Ordinary Characters in Format Strings

The concept of pattern-matching can be taken one step further by writing format strings that contain ordinary characters in addition to conversion specifications. The action that `scanf` takes when it processes an ordinary character in a format string depends on whether or not it's a white-space character.

- ***White-space characters.*** When it encounters one or more consecutive white-space characters in a format string, `scanf` repeatedly reads white-space characters from the input until it reaches a non-white-space character (which is "put back"). The number of white-space characters in the format string is irrelevant; one white-space character in the format string will match any number of white-space characters in the input. (Incidentally, putting a white-space character in a format string doesn't force the input to contain white-space characters. A white-space character in a format string matches *any* number of white-space characters in the input, including none.)
- ***Other characters.*** When it encounters a non-white-space character in a format string, `scanf` compares it with the next input character. If the two characters match, `scanf` discards the input character and continues processing the format string. If the characters don't match, `scanf` puts the offending character back into the input, then aborts without further processing the format string or reading characters from the input.

  For example, suppose that the format string is `"%d/%d"`. If the input is

```
•5/•96
```

`scanf` skips the first space while looking for an integer, matches `%d` with `5`, matches `/` with `/`, skips a space while looking for another integer, and matches `%d` with `96`. On the other hand, if the input is

```
•5•/•96
```

`scanf` skips one space, matches `%d` with `5`, then attempts to match the `/` in the format string with a space in the input. There's no match, so `scanf` puts the space back; the `•/•96` characters remain to be read by the next call of `scanf`. To allow spaces after the first number, we should use the format string `"%d /%d"` instead.

Confusing `printf` with `scanf`

Although calls of `scanf` and `printf` may appear similar, there are significant differences between the two functions; ignoring these differences can be hazardous to the health of your program.

One common mistake is to put `&` in front of variables in a call of `printf`:

```
printf("%d %d\n", &i, &j);    /*** WRONG ***/
```