Let's analyze the loop condition:

```c
while ((ch = cin.get()) != EOF)
```

The parentheses that enclose the subexpression `ch=cin.get()` cause the program to evaluate that expression first. To do the evaluation, the program first has to call the `cin.get()` function. Next, it assigns the function return value to `ch`. Because the value of an assignment statement is the value of the left operand, the whole subexpression reduces to the value of `ch`. If this value is EOF, the loop terminates; otherwise, it continues. The test condition needs all the parentheses. Suppose we leave some out:

```c
while (ch = cin.get() != EOF)
```

The `!=` operator has higher precedence than `=`, so first the program compares `cin.get()`'s return value to EOF. A comparison produces a `false` or `true` result; that `bool` value is converted to 0 or 1, and that's the value that gets assigned to `ch`.

Using `cin.get(ch)` (with an argument) for input, on the other hand, doesn't create any type problems. The `cin.get(char)` function, recall, doesn't assign a special value to `ch` on end-of-file. In fact it doesn't assign anything to `ch` in that case. `ch` is never called upon to hold a non-`char` value. Table 5.3 summarizes the differences between `cin.get(char)` and `cin.get()`.

So which should you use, `cin.get()` or `cin.get(char)`? The form with the character argument is integrated more fully into the object approach because its return value is an `istream` object. This means, for example, that you can chain uses. For example, the following code means read the next input character into `ch1` and the following input character into `ch2`:

```c
cin.get(ch1).get(ch2);
```

This works because the function call `cin.get(ch1)` returns the `cin` object, which then acts as the object to which `get(ch2)` is attached.

Probably the main use for the `get()` form is to let you make quick-and-dirty conversions from the `getchar()` and `putchar()` functions of `stdio.h` to the `cin.get()` and `cout.put()`