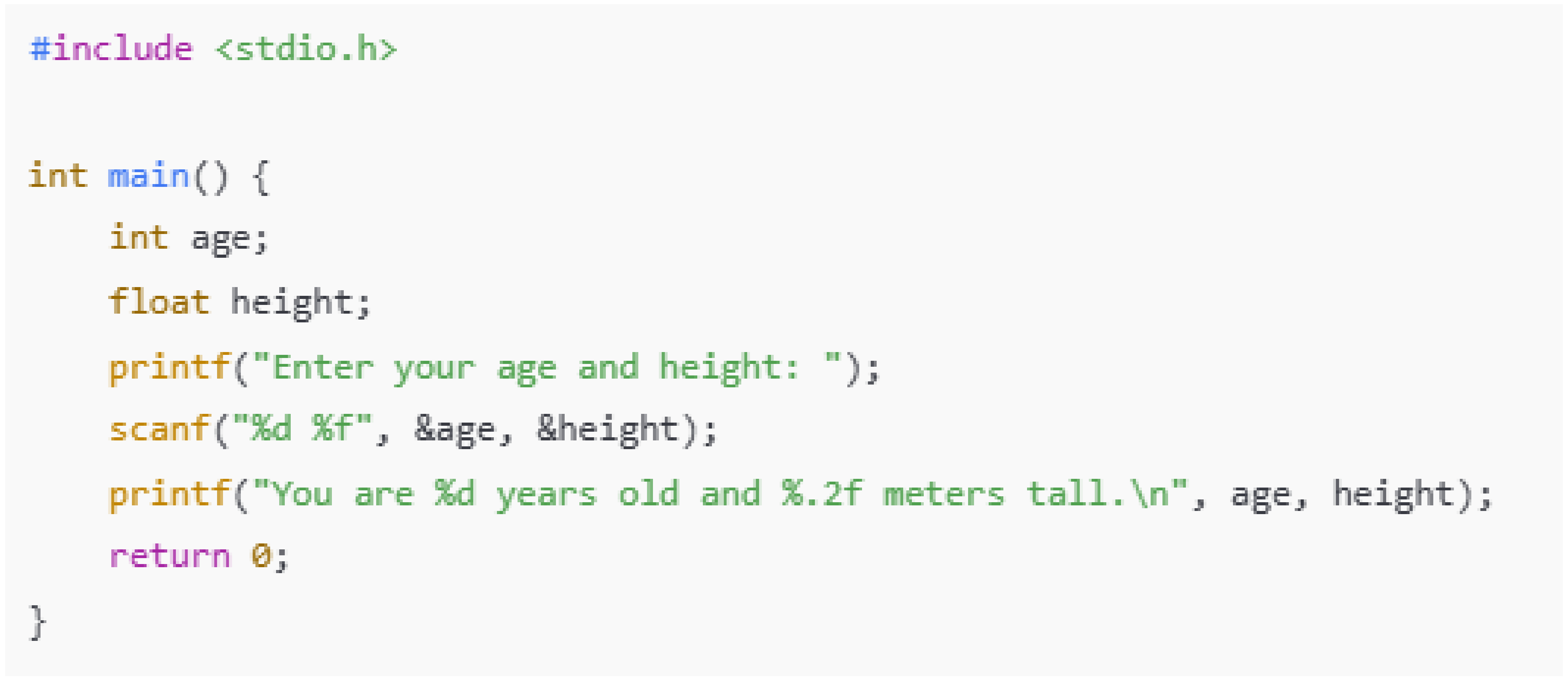
Standard Input (scanf)

A I & B i g
D a t a

1. Syntax of **scanf**()

```
#include <stdio.h>
int scanf(const char *format, ...);
```

- scanf reads formatted input from stdin (usually the keyboard).
- It requires format specifiers to determine the type of input.
- It stops reading when encountering whitespace (spaces, tabs, newlines, etc.).
- it remains ‘\n’ in buffer. To avoid → getchar();



```c
#include <stdio.h>

int main() {
    int age;
    float height;
    printf("Enter your age and height: ");
    scanf("%d %f", &age, &height);
    printf("You are %d years old and %.2f meters tall.\n", age, height);
    return 0;
}
```

Practice

Standard Input (scanf)

2. Key Characteristics of **scanf**()

- Can read multiple values at once.
- Requires the address-of operator (&) for non-string variables.
- Stops reading at the first whitespace character (space, tab, or newline).
- Can cause buffer issues if not used carefully (e.g., failing to handle newline characters properly).

