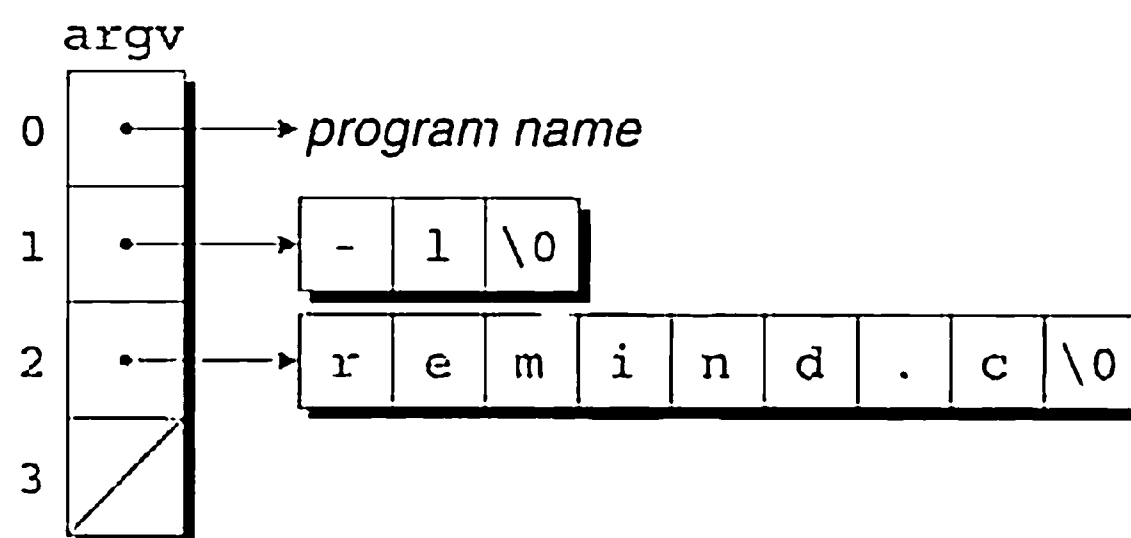


name, `argv[1]` will point to the string `"-l"`, `argv[2]` will point to the string `"remind.c"`, and `argv[3]` will be a null pointer:



This figure doesn't show the program name in detail, since it may include a path or other information that depends on the operating system. If the program name isn't available, `argv[0]` points to an empty string.

Since `argv` is an array of pointers, accessing command-line arguments is easy. Typically, a program that expects command-line arguments will set up a loop that examines each argument in turn. One way to write such a loop is to use an integer variable as an index into the `argv` array. For example, the following loop prints the command-line arguments, one per line:

```
int i;

for (i = 1; i < argc; i++)
    printf("%s\n", argv[i]);
```

Another technique is to set up a pointer to `argv[1]`, then increment the pointer repeatedly to step through the rest of the array. Since the last element of `argv` is always a null pointer, the loop can terminate when it finds a null pointer in the array:

```
char **p;

for (p = &argv[1]; *p != NULL; p++)
    printf("%s\n", *p);
```

Since `p` is a *pointer* to a *pointer* to a character, we've got to use it carefully. Setting `p` equal to `&argv[1]` makes sense; `argv[1]` is a pointer to a character, so `&argv[1]` will be a pointer to a pointer. The test `*p != NULL` is OK, since `*p` and `NULL` are both pointers. Incrementing `p` looks good; `p` points to an array element, so incrementing it will advance it to the next element. Printing `*p` is fine, since `*p` points to the first character in a string.

PROGRAM Checking Planet Names

Our next program, `planet.c`, illustrates how to access command-line arguments. The program is designed to check a series of strings to see which ones are names of planets. When the program is run, the user will put the strings to be tested on the command line:

```
planet Jupiter venus Earth fred
```