C Programming Language Cheatsheet

A concise reference for the C programming language, covering syntax, data types, standard library functions, and memory management.

Basic Syntax and Data Types

Program Structure

A C program consists of preprocessor directives, type definitions, function declarations, and function definitions. The `main` function is the entry point.

```c
#include <stdio.h>

int main() {
  printf("Hello, World!\n");
  return 0;
}
```

Comments can be single-line (`//`) or multi-line (`/* ... */`).

Data Types

`int`	Integer (e.g., `10` , `-5` , `0`)
`float`	Single-precision floating-point number (e.g., `3.14` , `-2.5`)
`double`	Double-precision floating-point number (e.g., `3.14159265359`)
`char`	Single character (e.g., `'A'` , `'7'` , `'\n'`)
`void`	Represents the absence of a type. Used for functions that do not return a value or for generic pointers.

Variables

Variables must be declared before use.

Syntax: `data_type variable_name;`

Example:

```c
int age;
float salary;
char initial;
```

Initialization can be done during declaration: `int age = 30;`

Use `const` keyword to declare constant variables: `const float PI = 3.14159;`

Operators and Control Flow

Operators

Arithmetic	`+` , `-` , `*` , `/` , `%` (modulus)
Assignment	`=` , `+=` , `-=` , `*=` , `/=` , `%=`
Comparison	`==` , `!=` , `>` , `<` , `>=` , `<=`
Logical	`&&` (AND), `||` (OR), `!` (NOT)
Bitwise	`&` (AND), `|` (OR), `^` (XOR), `~` (NOT), `<<` (left shift), `>>` (right shift)
Increment/Decrement	`++` , `--` (prefix and postfix)

Control Flow Statements

if-else Statement:

```c
if (condition) {
  // code to execute if condition is true
} else {
  // code to execute if condition is false
}
```

switch Statement:

```c
switch (expression) {
  case constant1:
    // code to execute if expression == constant1
    break;
  case constant2:
    // code to execute if expression == constant2
    break;
  default:
    // code to execute if no case matches
}
```

for Loop:

```c
for (initialization; condition; increment) {
  // code to execute repeatedly
}
```

while Loop:

```c
while (condition) {
  // code to execute repeatedly while condition is true
}
```

do-while Loop:

```c
do {
  // code to execute at least once
} while (condition);
```

break and continue:

`break` exits the loop or switch statement.

`continue` skips the current iteration of the loop.

Functions and Pointers

Functions

Functions are blocks of code that perform a specific task.

Syntax:

```
return_type function_name(parameter_list) {
  // function body
  return value;
}
```

Example:

```
int add(int a, int b) {
  return a + b;
}
```

Function prototypes declare the function before its definition.

Example:

```
int add(int a, int b);
```

Pointers

Declaration	`int *ptr;` (pointer to an integer)
Address-of Operator	`&variable` (returns the address of the variable)
Dereference Operator	`*ptr` (accesses the value pointed to by `ptr`)

Example	`int x = 10;`
	`int *ptr = &x;`
	`printf("%d", *ptr); // Output: 10`

Arrays and Pointers

Arrays and pointers are closely related. The name of an array is a pointer to the first element of the array.

Example:

```
int arr[5] = {1, 2, 3, 4, 5};
int *ptr = arr; // ptr points to arr[0]
printf("%d", *ptr); // Output: 1
printf("%d", arr[0]); // Output: 1
```

Pointer Arithmetic:

```
printf("%d", *(ptr + 1)); // Output: 2 (accesses arr[1])
```

Standard Library and Memory Management

Standard Library Functions (stdio.h)

`printf()`	Formatted output to the console.
`scanf()`	Formatted input from the console.
`fopen()`	Opens a file.
`fclose()`	Closes a file.
`fprintf()`	Formatted output to a file.
`fscanf()`	Formatted input from a file.

Memory Management (stdlib.h)

`malloc()`: Allocates a block of memory on the heap.

Syntax:

```
void *malloc(size_t size);
```

`calloc()`: Allocates a block of memory and initializes it to zero.

Syntax:

```
void *calloc(size_t num, size_t size);
```

`realloc()`: Resizes a previously allocated block of memory.

Syntax:

```
void *realloc(void *ptr, size_t size);
```

`free()`: Deallocates a block of memory.

Syntax:

```
void free(void *ptr);
```

Example:

```
int *arr = (int *)malloc(5 * sizeof(int));
if (arr == NULL) {
  // Handle memory allocation failure
}
// Use arr
free(arr);
arr = NULL; // Good practice to prevent dangling pointers
```