If the user types hello⏎:

- The OS puts hello\n in the stdin buffer.
- scanf("%s", str) reads hello and stops at the newline.
- The \n stays in the buffer.

Standard Input (scanf)

2. Key Characteristics of **scanf**()

```
int a, b;
scanf("%d%d", &a, &b);   // works the same as "%d %d“
```

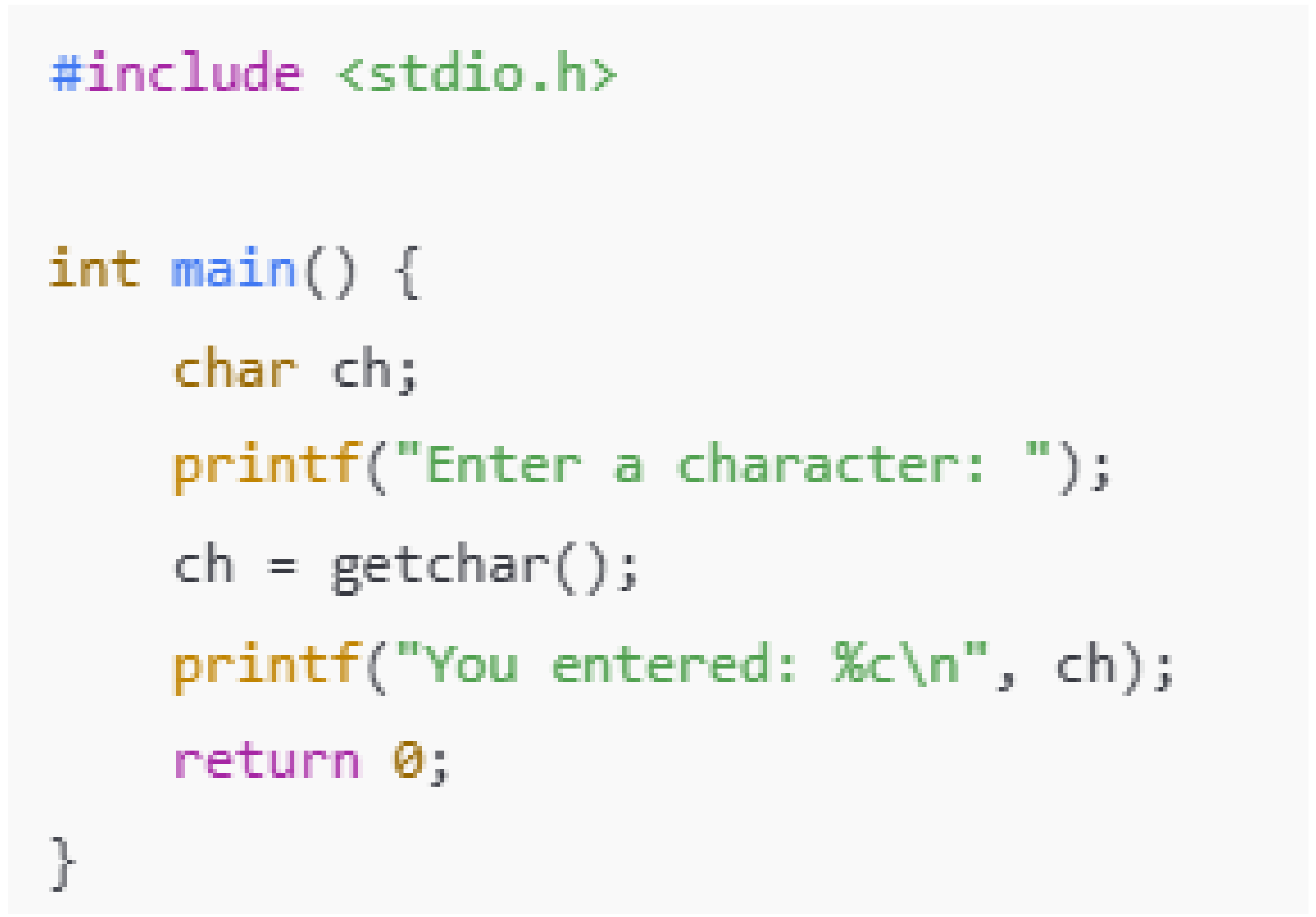
Input: 1 2 (multiple spaces or tabs or newline)
  - First %d → 1
  - Whitespace skipped automatically
  - Second %d → 2

Standard Input (getchar)

1. Syntax of **getchar**()

```
#include <stdio.h>
int getchar(void);
```

- getchar reads a single character from stdin.
- It includes whitespace characters like spaces and newlines.
- Returns the character as an unsigned char (cast to int) or EOF on error.



```
#include <stdio.h>

int main() {
    char ch;
    printf("Enter a character: ");
    ch = getchar();
    printf("You entered: %c\n", ch);
    return 0;
}
```

Practice

Standard Input (Handling Newline (`\n`) Issues)

AI & Big Data

problem: Buffer Retains `\n`

If `getchar` is used after `scanf`, it may read the leftover newline (`\n`).

