2. Input/Output Operations

2.1 Output Operations

2.1.1 Basic Output - printf()

Function Signature:

```
int printf(const char *format, ...);
```

Python vs C Comparison:

Python	C
`print("Hello")`	`printf("Hello\n");`
`print("Value:", x)`	`printf("Value: %d\n", x);`
`print(f"x = {x}")`	`printf("x = %d\n", x);`

Keep in mind that `print()` in python automatically creates a new line by default

2.1.2 Format Specifiers

Data Type	Format Specifier	Example
int	`%d` or `%i`	`printf("%d", 42);`
float	`%f`	`printf("%.2f", 3.14);`
double	`%lf`	`printf("%.2lf", 3.14159);`
char	`%c`	`printf("%c", 'A');`
string	`%s`	`printf("%s", "Hello");`
hexadecimal	`%x` or `%X`	`printf("%x", 255);`
unsigned int	`%u`	`printf("%u", 42u);`

2.1.3 Advanced printf() Features

Width and Precision:

```
printf("%5d", 42);        // Right-aligned in 5 characters: "   42"
printf("%-5d", 42);       // Left-aligned in 5 characters: "42   "
printf("%05d", 42);       // Zero-padded: "00042"
```

```
printf("%.2f", 3.14159);  // 2 decimal places: "3.14"
printf("%8.2f", 3.14159); // 8 characters, 2 decimals: "    3.14"
```

2.1.4 Escape Characters

Escape characters are special character sequences that represent characters that are difficult or impossible to type directly. They start with a backslash (`\`).

Common Escape Characters:

Escape Sequence	Character	Description	Example
`\n`	Newline	Moves cursor to next line	`printf("Line 1\nLine 2");`
`\t`	Tab	Horizontal tab (8 spaces)	`printf("Name:\tJohn");`
`\"`	Double Quote	Literal double quote	`printf("He said \"Hello\"");`
`\'`	Single Quote	Literal single quote	`printf("It\'s working");`
`\\`	Backslash	Literal backslash	`printf("Path: C:\\Program Files");`
`\r`	Carriage Return	Return to beginning of line	`printf("Loading\rDone");`
`\b`	Backspace	Move cursor back one position	`printf("ABC\bD");` → "ABD"
`\0`	Null Character	String terminator	`char str[] = "Hi\0lo";`
`\a`	Alert (Bell)	System beep/alert sound	`printf("\aError!");`
`\f`	Form Feed	Page break	`printf("Page 1\fPage 2");`
`\v`	Vertical Tab	Vertical tab	`printf("Line 1\vLine 2");`

Python vs C Escape Characters:

Purpose	Python	C
New line	`print("Line 1\nLine 2")`	`printf("Line 1\nLine 2");`
Tab spacing	`print("Name:\tAge")`	`printf("Name:\tAge");`
Quote in string	`print("She said \"Hi\"")`	`printf("She said \"Hi\"");`
Backslash	`print("C:\\folder")`	`printf("C:\\\\folder");`

Practical Examples:

```
// Creating formatted output with escape characters
printf("Student Information:\n");
printf("Name:\t\tJohn Doe\n");
printf("Age:\t\t20\n");
```

```
printf("GPA:\t\t3.75\n");

// Output:
// Student Information:
// Name:□□John Doe
// Age:□□20
// GPA:□□3.75
```

```
// Using quotes within strings
printf("The teacher said, \"Programming is fun!\"\n");
// Output: The teacher said, "Programming is fun!"

// File paths (especially important for Windows)
printf("Save file to: C:\\Documents\\Programs\\myfile.txt\n");
// Output: Save file to: C:\Documents\Programs\myfile.txt
```

Important Notes:

- In C strings, `\0` (null character) automatically terminates the string
- When counting string length, `\n`, `\t`, etc. each count as ONE character
- Escape characters work in both printf() format strings and character/string literals

2.2 Input Operations

2.2.1 Basic Input - scanf()

Function Signature:

```
int scanf(const char *format, ...);
```

Python vs C Comparison:

Python	C
`x = int(input())`	`scanf("%d", &x);`
`name = input()`	`scanf("%s", name);`
`x = float(input())`	`scanf("%f", &x);`

2.2.2 Important scanf() Considerations

Address Operator (&):

- Most variables need `&` before the variable name

- Exception: strings (character arrays) don't need `&`

```
int age;
char name[50];
float height;

scanf("%d", &age);     // & required for int
scanf("%s", name);     // & NOT needed for string
scanf("%f", &height);  // & required for float
```

Input Buffer Issues and Whitespace Handling:

