After preprocessing, the declaration will have the following appearance:

```
char empty_string[] = "";
```

If one of the arguments of the ## operator is empty, it's replaced by an invisible "placemarker" token. Concatenating an ordinary token with a placemarker token yields the original token (the placemarker disappears). If two placemarker tokens are concatenated, the result is a single placemarker. Once macro expansion has been completed, placemarker tokens disappear from the program. Consider the following example:

```
#define JOIN(x,y,z) x##y##z
...
int JOIN(a,b,c), JOIN(a,b,), JOIN(a,,c), JOIN(,,c);
```

After preprocessing, the declaration will have the following appearance:

```
int abc, ab, ac, c;
```

The missing arguments were replaced by placemarker tokens, which then disappeared when concatenated with any nonempty arguments. All three arguments to the JOIN macro could even be missing, which would yield an empty result.

C99 Macros with a Variable Number of Arguments

variable-length argument lists
➤26.1

In C89, a macro must have a fixed number of arguments, if it has any at all. C99 loosens things up a bit, allowing macros that take an unlimited number of arguments. This feature has long been available for functions, so it's not surprising that macros were finally put on an equal footing.

The primary reason for having a macro with a variable number of arguments is that it can pass these arguments to a function that accepts a variable number of arguments, such as `printf` or `scanf`. Here's an example:

```
#define TEST(condition, ...) ((condition)? \
  printf("Passed test: %s\n", #condition): \
  printf(__VA_ARGS__))
```

The . . . token, known as ***ellipsis***, goes at the end of a macro's parameter list, preceded by ordinary parameters, if there are any. `__VA_ARGS__` is a special identifier that can appear only in the replacement list of a macro with a variable number of arguments; it represents all the arguments that correspond to the ellipsis. (There must be at least one argument that corresponds to the ellipsis, although that argument may be empty.) The `TEST` macro requires at least two arguments. The first argument matches the `condition` parameter; the remaining arguments match the ellipsis.

Here's an example that shows how the `TEST` macro might be used:

```
TEST(voltage <= max_voltage,
     "Voltage %d exceeds %d\n", voltage, max_voltage);
```

The preprocessor will produce the following output (reformatted for readability):