* Find the problem below

```
#include <stdio.h>

int main() {
    int num;
    char ch;

    printf("Enter a number: ");
    scanf("%d", &num);  // Reads a number

    printf("Enter a character: ");
    ch = getchar();  // Problem: This reads the newline ('\n') from the buffer!

    printf("Number: %d, Character: %c\n", num, ch);
    return 0;
}
```

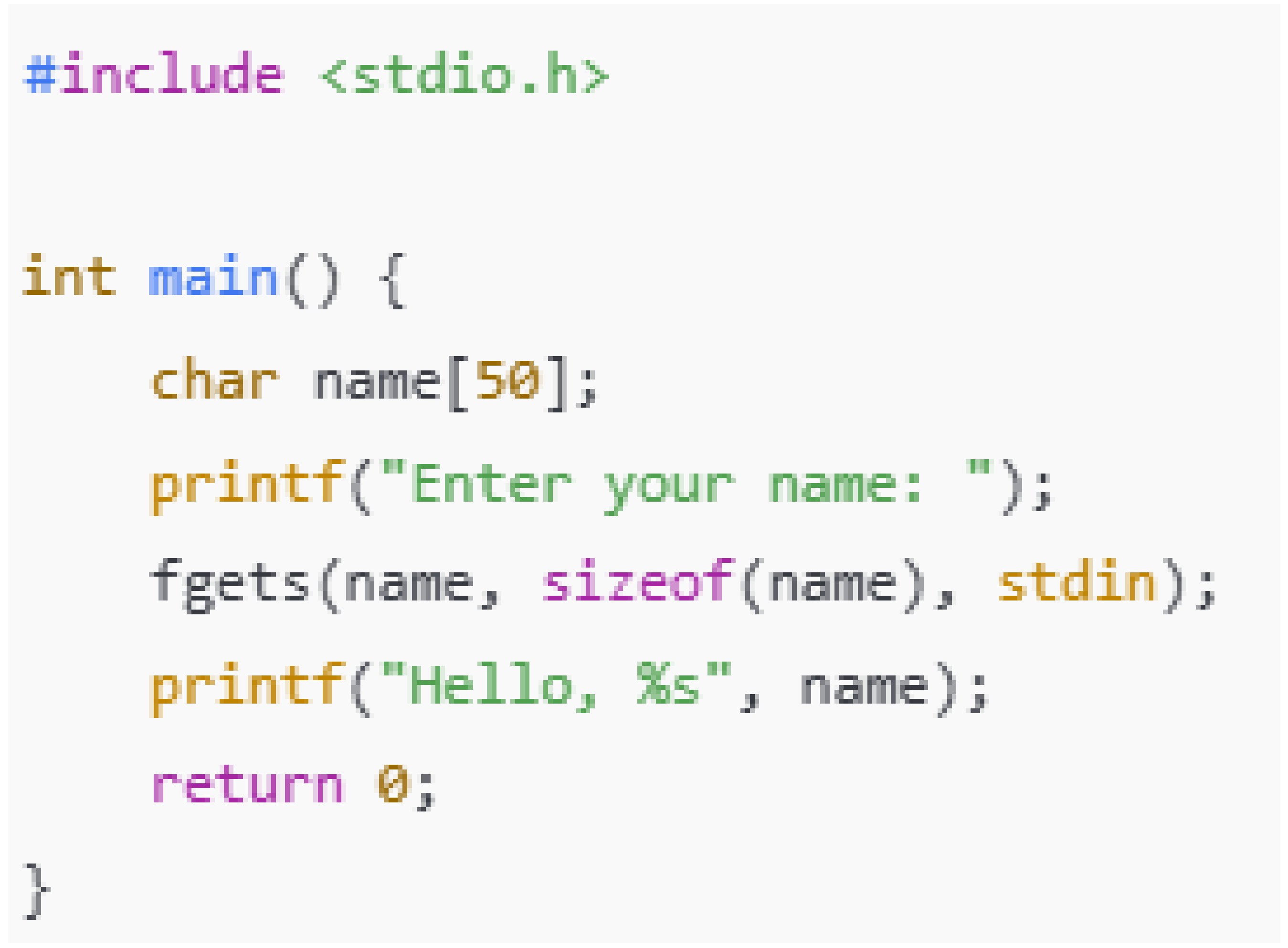
Find the problems

Standard Input (fgets)

