Subsections

The C Preprocessor

Recall that preprocessing is the first step in the C program compilation stage -- this feature is unique to C compilers.

The preprocessor more or less provides its own language which can be a very powerful tool to the programmer. Recall that all preprocessor directives or commands begin with a #.

Use of the preprocessor is advantageous since it makes:

- programs easier to develop,
- easier to read,
- easier to modify
- C code more transportable between different machine architectures.

The preprocessor also lets us customise the language. For example to replace { ... } block statements delimiters by PASCAL like `begin ... end` we can do:

```
#define begin {
             #define end  }
```

During compilation all occurrences of `begin and end get replaced by corresponding { or } and so the subsequent C compilation stage does not know any difference!!!.`

`Lets look at #define in more detail`

#define

Use this to define constants or any macro substitution. Use as follows:

```
#define  <macro> <replacement name>
```

For Example

```
#define FALSE  0
#define TRUE  !FALSE
```

We can also define small ``functions" using `#define`. For example max. of two variables:

```
   #define max(A,B) ( (A) > (B) ? (A):(B))
```

? is the ternary operator in C.

Note: that this does not define a proper function max.

All it means that wherever we place max(C†,D†) the text gets replaced by the appropriate definition. [† = any variable names - not necessarily C and D]

So if in our C code we typed something like:

```
   x = max(q+r,s+t);
```

after preprocessing, if we were able to look at the code it would appear like this:

```
   x = ( (q+r) > (r+s) ? (q+r) : (s+t));
```

Other examples of #define could be:

```
#define Deg_to_Rad(X) (X*M_PI/180.0)
/* converts degrees to radians, M_PI is the value
of pi and is defined in math.h library */

#define LEFT_SHIFT_8 <<8


NOTE: The last macro LEFT_SHIFT_8 is only
valid so long as replacement context is valid i.e.
x = y LEFT_SHIFT_8.
```

#undef

This commands undefined a macro. A macro **must** be undefined before being redefined to a different value.

#include

This directive includes a file into code.

It has two possible forms:

```
   #include <file>
```

or

```
                #include ``file''
```

<file> tells the compiler to look where system include files are held.

Usually UNIX systems store files in ***\usr\include*** directory.

``file'' looks for a file in the current directory (where program was run from)

Included files usually contain C prototypes and declarations from header files and not (algorithmic) C code (SEE next Chapter for reasons)

#if -- Conditional inclusion

`#if` evaluates a constant integer expression. You always need a `#endif` to delimit end of statement.

We can have ***else etc.*** as well by using `#else` and `#elif` -- else if.

Another common use of `#if` is with:

`#ifdef`
-- if defined and
`#ifndef`
-- if not defined

These are useful for checking if macros are set -- perhaps from different program modules and header files.

For example, to set integer size for a portable C program between TurboC (on MSDOS) and Unix (or other) Operating systems. Recall that TurboC uses 16 bits/integer and UNIX 32 bits/integer.

Assume that if TurboC is running a macro `TURBOC` will be defined. So we just need to check for this:

```
#ifdef TURBOC
                  #define INT_SIZE 16
                #else
                                #define INT_SIZE  32
                #endif
```

As another example if running program on MSDOS machine we want to include file msdos.h otherwise a default.h file. A macro `SYSTEM is set (by OS) to type of system so check for this:`

```
#if SYSTEM == MSDOS
                  #include <msdos.h>
                #else
                                #include ``default.h''
                #endif
```

Preprocessor Compiler Control

You can use the `cc` compiler to control what values are set or defined from the command line. This gives some flexibility in setting customised values and has some other useful

functions. The `-D` compiler option is used. For example:

```
cc -DLINELENGTH=80 prog.c -o prog
```

has the same effect as:

```
#define LINELENGTH 80
```

Note that any `#define` or `#undef` **within** the program (`prog.c` above) **override** command line settings.

You can also set a symbol without a value, for example:

```
cc -DDEBUG prog.c -o prog
```

Here the value is assumed to be 1.

The setting of such flags is useful, especially for debugging. You can put commands like:

```
#ifdef DEBUG
     print("Debugging: Program Version 1\");
#else
     print("Program Version 1 (Production)\");
#endif
```

Also since preprocessor command can be written anywhere in a C program you can filter out variables etc for printing ***etc.*** when debugging:

```
x = y *3;

#ifdef DEBUG
     print("Debugging: Variables (x,y) = \",x,y);
#endif
```

The `-E` command line is worth mentioning just for academic reasons. It is not that practical a command. The `-E command` will force the compiler to stop after the preprocessing stage and output the current state of your program. Apart from being debugging aid for preprocessor commands and also as a useful initial learning tool (try this option out with some of the examples above) it is not that commonly used.

Other Preprocessor Commands

There are few other preprocessor directives available:

`#error`
`text of error message` -- generates an appropriate compiler error message. ***e.g***

```
#ifdef OS_MSDOS
                #include <msdos.h>
              #elifdef OS_UNIX
                                #include ``default.h''
             #else
                              #error Wrong OS!!
             #endif
```

 # line
`number "string" -- informs the preprocessor that the number is the next number of line of input. "string" is optional and names the next line of input. This is most often used with programs that`

```
translate other languages to C. For example, error messages
produced by the C compiler can reference the file name and line
numbers of the original source files instead of the intermediate C
(translated) source files.
```

Exercises

Exercise 12529

Define a preprocessor macro `swap(t, x, y)` that will swap two arguments `x` and `y` of a given type `t`.

Exercise 12531

Define a preprocessor macro to select:

- the least significant bit from an `unsigned char`
- the *n*th (assuming least significant is 0) bit from an `unsigned char`.

Dave Marshall
1/5/1999