

THE C PREPROCESSOR

Mayank Yadav

Department Of Computer Science and Engineering,
Dronacharya College of Engineering, Gurgaon

Abstract- In today's generation, millions are spent on developing softwares, and a number of them are made on c/c++ language. The C Preprocessor is not part of the compiler, but is a separate step in the compilation process. In simplistic terms, a C Preprocessor is just a text substitution tool and they instruct compiler to do required pre-processing before actual compilation. We'll refer to the C Preprocessor as the CPP.

Index Terms- preprocessor, compiler

I. INTRODUCTION

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

Directive	Description
#define	Substitutes a preprocessor macro
#include	Inserts a particular header from another file
#undef	Undefines a preprocessor macro
#ifdef	Returns true if this macro is defined
#ifndef	Returns true if this macro is not defined
#if	Tests if a compile time condition is true
#else	The alternative for #if
#elif	#else an #if in one statement
#endif	Ends preprocessor conditional
#error	Prints error message on stderr
#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
```

This tells the CPP to undefine existing FILE_SIZE and define it as 42.

```
#ifndef MESSAGE
#define MESSAGE "You wish!"
#endif
```

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

```
#ifdef DEBUG
/* Your debugging statements here */
#endif
```

This tells the CPP to do the process the statements enclosed if DEBUG is defined. This is useful if you pass the -DDEBUG flag to 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 your use in programming, the predefined macros should not be directly modified.

Macro	Description
__DATE__	The current date as a character literal in "MMM DD YYYY" format
__TIME__	The current time as a character literal in "HH:MM:SS" format

`__FILE__` This contains the current filename as a string literal.

`__LINE__` This contains the current line number as a decimal constant.

`__STDC__` Defined as 1 when the compiler complies with the ANSI standard.

Let's try the following example:

```
#include <stdio.h>

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 following operators to help you in creating macros:

Macro Continuation (\)

A macro usually must be contained on 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")

```

Stringize (#)

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 that has 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;
}

```

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

Carole and Debra: We love you!

Token Pasting (##)

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>

#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

```

How it happened, 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>

```

REFERENCES

The c programming language by Brain W.Kernighan,Dennis M.ritchie

```
#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 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 and macro name and open parenthesis. For example:

```
#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