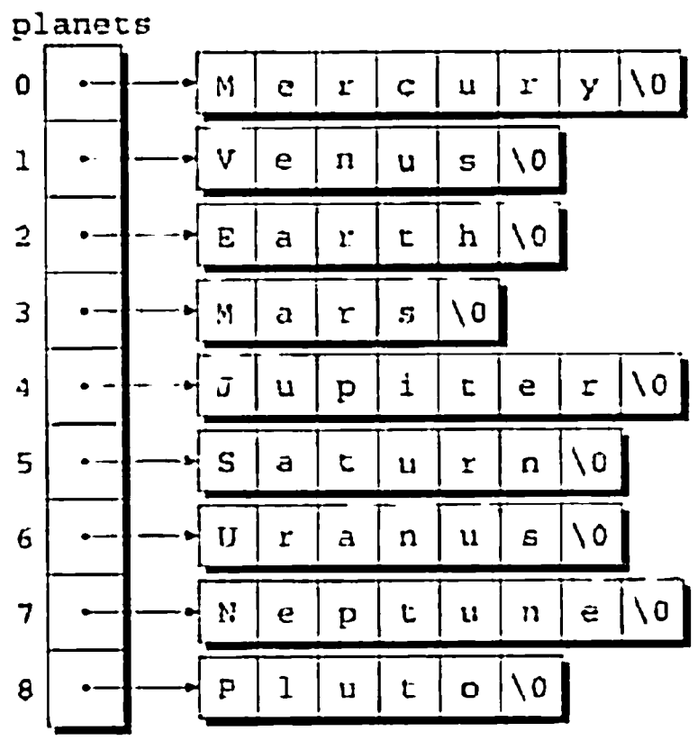
	0	1	2	3	4	5	6	7
0	M	e	r	c	u	r	y	\0
1	V	e	n	u	s	\0	\0	\0
2	E	a	r	t	h	\0	\0	\0
3	M	a	r	s	\0	\0	\0	\0
4	J	u	p	i	t	e	r	\0
5	S	a	t	u	r	n	\0	\0
6	U	r	a	n	u	s	\0	\0
7	N	e	p	t	u	n	e	\0
8	P	l	u	t	o	\0	\0	\0

The inefficiency that's apparent in these examples is common when working with strings, since most collections of strings will have a mixture of long strings and short strings. What we need is a *ragged array:* a two-dimensional array whose rows can have different lengths. C doesn't provide a "ragged array type," but it does give us the tools to simulate one. The secret is to create an array whose elements are *pointers* to strings.

Here's the `planets` array again. this time as an array of pointers to strings:

```
char *planets[] = {"Mercury", "Venus", "Earth",
                   "Mars", "Jupiter", "Saturn",
                   "Uranus", "Neptune", "Pluto"};
```

Not much of a change, eh? We simply removed one pair of brackets and put an asterisk in front of `planets`. The effect on how `planets` is stored is dramatic. though:



Each element of `planets` is a pointer to a null-terminated string. There are no longer any wasted characters in the strings, although we've had to allocate space for the pointers in the `planets` array.

To access one of the planet names, all we need do is subscript the `planets` array. Because of the relationship between pointers and arrays, accessing a character in a planet name is done in the same way as accessing an element of a two-