

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 –

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.

Let's try the following example –

```
#include <stdio.h>
```

[Live Demo](#)

```
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 –

```
#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;  
}
```

[Live Demo](#)

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 –

```
#include <stdio.h>
```

[Live Demo](#)

```
#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 –

```
#include <stdio.h>
```

[Live Demo](#)

```
#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 –

```
#include <stdio.h>
```

[Live Demo](#)

```
#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
```