1. Syntax of **fgets**()

#include <stdio.h>
char *fgets(char *str, int n, FILE *stream);

- fgets reads a whole line from the input (up to n-1 characters).
- It includes spaces and stops at a newline (\n).
- It prevents buffer overflow by specifying the maximum number of characters.
- It includes ‘\n’ → hello\n\0



```c
#include <stdio.h>

int main() {
    char name[50];
    printf("Enter your name: ");
    fgets(name, sizeof(name), stdin);
    printf("Hello, %s", name);
    return 0;
}
```

Practice

AI & Big Data

Standard Input (fgets)

2. Key Characteristics of **fgets**()

- Reads a full line, including spaces.

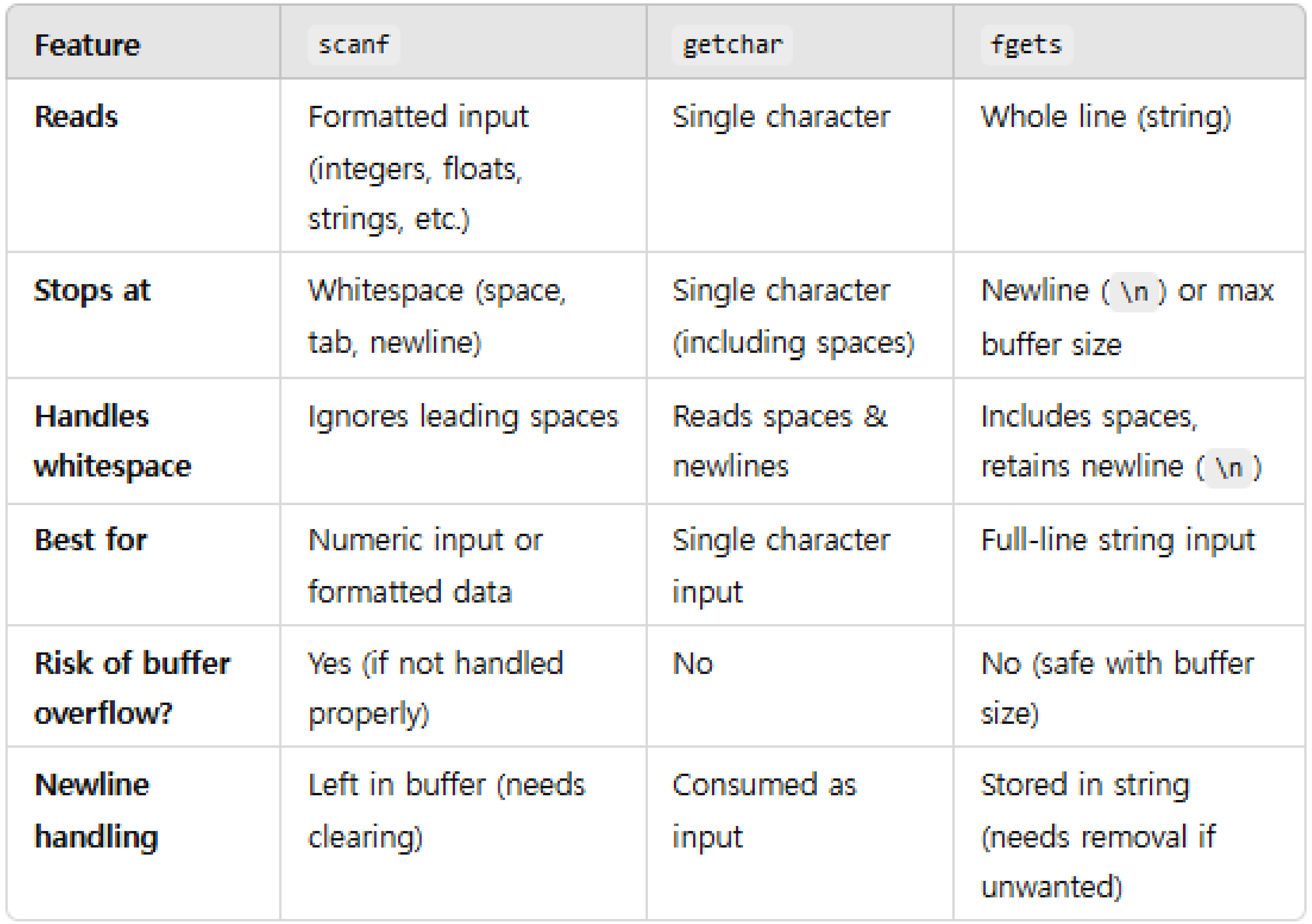
- Stops when newline (\n) or buffer limit (n-1 characters) is reached.

