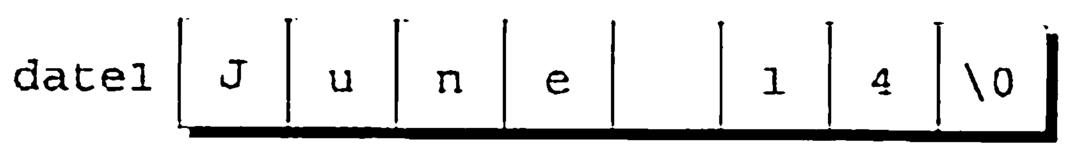
The compiler will put the characters from `"June 14"` in the `date1` array, then add a null character so that `date1` can be used as a string. Here's what `date1` will look like:



Although `"June 14"` appears to be a string literal, it's not. Instead, C views it as an abbreviation for an array initializer. In fact, we could have written

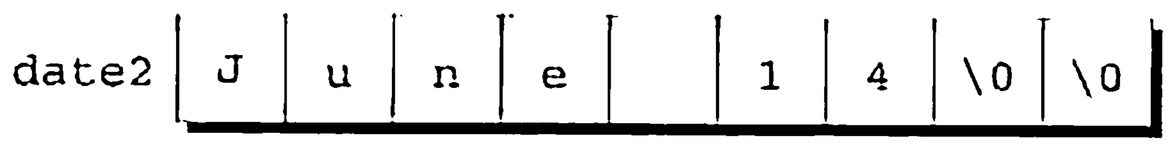
```
char date1[8] = {'J', 'u', 'n', 'e', ' ', '1', '4', '\0'};
```

I think you'll agree that the original version is easier to read.

What if the initializer is too short to fill the string variable? In that case, the compiler adds extra null characters. Thus, after the declaration

```
char date2[9] = "June 14";
```

`date2` will have the following appearance:



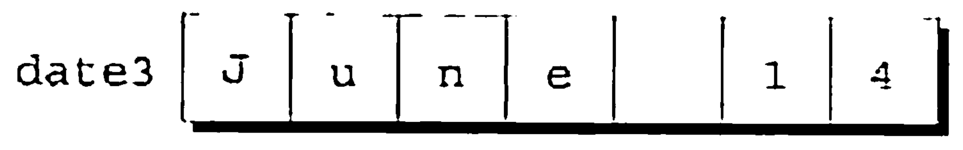
array initializers ➤8.1

This behavior is consistent with C's treatment of array initializers in general. When an array initializer is shorter than the array itself, the remaining elements are initialized to zero. By initializing the leftover elements of a character array to `\0`, the compiler is following the same rule.

What if the initializer is longer than the string variable? That's illegal for strings, just as it's illegal for other arrays. However, C does allow the initializer (not counting the null character) to have exactly the same length as the variable:

```
char date3[7] = "June 14";
```

There's no room for the null character, so the compiler makes no attempt to store one:



If you're planning to initialize a character array to contain a string, be sure that the length of the array is longer than the length of the initializer. Otherwise, the compiler will quietly omit the null character, making the array unusable as a string.

The declaration of a string variable may omit its length, in which case the compiler computes it:

```
char date4[] = "June 14";
```