OBJECTIVE-C PREPROCESSORS

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

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 OCPP to replace instances of MAX_ARRAY_LENGTH with 20. Use *#define* for constants to increase readability.

```
#import <Foundation/Foundation.h>
#include "myheader.h"
```

These directives tell the OCPP to get foundation.h from **Foundation Framework** and add the text to the current source file. The next line tells OCPP 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 OCPP to undefine existing FILE_SIZE and define it as 42.

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

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

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

This tells the OCPP 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:

```
#import <Foundation/Foundation.h>

int main()
{
   NSLog(@"File :%s\n", __FILE__ );
   NSLog(@"Date :%s\n", __DATE__ );
   NSLog(@"Time :%s\n", __TIME__ );
   NSLog(@"Line :%d\n", __LINE__ );
   NSLog(@"ANSI :%d\n", __STDC__ );

   return 0;
}
```

When the above code in a file **main.m** is compiled and executed, it produces the following result:

```
2013-09-14 04:46:14.859 demo[20683] File :main.m
2013-09-14 04:46:14.859 demo[20683] Date :Sep 14 2013
2013-09-14 04:46:14.859 demo[20683] Time :04:46:14
2013-09-14 04:46:14.859 demo[20683] Line :8
2013-09-14 04:46:14.859 demo[20683] ANSI :1
```

Preprocessor Operators

The Objective-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)  \
    NSLog(@#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:

```
#import <Foundation/Foundation.h>

#define  message_for(a, b)  \
    NSLog(@#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:

```
2013-09-14 05:46:14.859 demo[20683] 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:

```
#import <Foundation/Foundation.h>

#define tokenpaster(n) NSLog (@"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:

```
2013-09-14 05:48:14.859 demo[20683] token34 = 40
```

How it happened, because this example results in the following actual output from the preprocessor:

```
NSLog (@"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:

```
#import <Foundation/Foundation.h>

#if !defined (MESSAGE)
   #define MESSAGE "You wish!"
#endif

int main(void)
```

```
{
   NSLog(@"Here is the message: %s\n", MESSAGE);
   return 0;
}
```

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

```
2013-09-14 05:48:19.859 demo[20683] Here is the message: You wish!
```

Parameterized Macros

One of the powerful functions of the OCPP 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 macro name and open parenthesis. For example:

```
#import <Foundation/Foundation.h>

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

int main(void)
{
   NSLog(@"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:

```
2013-09-14 05:52:15.859 demo[20683] Max between 20 and 10 is 20
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js