Memory and Valgrind Refresher - Cheat Sheet

Key Rule: To modify a variable (including pointers) inside a function, you must pass its address (pointer). Otherwise, you are only modifying a copy.

Double Pointers: To modify a pointer in a function, you need to pass a pointer to that pointer (double pointer `**`).

Key Concepts:
*   **Memory Address:** The value of a pointer is a memory address.
*   **NULL:** A pointer can be `NULL` (usually 0), indicating it doesn't point to anything.
*   **Memory Allocation:** Use `malloc()` (or `calloc()`, `realloc()`) to dynamically allocate memory.
*   **Memory Deallocation:** Use `free()` to deallocate dynamically allocated memory to prevent memory leaks.

Formulas/Syntax Examples:

*   **Modifying an `int`:**
    ```c
    void add2(int* val) {
        *val = *val + 2;
    }
    int n = 2;
    add2(&n); // Pass the address of n
    ```
*   **Modifying a pointer:**
    ```c
    void allocate(int** p, int len) {
        *p = malloc(sizeof(int) * len);
    }
    int* array = NULL;
    allocate(&array, arrLen); // Pass the address of array
    ```

Valgrind:

*   **Definition:** A memory debugging tool for detecting memory leaks and

other memory-related errors.
*   **Availability:** Unix/Linux-based systems (including Ubuntu shell on Windows and macOS up to 10.11).
*   **Usage:** `valgrind ./your_program`
*   **Compilation:** Compile with the `-g` flag: `gcc your_file.c -o your_program -g`
*   **Key Output Sections:**
    *   `HEAP SUMMARY`
    *   `LEAK SUMMARY`
    *   `ERROR SUMMARY`
*   **Leak Types (LEAK SUMMARY):**
    *   `definitely lost`: Memory is leaked; fix immediately.
    *   `indirectly lost`: Memory is leaked through pointer-based structures; fix the `definitely lost` leaks first.
    *   `possibly lost`: Memory may be leaked, or due to unusual pointer usage; investigate.
*   **Common Errors Detected:**
    *   Using an uninitialized variable.
    *   Writing to memory that was not allocated (e.g., buffer overflow).

Common Tips/Pitfalls:

*   **Memory Leaks:** Always `free()` memory allocated with `malloc()`, `calloc()`, or `realloc()`.
*   **Library Functions:** Be aware that some library functions (e.g., `strdup()`, `toString()`) allocate memory that *you* are responsible for freeing.
*   **Uninitialized Variables:** Always initialize variables before using them. Valgrind can help detect this.
*   **Insufficient Memory Allocation:** Ensure you allocate enough memory for your data.
*   **Double Pointers:** Be careful when dereferencing double pointers, as they involve accessing the address stored in a pointer.
*   **Moral of the story:** to modify a thing in a function, we must pass its address - otherwise we're modifying a copy!