integer constant `199409L` (representing the year and month of the amendment). If a compiler conforms to the C99 standard, the value is `199901L`. For each subsequent version of the standard (and each amendment to the standard), this macro will have a different value.

A C99 compiler may (or may not) define three additional macros. Each macro is defined only if the compiler meets a certain requirement:

IEEE floating-point standard ➤7.2

- `__STDC_IEC_559__` is defined (and has the value 1) if the compiler performs floating-point arithmetic according to the IEC 60559 standard (another name for the IEEE 754 standard).
- `__STDC_IEC_559_COMPLEX__` is defined (and has the value 1) if the compiler performs complex arithmetic according to the IEC 60559 standard.
- `__STDC_ISO_10646__` is defined as an integer constant of the form *yyyymm*`L` (for example, `199712L`) if values of type `wchar_t` are represented by the codes in the ISO/IEC 10646 standard (with revisions as of the specified year and month).

`wchar_t` type ➤25.2
ISO/IEC 10646 standard ➤25.2

C99 Empty Macro Arguments

C99 allows any or all of the arguments in a macro call to be empty. Such a call will contain the same number of commas as a normal call, however. (That way, it's easy to see which arguments have been omitted.)

In most cases, the effect of an empty argument is clear. Wherever the corresponding parameter name appears in the replacement list, it's replaced by nothing—it simply disappears from the replacement list. Here's an example:

```
#define ADD(x,y) (x+y)
```

After preprocessing, the statement

```
i = ADD(j,k);
```

becomes

```
i = (j+k);
```

whereas the statement

```
i = ADD(,k);
```

becomes

```
i = (+k);
```

When an empty argument is an operand of the # or ## operators, special rules apply. If an empty argument is "stringized" by the # operator, the result is `""` (the empty string):

```
#define MK_STR(x) #x
...
char empty_string[] = MK_STR();
```