will fail (but not generate an error message), while the test

```
#if !DEBUG
```

will succeed.

The `defined` Operator

We encountered the # and ## operators in Section 14.3. There's just one other operator, `defined`, that's specific to the preprocessor. When applied to an identifier, `defined` produces the value 1 if the identifier is a currently defined macro; it produces 0 otherwise. The `defined` operator is normally used in conjunction with the `#if` directive; it allows us to write

```
#if defined(DEBUG)
...
#endif
```

The lines between the `#if` and `#endif` directives will be included in the program only if `DEBUG` is defined as a macro. The parentheses around `DEBUG` aren't required; we could simply write

```
#if defined DEBUG
```

Since `defined` tests only whether `DEBUG` is defined or not, it's not necessary to give `DEBUG` a value:

```
#define DEBUG
```

The `#ifdef` and `#ifndef` Directives

The `#ifdef` directive tests whether an identifier is currently defined as a macro:

`#ifdef` directive

```
#ifdef identifier
```

Using `#ifdef` is similar to using `#if`:

```
#ifdef identifier
Lines to be included if identifier is defined as a macro
#endif
```

Q&A Strictly speaking, there's no need for `#ifdef`, since we can combine the `#if` directive with the `defined` operator to get the same effect. In other words, the directive

```
#ifdef identifier
```

is equivalent to

```
#if defined(identifier)
```