C Preprocessor Cheatsheet

A comprehensive cheat sheet for the C preprocessor, covering directives, macros, conditional compilation, and more. This guide provides a quick reference for developers working with C and C++.

Directives Overview

Include Directives

`#include <file.h>`	Searches standard system directories for `file.h`.
`#include "file.h"`	Searches the current directory first, then standard system directories.
`#include MACRO`	If `MACRO` expands to `<...>` or `"..."`, the include directive is processed accordingly.

Example:

```
#define HEADER_FILE "my_header.h"
#include HEADER_FILE
```

Define and Undefine

`#define SYMBOL`	Defines a simple symbol. Commonly used for conditional compilation.
`#define SYMBOL value`	Defines a symbol with a value. The value can be any text.
`#undef SYMBOL`	Undefines a previously defined symbol.

Example:

```
#define PI 3.14159
#undef PI
```

Conditional Compilation

```
#ifdef SYMBOL
  // Code to compile if SYMBOL is defined
#endif
```

```
#ifndef SYMBOL
  // Code to compile if SYMBOL is not defined
#endif
```

```
#if expression
  // Code to compile if expression is true
#elif expression2
  // Code to compile if expression2 is true
#else
  // Code to compile if none of the above are true
#endif
```

```
#if defined(SYMBOL)
  // Code to compile if SYMBOL is defined
#endif
```

Macros in Detail

Basic Macros

`#define MACRO(arg) ((arg) * 2)`	A simple macro that doubles its argument.

Example Usage:

```
int x = 5;
int y = MACRO(x); // y will be 10
```

Important: Always enclose macro arguments in parentheses to avoid unexpected behavior due to operator precedence.

Stringification Operator (#)

`#define STRINGIFY(x) #x`	Converts a macro parameter into a string literal.

Example:

```
STRINGIFY(hello world) // expands to "hello world"
```

Token Pasting Operator (##)

`#define CONCAT(x, y) x ## y`	Concatenates two tokens into a single token.

Example:

```
CONCAT(var, 123) // creates a single token var123
```

Variadic Macros

`#define LOG(format, ...) printf(format, __VA_ARGS__)`	Creates macros that accept a variable number of arguments.

Example:

```
LOG("Error: %d", errno); // Prints an error message with the error number
```

Predefined Macros

Standard Predefined Macros

`__FILE__`	The name of the current input file, as a string literal.
`__LINE__`	The current line number in the input file, as a decimal integer.
`__DATE__`	The date of compilation, as a string literal in “Mmm dd yyyy” format.
`__TIME__`	The time of compilation, as a string literal in “hh:mm:ss” format.
`__STDC__`	Indicates that the implementation conforms to the C standard. Defined as 1.

Example Usage:

```
printf("File: %s, Line: %d\n", __FILE__, __LINE__);
```

Compiler-Specific Macros

Compilers often define their own macros (e.g., `__GNUC__` for GCC, `_MSC_VER` for Microsoft Visual C++).

These can be used for compiler-specific conditional compilation.

Example:

```
#ifdef __GNUC__
  // GCC specific code
#endif
```

Advanced Directives and Techniques

Error and Warning Directives

`#error message`	Causes the preprocessor to issue a fatal error message and halt compilation.
`#warning message`	Causes the preprocessor to issue a warning message, but compilation continues.

Example:

```
#if !defined(DEBUG)
  #error "DEBUG must be defined for this build."
#endif
```

Pragma Directive

`#pragma directive`

Specifies various compiler-specific instructions. Behavior varies between compilers.

Common Usage:

`#pragma once` - Prevents a header file from being included more than once in a single compilation unit.

Line Control

`#line number "filename"`	Resets the line number and filename reported by the compiler for subsequent code.

Example:

```
#line 100 "newfile.c"
// The next line will be reported as line 100 of newfile.c
```