# **C - Preprocessors**

The C Preprocessor is not a part of the compiler, but is a separate step in the compilation process. In simple terms, a C Preprocessor is just a text substitution tool and it instructs the compiler to do required pre-processing before the actual compilation. We'll refer to the C Preprocessor as CPP.

All preprocessor commands begin with a hash symbol (#). It must be the first nonblank character, and for readability, a preprocessor directive should begin in the first column. The following section lists down all the important preprocessor directives −

|  |  |
| --- | --- |
| **Sr.No.** | **Directive & Description** |
| 1 | #define  Substitutes a preprocessor macro. |
| 2 | #include  Inserts a particular header from another file. |
| 3 | #undef  Undefines a preprocessor macro. |
| 4 | #ifdef  Returns true if this macro is defined. |
| 5 | #ifndef  Returns true if this macro is not defined. |
| 6 | #if  Tests if a compile time condition is true. |
| 7 | #else  The alternative for #if. |
| 8 | #elif  #else and #if in one statement. |
| 9 | #endif  Ends preprocessor conditional. |
| 10 | #error  Prints error message on stderr. |
| 11 | #pragma  Issues special commands to the compiler, using a standardized method. |

## Preprocessors Examples

Analyze the following examples to understand various directives.

#define MAX\_ARRAY\_LENGTH 20

This directive tells the CPP to replace instances of MAX\_ARRAY\_LENGTH with 20. Use #define for constants to increase readability.

#include <stdio.h>

#include "myheader.h"

These directives tell the CPP to get stdio.h from System Libraries and add the text to the current source file. The next line tells CPP to get myheader.h from the local directory and add the content to the current source file.

#undef FILE\_SIZE

#define FILE\_SIZE 42

It tells the CPP to undefine existing FILE\_SIZE and define it as 42.

#ifndef MESSAGE

#define MESSAGE "You wish!"

#endif

It tells the CPP to define MESSAGE only if MESSAGE isn't already defined.

#ifdef DEBUG

/\* Your debugging statements here \*/

#endif

It tells the CPP to process the statements enclosed if DEBUG is defined. This is useful if you pass the -DDEBUG flag to the gcc compiler at the time of compilation. This will define DEBUG, so you can turn debugging on and off on the fly during compilation.

## Predefined Macros

ANSI C defines a number of macros. Although each one is available for use in programming, the predefined macros should not be directly modified.

|  |  |
| --- | --- |
| **Sr.No.** | **Macro & Description** |
| 1 | \_\_DATE\_\_  The current date as a character literal in "MMM DD YYYY" format. |
| 2 | \_\_TIME\_\_  The current time as a character literal in "HH:MM:SS" format. |
| 3 | \_\_FILE\_\_  This contains the current filename as a string literal. |
| 4 | \_\_LINE\_\_  This contains the current line number as a decimal constant. |
| 5 | \_\_STDC\_\_  Defined as 1 when the compiler complies with the ANSI standard. |

Let's try the following example −

[Live Demo](http://tpcg.io/BMJM0C)

#include <stdio.h>

int main() {

printf("File :%s\n", \_\_FILE\_\_ );

printf("Date :%s\n", \_\_DATE\_\_ );

printf("Time :%s\n", \_\_TIME\_\_ );

printf("Line :%d\n", \_\_LINE\_\_ );

printf("ANSI :%d\n", \_\_STDC\_\_ );

}

When the above code in a file test.c is compiled and executed, it produces the following result −

File :test.c

Date :Jun 2 2012

Time :03:36:24

Line :8

ANSI :1

## Preprocessor Operators

The C preprocessor offers the following operators to help create macros −

### The Macro Continuation (\) Operator

A macro is normally confined to a single line. The macro continuation operator (\) is used to continue a macro that is too long for a single line. For example −

#define message\_for(a, b) \

printf(#a " and " #b ": We love you!\n")

### The Stringize (#) Operator

The stringize or number-sign operator ( '#' ), when used within a macro definition, converts a macro parameter into a string constant. This operator may be used only in a macro having a specified argument or parameter list. For example −

[Live Demo](http://tpcg.io/qLYOKm)

#include <stdio.h>

#define message\_for(a, b) \

printf(#a " and " #b ": We love you!\n")

int main(void) {

message\_for(Carole, Debra);

return 0;

}

When the above code is compiled and executed, it produces the following result −

Carole and Debra: We love you!

### The Token Pasting (##) Operator

The token-pasting operator (##) within a macro definition combines two arguments. It permits two separate tokens in the macro definition to be joined into a single token. For example −

[Live Demo](http://tpcg.io/2ZlJsc)

#include <stdio.h>

#define tokenpaster(n) printf ("token" #n " = %d", token##n)

int main(void) {

int token34 = 40;

tokenpaster(34);

return 0;

}

When the above code is compiled and executed, it produces the following result −

token34 = 40

It happened so because this example results in the following actual output from the preprocessor −

printf ("token34 = %d", token34);

This example shows the concatenation of token##n into token34 and here we have used both stringize and token-pasting.

### The Defined() Operator

The preprocessor defined operator is used in constant expressions to determine if an identifier is defined using #define. If the specified identifier is defined, the value is true (non-zero). If the symbol is not defined, the value is false (zero). The defined operator is specified as follows −

[Live Demo](http://tpcg.io/Fwp5AX)

#include <stdio.h>

#if !defined (MESSAGE)

#define MESSAGE "You wish!"

#endif

int main(void) {

printf("Here is the message: %s\n", MESSAGE);

return 0;

}

When the above code is compiled and executed, it produces the following result −

Here is the message: You wish!

## Parameterized Macros

One of the powerful functions of the CPP is the ability to simulate functions using parameterized macros. For example, we might have some code to square a number as follows −

int square(int x) {

return x \* x;

}

We can rewrite above the code using a macro as follows −

#define square(x) ((x) \* (x))

Macros with arguments must be defined using the #define directive before they can be used. The argument list is enclosed in parentheses and must immediately follow the macro name. Spaces are not allowed between the macro name and open parenthesis. For example −

[Live Demo](http://tpcg.io/gXEa63)

#include <stdio.h>

#define MAX(x,y) ((x) > (y) ? (x) : (y))

int main(void) {

printf("Max between 20 and 10 is %d\n", MAX(10, 20));

return 0;

}

When the above code is compiled and executed, it produces the following result −

Max between 20 and 10 is 20