---

1. Pointers & Arrays (The Most Important Topic)

Short Note: A pointer stores the memory address of another variable. The array name itself is a constant pointer to the first element.

Crucial Exam Trick:

- `*ptr` = Value at address.
- `ptr + 1` = Skips **sizeof(data_type)** bytes forward. (It doesn't just add 1 to the address number).

Example:

```C
int arr[] = {10, 20, 30, 40};
int *p = arr;
p++; // Moves to next integer (20)
printf("%d", *p);
```

- **Output:** `20`
- **Trick:** If you did `*p++`, it treats it as `*(p++)` (returns value, then increments pointer). If you did `(*p)++`, it increments the *value* (10 becomes 11).

---

2. Preprocessor Macros (`#define`)

Short Note: Macros are **text substitutions** that happen *before* compilation. They do not follow math rules (BODMAS) unless you use parentheses.

The "SQUARE" Trap (Repeated 5+ times):

```C
#define SQR(x) x*x
int main() {
    int a = 3;
    printf("%d", SQR(a+2));
}
```

- **Logic:** You might think $(3+2)^2 = 25$.
- **Reality:** The compiler expands it literally: `3 + 2 * 3 + 2`.
- **Calculation:** $3 + (2*3) + 2 = 3 + 6 + 2 = 11$.
- **Output:** `11` (Not 25).

---

3. Increment/Decrement Operators

Short Note:

- `i++` (Post-increment): Use value first, then increase.
- `++i` (Pre-increment): Increase value first, then use.

Example:

```
C

int i = 5;
int x = i++ + ++i;
printf("%d", x);
```

- **Execution:**
  1. `i++`: Use 5. (Current `i` becomes 6).
  2. `+`: Operator.
  3. `++i`: Increment `i` (6 becomes 7). Use 7.
  4. `x = 5 + 7 = 12`.
- **Output:** `12`

---

4. Storage Classes (`static`)

Short Note:

- **`auto`**: Default local variable. Dies when function ends.
- **`static`**: Retains value between function calls. Dies only when program ends.

Example:

```
C

void count() {
    static int c = 0;
    c++;
    printf("%d ", c);
}
int main() {
    count(); count();
}
```

- **Output:** `1 2`
- **Reason:** The variable `c` is **not** re-initialized to 0 in the second call. It remembers it was 1.

---

5. Structures vs. Unions

Short Note:

- **Structure (`struct`):** Total size = Sum of all members (plus padding).
- **Union (`union`):** Total size = Size of the **largest** member only. All members share the same memory.

Example:

```
C

union Data {
    int i;   // 4 bytes
    char c;  // 1 byte
};
// Size of union is 4 bytes (largest member).
// If you change 'i', 'c' gets corrupted/changed automatically.
```

---

6. String Handling & `sizeof` vs `strlen`

Short Note:

- **`sizeof()`**: Operator. Returns actual memory allocated (including null terminator `\0`).
- **`strlen()`**: Function. Returns length of string (excluding `\0`).

Example:

```
C

char s[] = "CDAC";
printf("%d %d", sizeof(s), strlen(s));
```

- **Output:** `5 4`
- **Explanation:** "CDAC" is 'C','D','A','C','\0'. Size is 5. Length is 4.

---

7. Recursion

Short Note: A function calling itself. You must identify the **Base Case** (stop condition).

Exam Strategy:

Don't guess. Draw a "Stack" on your rough paper.

```
C

void fun(int n) {
    if (n == 0) return;
    printf("%d", n);
```

```
    fun(n-1);
    printf("%d", n);
}
// Call: fun(3)
```

- **Tracing:**
    - 3 -> print 3 -> call fun(2)
        - 2 -> print 2 -> call fun(1)
            - 1 -> print 1 -> call fun(0) -> returns
            - (resume fun 1) -> print 1
        - (resume fun 2) -> print 2
    - (resume fun 3) -> print 3
- **Output:** 321123

---

8. Bitwise Operators (The Scary Ones)

Short Note:

- & (AND), | (OR), ^ (XOR), << (Left Shift), >> (Right Shift).
- **Left Shift (x << n):** Multiplies x by 2^n.
- **Right Shift (x >> n):** Divides x by 2^n.

Example:

```
C

int a = 10; // Binary 1010
int b = a << 1;
printf("%d", b);
```

- **Logic:** $10 \times 2^1 = 20$.
- **Output:** 20

Top 3 "Gotcha" Questions for Exam

Q1: The `break` vs `continue`

- **`break`**: Exits the loop immediately.
- **`continue`**: Skips the current iteration and goes to the next one.

Q2: The "Dangling Else"

```
C

if (a)
    if (b) x++;
```

```
else y++;
```

- **Rule:** The `else` belongs to the **nearest** `if`. Here, `else` belongs to `if(b)`, not `if(a)`.

Q3: Floating Point Comparison

```
C

float f = 0.1;
if (f == 0.1) printf("Yes");
else printf("No");
```

- **Output:** `No`
- **Reason:** `0.1` is a `double` by default in C. `float` 0.1 is slightly less precise than `double` 0.1.
- *Fix:* Use `0.1f`.
