PREPROCESSING : commands, file inclusion, macros, conditional ...

PREPROCESSOR COMMANDS

#define Define a preprocessor macro.

Usage: #define MAX 256

#undef Remove a preprocessor macro definition.

Usage: #undef MAX

#include Insert text from another source file.

Usage: #include <file> ->searches 'standard' locations
#include "file" ->searches 'local' locations

#if Conditionally include some text, based on the value of a constant expression.

Usage: #if MAX > 256
#include "altSize.h"
#endif

#ifdef Conditionally include some text, based on whether a macro name is defined.

#ifndef Conditionally include some text, based on whether a macro name is not defined.

Usage: #ifndef MAX
#define MAX 256
#endif

#else Alternatively include some text, if the previous #if, #ifdef, #ifndef, or #elif test failed.

#endif Terminate conditional text.

#line Supply a line number for compiler messages.

#elif Alternatively include some text based on the value of another constant expression, if the previous #if, #ifdef, #ifndef, or #elif test failed.

defined Preprocessor function that yields 1 if a name is defined as a preprocessor macro and 0 otherwise; used in #if and #elif statements.

Usage: #if defined TOKEN -or- #if defined(TOKEN)

unary # Operator to replace macro parameter with a string constant containing the parameter's value.

Usage: #define PRINT(a) printf("value = " #a "\n")

'PRINT(5);' becomes 'printf("value = 5\n");'

binary ## Operator to create a single token out of two adjacent tokens.

Usage: #define INPUT(i) input ## i

'INPUT(1) = INPUT(2);' becomes 'temp1 = temp2;'

#pragma Specify implementation-dependent information to the compiler. Although it's best not to have implementation-dependent pre-processing statements, it's a very pragmatic inclusion to the list of preprocessing commands (hence it's name).

#error Produce a compile-time error with a designated message.

Usage: #error "Opps! MAX not Defined."

FILE INCLUSION

You use the '#include' preprocessing directive to include header files. The entire contents of the specified file is inserted in place of the file inclusion statement. The following are some examples:

```
#include <iostream.h>
#include <stdlib.h>
#include "myHeader.h"
```

The '<>' brackets identify standard libraries which are part of the compiler and follow implementation-defined rules to find the file. The double-quote brackets define local files (usually within the directory or sub-directories that your project file is located in).

MACROS

Macros use the '#define' preprocessing directive to substitute any occurrence of the macro 'name' with the identified text in the macro definition. Examples include:

```
#define MAX_CHARS 255
#define ERROR_STR "\pError, try again."
#define MAX(A,B)  ( (A)>(B) ? (A) : (B) )
```

CONDITIONAL INCLUSION

You can use conditional inclusion preprocessing directives to define or include items which are compiler or operating system specific.

```
#ifndef OS
#define OS MAC
#endif

#if OS == MAC
  #define SYS_HEADER "MacOS.h"
#elif OS == WIN
  #define SYS_HEADER "Win.h"
#elif OS == WIN95
  #define SYS_HEADER "Win95.h"
#elif OS == UNIX
  #define SYS_HEADER "Unix.h"
#endif
#include SYS_HEADER
```

PRAGMA DIRECTIVES

Pragma directives are compiler-specific and therefore, tend to be less portable. The format for THINK C pragma directives is:

```
#pragma [SC] pragma_directive [pragma_args]
```

If you specify SC, the directive must be recognized by Symantec C/C++ compilers. Some examples of THINK C pragma directives include:

```
#pragma [SC] align [1/2/4]            // sets byte alignments within
                                      // structures.

#pragma [SC] message "text"           // prints text while compiling.

#pragma [SC] once                     // when included in a header file,
                                      // file is included only once even
                                      // if #include directives include
                                      // it multiple times.

#pragma [SC] options align=power      // equivalent to align 4 directive.
#pragma [SC] options align=native

#pragma [SC] options align=mac68k     // equivalent to align 2 directive.

#pragma [SC] options align=reset      // default alignment.

#pragma [SC] template class<args>     // produces instantiations of a
                                      // template.

#pragma [SC] template_access code     // code type can be public, extern,
                                      // or static.
```