C Fundamentals & Operators
🧱Data Types & Basics

- **Char Range:**
  - `signed char`: -128 to +127.
  - `unsigned char`: 0 to 255.
- **Variable Naming:** Must start with a letter or underscore (`_`). Case sensitive. Cannot contain spaces or special characters (`@`, `#`).

🔥 Vital Exam Points

- **The Modulo Trap:** The `%` operator **cannot** be used with `float` or `double`. It only works on integers. `5.5 % 2` causes a **Compiler Error**.
- **Logical Short Circuit:**
  - In `(A && B)`, if A is `False`, B is **never** executed.
  - In `(A || B)`, if A is `True`, B is **never** executed.
- **Sizeof Operator:** `sizeof` is an operator, not a function. It is calculated at **Compile Time**, so `sizeof(i++)` will **not** increment `i` because the code inside isn't actually run, just measured.

-----2. Control Flow Statements 🚦 Loops & Decisions

- **Switch Rules:** You can only switch on **Integers** (`int`, `char`, `enum`). You **cannot** switch on `float`, `double`, or strings.
- **Default:** The `default` case can be placed **anywhere** in the switch block, not just at the end.

🔥 Vital Exam Points

- **The Semicolon Killer:** `if(condition);` or `for(...);` with a semicolon immediately after means the loop/condition body is **empty**. The code following it runs unconditionally (or once the empty loop finishes).
- **Dangling Else:** An `else` always pairs with the **nearest** unmatched `if`.

```
if (A)
  if (B) stmt;
else stmt; // This belongs to "if (B)", NOT "if (A)"!
```

-----3. Arrays & Strings 🔡Memory Layout

- **Array Name:** `arr` is a **constant** pointer. You can do `ptr = arr`, but you **cannot** do `arr++` or `arr = p`.

🔥 Vital Exam Points

- **String Literals vs. Arrays:**
  - `char *s = "Hello";` -> Stored in **Read-Only** memory. `s[0] = 'h'` causes a **Segmentation Fault** (Crash).

    - `char s[] = "Hello";` -> Stored on **Stack**. `s[0] = 'h'` is **Valid**.
- **Scanf Issue:** `scanf("%s", str)` stops reading at the first **space**. To read a full line with spaces, use `gets(str)` (dangerous) or `fgets(str, size, stdin)` (safe).

-----4. Functions & Pointers 🧠Pointers & Dynamic Memory

- **Pointer Scale:** `ptr + 1` adds `sizeof(type)` bytes to the address.
- **Void Pointer:** `void *p`. Can hold any address but **cannot** be dereferenced (`*p`) or used in arithmetic (`p++`) without casting.

🔥 Vital Exam Points

- **Dangling Pointer:** A pointer pointing to memory that has been freed. Always set `ptr = NULL` after calling `free(ptr)`.
- **Memory Leak:** Occurs if you allocate with `malloc` but lose the pointer before calling `free`.
- **Return Local Pointer:** Never return the address of a local variable (`int x`) from a function. That memory dies when the function ends, leading to undefined behavior.

-----5. Derived Data Types & File Handling 📂Structures & Unions

- **Bit Fields:** You can define specific bit sizes for struct members to save space (e.g., `int flag : 1;`).
- **Self-Referential Structs:** Used for Linked Lists (`struct Node *next;`).

File Handling

- **`fseek(fp, offset, origin)`:** Moves the cursor.
    - `SEEK_SET` (Start)
    - `SEEK_CUR` (Current)
    - `SEEK_END` (End)
- **`ftell(fp)`:** Returns the current cursor position (in bytes).
- **`rewind(fp)`:** Moves cursor back to the start.

🔥 Vital Exam Points

- **Union Overwrite:** If you write to one union member, **all** other members act as if they contain garbage (or the binary representation of the new data).
- **Macro Side Effects:** Macros copy-paste arguments. `SQUARE(i++)` becomes `(i++) * (i++)`, incrementing `i` twice. Always prefer `inline` functions or use macros cautiously.