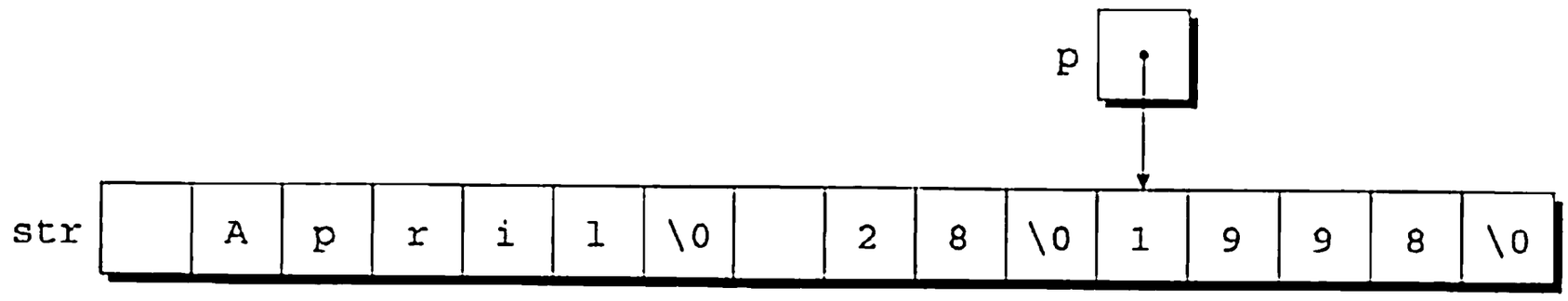
After this call, `str` will have the following appearance:



When `strtok` is called repeatedly to break a string into tokens, the second argument isn't required to be the same in each call. In our example, the second call of `strtok` has the argument `" \t,"` instead of `" \t"`.

`strtok` has several well-known problems that limit its usefulness; I'll mention just a couple. First, it works with only one string at a time; it can't conduct simultaneous searches through two different strings. Also, `strtok` treats a sequence of delimiters in the same way as a single delimiter, making it unsuitable for applications in which a string contains a series of fields separated by a delimiter (such as a comma) and some of the fields are empty.

Miscellaneous Functions

```
void *memset(void *s, int c, size_t n);
size_t strlen(const char *s);
```

memset

`memset` stores multiple copies of a character in a specified area of memory. If `p` points to a block of `N` bytes, for example, the call

```
memset(p, ' ', N);
```

will store a space in every byte of the block. One of `memset`'s uses is initializing an array to zero bits:

```
memset(a, 0, sizeof(a));
```

`memset` returns its first argument (a pointer).

strlen

`strlen` returns the length of a string, not counting the null character. See Section 13.5 for examples of `strlen` calls.

`strerror` function ➤*24.2*

There's one other miscellaneous string function, `strerror`, which is covered along with the `<errno.h>` header.

Q & A

Q: **Why does the `expm1` function exist, since all it does is subtract 1 from the value returned by the `exp` function? [p. 605]**

A: When applied to numbers that are close to zero, the `exp` function returns a value that's very close to 1. The result of subtracting 1 from the value returned by `exp` may not be accurate because of round-off error. `expm1` is designed to give a more accurate result in this situation.