Commonly used C Preprocessor Directives

Contributed by shabbir () On 22nd September, 2006
This is an article on Commonly used C Preprocessor Directives **in** C.
Commonly used C Preprocessor Directives

#define

You can use the #define directive to give a meaningful name to a constant in your program. The two forms of the syntax are:

Syntax :
```
#define identifier token-string

#define identifier[( identifier, ... , identifier )] token-string
```

The #define directive substitutes token-string for all subsequent occurrences of an identifier in the source file.

#error

Syntax :
```
#error token-string
```

When `#error` directives are encountered, compilation terminates. These directives are most useful for detecting programmer inconsistencies and violation of constraints during preprocessing.

Example :
Code:

```
#if !defined(__cplusplus)
#error C++ compiler required.
#endif
```

#undef

As its name implies, the #undef directive removes (undefines) a name previously created with #define.

Syntax

```
#undef identifier
```

#if, #elif, #else, #endif

The #if directive, with the #elif, #else, and #endif directives, controls compilation of portions of a source file. If the expression you write (after the #if) has a nonzero value, the line group immediately following the #if directive is retained in the translation unit.

Syntax
Code:

```
#if
    line text
#elif
    line text
#else
    line text
#endif
```

#include

The #include directive tells the preprocessor to treat the contents of a specified file as if those contents had appeared in the source program at the point where the directive appears.

Syntax

```
#include "path-spec"
```

This form instructs the preprocessor to look for include files in the same directory of the file that contains the #include statement

```
#include <path-spec>
```

This form instructs the preprocessor to search for include files first along the path specified by the /I compiler option, then along the path specified by the INCLUDE environment variable.

#ifdef, #ifndef

The #ifdef and #ifndef directives perform the same task as the #if directive when it is used with defined(identifier).

Syntax

```
#ifdef identifier
```

```
#ifndef identifier
```

is equivalent to

```
#if defined identifier
```

```
#if !defined identifier
```

#pragma

Each implementation of C and C++ supports some features unique to its host machine or operating system. Some programs, for instance, need to exercise precise control over the memory areas where data is placed or to control the way certain functions receive parameters. The #pragma directives offer a way for each compiler to offer machine- and operating-system-specific features while retaining overall compatibility with the C and C++ languages. Pragmas are machine- or operating-system-specific by definition, and are usually different for every compiler.

Syntax

```
#pragma token-string
```

token-string can be any one of the following.

```
alloc_text
comment
init_seg1
optimize
auto_inline
component
inline_depth
pack
bss_seg
data_seg
inline_recursion
pointers_to_members1
check_stack
function
intrinsic
setlocale
code_seg
hdrstop
message
vtordisp1
const_seg
include_alias
once
warning
```

Instead of going into the details of what each one of them is I would discuss the most commonly used one here as thats the title of the article and that is `once`.

It specifies that the file, in which the pragma resides, will be included (opened) only once by the compiler in a building of a particular file.

```
#pragma once
```

There is one big confusion among the developers that header file is included once if it contains `#pragma once` only for the entire build but actually thats not the case. When you have two files code1.cpp and code2.cpp and one header file header.h. Put #pragma once and then a variable declaration in the header file as `int iTest = 10;` and try including the header file from both the cpp files and you will see an error like `iTest` already defined in xxxx.obj.

This is not because `#pragma` is not working but actually the concept is for the compilation of the .cpp file a header file can be included multiple times from different locations and that is prevented using the #pragma once