- Unlike scanf, it does not skip spaces.

- Adds a newline character (\n) if the user presses Enter.

Standard Input

Comparison of scanf, getchar, fgets

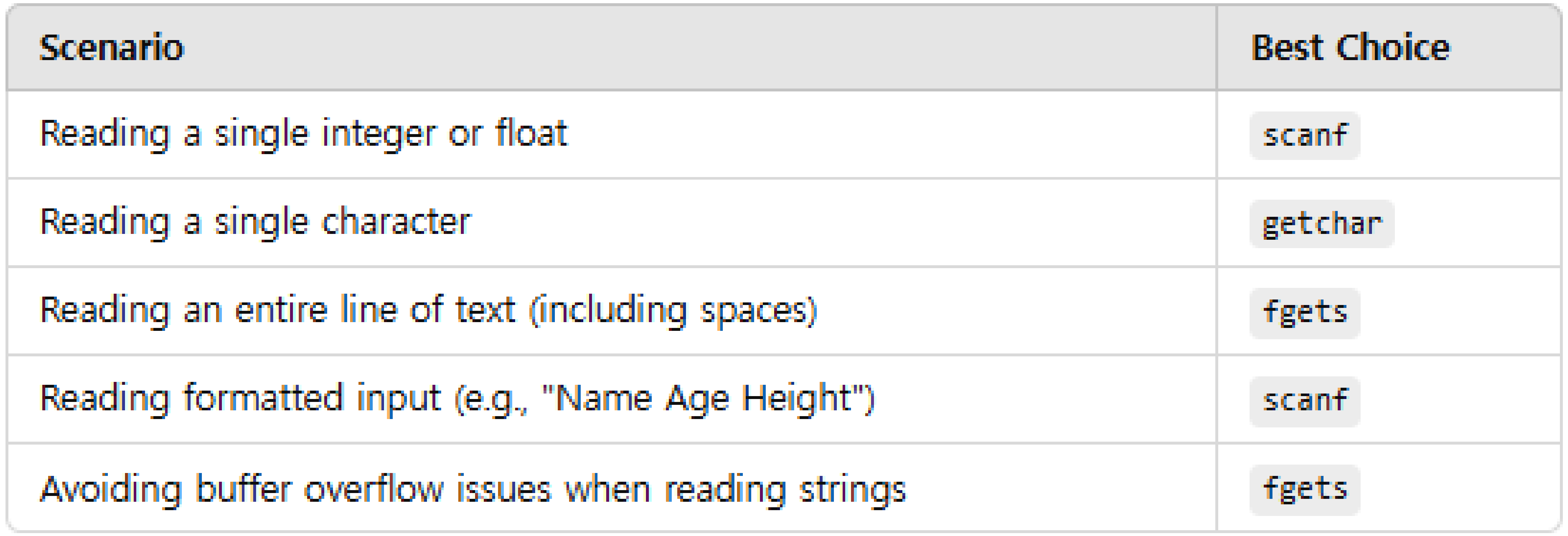


Feature	`scanf`	`getchar`	`fgets`
Reads	Formatted input (integers, floats, strings, etc.)	Single character	Whole line (string)
Stops at	Whitespace (space, tab, newline)	Single character (including spaces)	Newline (`\n`) or max buffer size
Handles whitespace	Ignores leading spaces	Reads spaces & newlines	Includes spaces, retains newline (`\n`)
Best for	Numeric input or formatted data	Single character input	Full-line string input
Risk of buffer overflow?	Yes (if not handled properly)	No	No (safe with buffer size)
Newline handling	Left in buffer (needs clearing)	Consumed as input	Stored in string (needs removal if unwanted)

Standard Input

When to use which?