The Whitespace Problem: When you press Enter after typing input, scanf() reads the data but leaves the newline character (`\n`) in the input buffer. This can cause problems with subsequent input operations.

```
// Problematic code:
int num;
char ch;

printf("Enter a number: ");
scanf("%d", &num);              // User types "5" and presses Enter
                                // Buffer now contains: \n (leftover newline)
printf("Enter a character: ");
scanf("%c", &ch);               // This reads the leftover \n, not user input!
printf("Character: %c\n", ch);
// Outputs:
// Character: (it shows nothing because it prints a newline)
```

What happens step by step:

1. User types "5" and presses Enter → Input buffer: `5\n`
2. `scanf("%d", &num)` reads "5" → Buffer remaining: `\n`
3. `scanf("%c", &ch)` immediately reads the leftover `\n`
4. Program doesn't wait for new character input

Solutions:

Solution 1: Space before %c

```
int num;
char ch;
```

```
printf("Enter a number: ");
scanf("%d", &num);
printf("Enter a character: ");
scanf(" %c", &ch);  // Space before %c consumes all whitespace (spaces, tabs, newlines)
```

Solution 2: Explicit buffer clearing

```
int num;
char ch;

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

// Clear the input buffer
while (getchar() != '\n');  // Read and discard until newline

printf("Enter a character: ");
scanf("%c", &ch);
```

Solution 3: Using getchar() to consume newline

```
int num;
char ch;

printf("Enter a number: ");
scanf("%d", &num);
getchar();  // Consume the leftover newline

printf("Enter a character: ");
scanf("%c", &ch);
```

Whitespace Characters in C:

- `\n` (newline) - ASCII 10
- `\t` (tab) - ASCII 9
- `\r` (carriage return) - ASCII 13
- ` ` (space) - ASCII 32
- `\f` (form feed) - ASCII 12
- `\v` (vertical tab) - ASCII 11

Important scanf() Whitespace Rules:

- `%d`, `%f`, `%s` automatically skip leading whitespace
- `%c` does NOT skip whitespace (reads exactly one character)
- Adding a space in format string (`%c`) makes scanf() skip whitespace
- `%[^\n]` does not skip leading whitespace but stops at newline

Advanced scanf() Format Specifiers:

1. Character Set Specifiers `[...]`:

```
char name[50];

// Read only alphabetic characters
scanf("%[a-zA-Z]", name);

// Read everything except newline
scanf("%[^\n]", name);  // Reads entire line including spaces

// Read only digits
scanf("%[0-9]", name);

// Read only vowels
scanf("%[aeiouAEIOU]", name);
```

2. Excluding Character Sets `[^...]`:

```
char input[100];

// Read everything EXCEPT newline (gets full line with spaces)
scanf("%[^\n]", input);

// Read everything EXCEPT spaces and tabs
scanf("%[^ \t]", input);

// Read everything EXCEPT digits
scanf("%[^0-9]", input);

// Read until comma is encountered
scanf("%[^,]", input);
```

3. Width Specifiers:

```
char buffer[10];

// Read maximum 9 characters (leaving room for \0)
scanf("%9s", buffer);

// Read exactly 5 characters
scanf("%5c", buffer);
```

4. Practical Examples:

```
// Example 1: Reading full name (including spaces)
char full_name[100];
printf("Enter your full name: ");
scanf(" %[^\n]", full_name);  // Space before % consumes previous newline

// Example 2: Reading until specific delimiter
char email[50];
printf("Enter email: ");
scanf("%[^@]", email);  // Read until @ symbol

// Example 3: Input validation
char grade[10];
printf("Enter grade (A, B, C, D, F): ");
scanf("%[ABCDFabcdf]", grade);  // Only accept valid grades
```

5. Combining Multiple Inputs:

```
int day, month, year;
char separator;

// Reading date in format: dd/mm/yyyy or dd-mm-yyyy
printf("Enter date (dd/mm/yyyy or dd-mm-yyyy): ");
scanf("%d%c%d%c%d", &day, &separator, &month, &separator, &year);

// Alternative: reading with specific separators
printf("Enter date (dd/mm/yyyy): ");
scanf("%d/%d/%d", &day, &month, &year);
```

2.2.3 Alternative Input Methods

getchar() and putchar():

```
char ch;
ch = getchar();  // Read single character
putchar(ch);     // Output single character
```

fgets() for Safe String Input:

```
char name[50];
printf("Enter your name: ");
fgets(name, sizeof(name), stdin);
```

Revision #2
Created 2025-09-01 03:32:15 UTC by DS
Updated 2025-09-01 03:46:50 UTC by DS