```
/**********************************************************
 * print: Prints a listing of all parts in the database,  *
 *        showing the part number, part name, and         *
 *        quantity on hand. Parts are printed in the      *
 *        order in which they were entered into the       *
 *        database.                                       *
 **********************************************************/
void print(void)
{
  int i;

  printf("Part Number   Part Name                  "
         "Quantity on Hand\n");
  for (i = 0; i < num_parts; i++)
    printf("%7d       %-25s%11d\n", inventory[i].number,
           inventory[i].name, inventory[i].on_hand);
}
```

In the `main` function, the format string `" %c"` allows `scanf` to skip over white space before reading the operation code. The space in the format string is crucial; without it, `scanf` would sometimes read the new-line character that terminated a previous line of input.

The program contains one function, `find_part`, that isn't called from `main`. This "helper" function helps us avoid redundant code and simplify the more important functions. By calling `find_part`, the `insert`, `search`, and `update` functions can locate a part in the database (or simply determine if the part exists).

There's just one detail left: the `read_line` function, which the program uses to read the part name. Section 13.3 discussed the issues that are involved in writing such a function. Unfortunately, the version of `read_line` in that section won't work properly in the current program. Consider what happens when the user inserts a part:

```
Enter part number: 528
Enter part name: Disk drive
```

The user presses the Enter key after entering the part number and again after entering the part name, each time leaving an invisible new-line character that the program must read. For the sake of discussion, let's pretend that these characters are visible:

```
Enter part number: 528¤
Enter part name: Disk drive¤
```

When we call `scanf` to read the part number, it consumes the `5`, `2`, and `8`, but leaves the ¤ character unread. If we try to read the part name using our original `read_line` function, it will encounter the ¤ character immediately and stop reading. This problem is common when numerical input is followed by character input. Our solution will be to write a version of `read_line` that skips white-