-

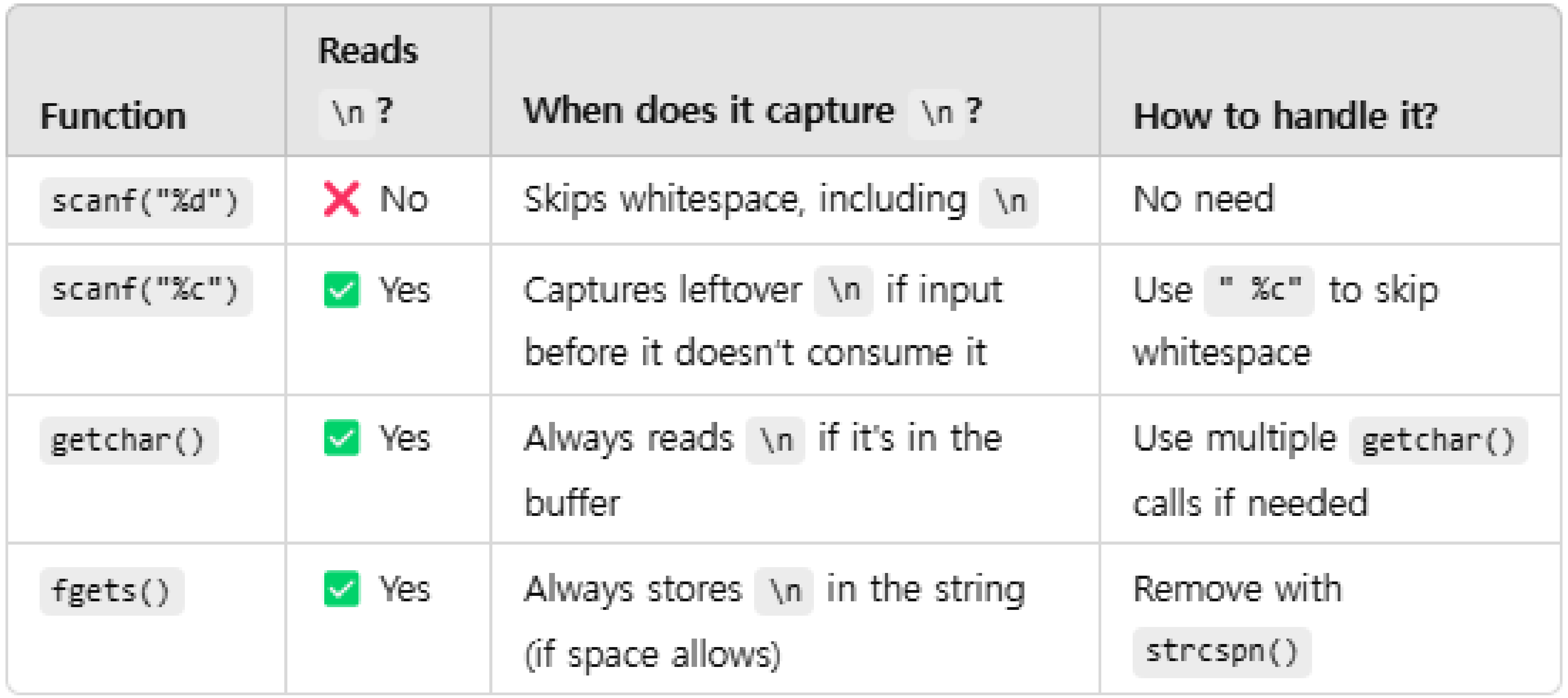


Scenario	Best Choice
Reading a single integer or float	`scanf`
Reading a single character	`getchar`
Reading an entire line of text (including spaces)	`fgets`
Reading formatted input (e.g., "Name Age Height")	`scanf`
Avoiding buffer overflow issues when reading strings	`fgets`

Standard Input

How each function handles Enter (\n)?

-



Function	Reads `\n` ?	When does it capture `\n` ?	How to handle it?
`scanf("%d")`	❌ No	Skips whitespace, including `\n`	No need
`scanf("%c")`	✅ Yes	Captures leftover `\n` if input before it doesn't consume it	Use `" %c"` to skip whitespace
`getchar()`	✅ Yes	Always reads `\n` if it's in the buffer	Use multiple `getchar()` calls if needed
`fgets()`	✅ Yes	Always stores `\n` in the string (if space allows)	Remove with `strcspn()`

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)

! (Exclamation mark) NOT operator

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)

See you next week!
DO NOT miss the classes

miricanvas.com