C Quick Reference

Syntax, pointers, memory management, standard library essentials

Basics

Hello World

```
#include <stdio.h>
int main(void) {
    printf("Hello, World!\n");
    return 0;
}
```

Compile & Run

```
gcc -o app main.c          # compile
gcc -Wall -Wextra -std=c17 main.c  # strict
./app                       # run
```

Comments

```
// single-line comment (C99+)
/* multi-line
   comment */
```

Data Types

Primitive Types

`char`	1 byte, character or small integer
`short`	At least 16 bits
`int`	At least 16 bits (typically 32)
`long`	At least 32 bits
`long long`	At least 64 bits (C99+)
`float`	32-bit IEEE-754
`double`	64-bit IEEE-754
`_Bool / bool`	`0` or `1` (use `<stdbool.h>` for `bool`)

Fixed-Width Types (stdint.h)

`int8_t, uint8_t`	Exact 8-bit signed / unsigned
`int16_t, uint16_t`	Exact 16-bit
`int32_t, uint32_t`	Exact 32-bit
`int64_t, uint64_t`	Exact 64-bit
`size_t`	Unsigned, result of `sizeof`

Type Casting

```
int i = (int)3.14;          // explicit cast
double d = (double)5 / 2;   // 2.5, not 2
char c = (char)65;          // 'A'
```

Control Flow

If / Else

```
if (x > 0) { printf("positive\n"); }
else if (x == 0) { printf("zero\n"); }
else { printf("negative\n"); }
```

Switch

```
switch (choice) {
    case 1:  printf("one\n"); break;
    case 2:  printf("two\n"); break;
    default: printf("other\n");
}
```

Loops

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

Jump Statements

`break`	Exit innermost loop or switch
`continue`	Skip to next iteration
`return`	Exit function with optional value
`goto label`	Jump to label (use sparingly)

Functions

Declaration & Definition

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

Function Pointers

```
int (*op)(int, int) = add;
int result = op(3, 4);       // calls add(3, 4)
typedef int (*MathFn)(int, int);
MathFn fn = add;
```

Static Functions

```
// visible only within this translation unit
static int helper(int x) {
    return x * 2;
}
```

Pointers

Pointer Basics

```
int x = 42;
int *p = &x;        // p points to x
printf("%d\n", *p); // dereference: 42
*p = 100;           // x is now 100
```

Pointer Arithmetic

```
int arr[] = {10, 20, 30};
int *p = arr;
printf("%d\n", *(p + 1));  // 20
printf("%d\n", p[2]);      // 30 (same as *(p+2))
```

Common Pointer Patterns

`int *p = NULL`	Null pointer (always initialize)
`void *`	Generic pointer (must cast to use)
`const int *p`	Pointer to constant (cannot modify value)
`int *const p`	Constant pointer (cannot reassign pointer)
`int **pp`	Pointer to pointer (double indirection)

Arrays & Strings

Arrays

```
int nums[5] = {1, 2, 3, 4, 5};
int matrix[2][3] = {{1,2,3}, {4,5,6}};
int len = sizeof(nums) / sizeof(nums[0]);
```

String Functions (string.h)

`strlen(s)`	Length (excluding null terminator)
`strcpy(dst, src)`	Copy string (unsafe, prefer `strncpy`)
`strncpy(dst, src, n)`	Copy at most n characters
`strcat(dst, src)`	Concatenate strings
`strcmp(a, b)`	Compare: 0 if equal, <0 or >0 otherwise
`strchr(s, c)`	Find first occurrence of character
`strstr(haystack, needle)`	Find substring

String Literals

```
char greeting[] = "hello";     // mutable array
const char *msg = "world";     // pointer to literal
char buf[64];
snprintf(buf, sizeof(buf), "%s %s", greeting, msg);
```

Structs

Definition & Usage

```
struct Point { double x; double y; };
struct Point p = {1.0, 2.0};
printf("(%g, %g)\n", p.x, p.y);
```

Typedef

```
typedef struct {
    char name[50];
    int age;
} Person;
Person p = {"Alice", 30};
```

Struct Pointers

```
void set_age(Person *p, int age) {
    p->age = age;    // arrow operator
}
```

Enums & Unions

```
enum Color { RED, GREEN, BLUE };
union Data { int i; float f; char c; };
// union members share the same memory
```

Memory Management

Dynamic Allocation (stdlib.h)

```
int *arr = malloc(10 * sizeof(int));
if (arr == NULL) { /* handle error */ }
arr = realloc(arr, 20 * sizeof(int));
free(arr);
arr = NULL;   // avoid dangling pointer
```

Allocation Functions

`malloc(size)`	Allocate uninitialized memory
`calloc(count, size)`	Allocate and zero-initialize
`realloc(ptr, size)`	Resize previously allocated block
`free(ptr)`	Release allocated memory

Common Pitfalls

`Memory leak`	Forgetting to `free()` allocated memory
`Double free`	Calling `free()` on same pointer twice
`Dangling pointer`	Using pointer after `free()` — set to NULL
`Buffer overflow`	Writing past allocated bounds

File I/O

Reading Files

```
FILE *f = fopen("data.txt", "r");
if (!f) { perror("open"); return 1; }
char line[256];
while (fgets(line, sizeof(line), f)) printf("%s", line);
fclose(f);
```

Writing to Files

```
FILE *f = fopen("out.txt", "w");
fprintf(f, "value: %d\n", 42);
fputs("hello\n", f);
fclose(f);
```

C Quick Reference

File Modes

`"r"`	Read (file must exist)
`"w"`	Write (truncates or creates)
`"a"`	Append (creates if needed)
`"rb", "wb"`	Binary read / write
`"r+"`	Read and write (file must exist)

Preprocessor

Directives

```
#include <stdio.h>       // system header
#include "myheader.h"    // local header
#define PI 3.14159
#define MAX(a, b) ((a) > (b) ? (a) : (b))
```

Conditional Compilation

```
#ifdef DEBUG
    printf("debug: x = %d\n", x);
#endif
#ifndef HEADER_H   /* include guard */
#define HEADER_H   /* ... */ #endif
```

Common Macros

`__FILE__`	Current source file name
`__LINE__`	Current line number
`__func__`	Current function name (C99+)
`__DATE__`	Compilation date string
`NULL`	Null pointer constant
`sizeof(x)`	Size of type or variable in bytes