The `do` loop in Example 8.3 could be replaced with:

```
cin >> word
while (*word)
{ cout << "\t\"" << word << "\"\n";
  cin >> word;
}
```

When Ctrl+Z is pressed, the call `cin >> word` assigns the empty C-string to `word`.

Example 8.3 and Example 8.1 illustrate an important distinction: the output operator `<<` behaves differently with pointers of type `char*` than with other pointer types. With a `char*` pointer, the operator outputs the entire character string to which the pointer points. But with any other pointer type, the operator will simply output the address of the pointer.

8.5 SOME `cin` MEMBER FUNCTIONS

The input stream object `cin` includes the input functions: `cin.getline()`, `cin.get()`, `cin.ignore()`, `cin.putback()`, and `cin.peek()`. Each of these function names includes the prefix "`cin.`" because they are "member functions" of the `cin` object.

The call `cin.getline(str,n)` reads up to `n` characters into `str` and ignores the rest.

EXAMPLE 8.4 The `cin.getline()` Function with Two Parameters

This program echoes the input, line by line:

```
int main()
{ char line[80];
  do
  { cin.getline(line, 80);
    if (*line) cout << "\t[" << line << "]\n";
  } while (*line);
}
```

Note that the condition `(*line)` will evaluate to "true" precisely when `line` contains a non-empty C-string, because only then will `line[0]` be different from the NUL character (ASCII value 0).

The call `cin.getline(str,n,ch)` reads all input up to the first occurrence of the delimiting character `ch` into `str`. If the specified character `ch` is the newline character `'\n'`, then this is equivalent to `cin.getline(str,n)`. This is illustrated in the next example where the delimiting character is the comma `','`.

EXAMPLE 8.5 The `cin.getline()` Function

This program echoes the input, clause by clause:

```
int main()
{ char clause[80];
  do
  { cin.getline(clause, 80, ',');
    if (*clause) cout << "\t[" << clause << "]\n";
  } while (*clause);
}
```