C Cheatsheet

Basic Structure and Compilation

C is a general-purpose, procedural programming language that provides low-level access to memory.

```c
#include <stdio.h> // Standard I/O library

int main() {
    printf("Hello, World!\n"); // Print formatted output
    return 0; // Successful execution
}
```

- Compilation (GCC):
  - `gcc -o hello hello.c`: Compile and generate executable.
  - `./hello`: Run the executable.
- Preprocessor: Commands starting with `#` are processed before compilation (e.g., `#include`, `#define`).

Data Types and Variables

- Basic Types: `int` (integers), `float` (floating-point), `double` (long-float), `char` (character).
- Data Sizes: `short`, `long`, `unsigned`, `signed`.
- Constants:
  - `#define PI 3.14159`: Preprocessor macro.
  - `const double PI = 3.14159`: Read-only variable.

Operators

- Arithmetic: `+`, `-`, `*`, `/`, `%` (modulo), `++`, `--`.
- Relational: `==`, `!=`, `<`, `>`, `<=`, `>=`.
- Logical: `&&` (AND), `||` (OR), `!` (NOT).
- Bitwise: `&`, `|`, `^`, `~`, `<<`, `>>`.

Control Flow

- `if-else` / `switch`:

```c
if (condition) { ... } else { ... }
```

- Loops:
  - `for (int i = 0; i < 10; i++) { ... }`
  - `while (condition) { ... }`
  - `do { ... } while (condition);`

Functions

```c
return_type
function_name(parameter_list) {
    // Logic
    return value;
}
```

- Prototype: Declare function signature before usage if defined later.

Arrays and Strings

- Array: `int arr[5] = {1, 2, 3, 4, 5};`
- String (Null-terminated character array):
  - `char str[] = "Hello";` (Ends with `\0`)
  - String library: `#include <string.h>` (`strlen`, `strcpy`, `strcmp`, `strcat`).

Pointers and Memory Management

- Pointer (`*`): A variable that stores the address of another variable.
  - `int x = 10; int *p = &x;`
  - Dereferece: `*p` gives the value 10.
- Dynamic Memory Allocation (`stdlib.h`):
  - `malloc(size)`: Allocate memory on the heap.
  - `calloc(n, size)`: Allocate and zero-initialize.
  - `free(pointer)`: Deallocate heap memory (crucial to avoid memory leaks).

Structures and Enums

- Struct: `struct Person { char name[50]; int age; };`
- Typedef: `typedef struct Person Person;` (create an alias).
- Enum: `enum Color { RED, GREEN, BLUE };`

File I/O (`stdio.h`)

- `FILE *fp = fopen("file.txt", "r");`
- `fprintf`, `fscanf`: Formatted file I/O.
- `fgets`, `fputs`: String file I/O.
- `fclose(fp);`

Google Style Guide

Basic Rules

- Use 2-space indentation (no tabs).
- Limit line length to 80 characters.
- Use `snake_case` for names (functions, variables, files).
- Prefer `const` and `enum` over `#define` for constants.

Naming Conventions

- Variables and Functions: `lower_snake_case`
- Types (structs, typedefs): `LowerSnakeCase` or `lower_snake_case` (consistent within project).
- Constants and Macros: `UPPER_SNAKE_CASE`

Formatting

- Braces: Place the opening brace on the same line as the statement.
- Pointer Alignment: `int *x` (space before the asterisk).

Programming Practices

- Always initialize variables.
- Check return values of heap allocation and file I/O.
- Use explicit sizes from `<stdint.h>` (e.g., `int32_t`, `uint64_t`) for cross-platform stability.
- Handle all `switch` cases or provide a `default` case.

Documentation

- Use `/* ... */` for comments.
- Provide file-level and function-level comments explaining behavior, parameters, and return values.