thus loses the only way the program had to access the newly allocated memory. Instead, you need to use the `strcpy()` library function:

```c
strcpy(ps, animal); // copy string to new storage
```

The `strcpy()` function takes two arguments. The first is the destination address, and the second is the address of the string to be copied. It's up to you to make certain that the destination really is allocated and has sufficient space to hold the copy. That's accomplished here by using `strlen()` to find the correct size and using `new` to get free memory.

Note that by using `strcpy()` and `new`, we get two separate copies of "fox":

```c
fox at 0x0065fd30
fox at 0x004301c8
```

Also note that `new` located the new storage at a memory location quite distant from that of the array `animal`.

Often you encounter the need to place a string into an array. Use the `=` operator when you initialize an array; otherwise, use `strcpy()` or `strncpy()`. You've seen the `strcpy()` function; it works like this:

```c
char food[20] = "carrots"; // initialization
strcpy(food, "flan"); // otherwise
```

Note that something like

```c
strcpy(food, "a picnic basket filled with many goodies");
```

can cause problems because the `food` array is smaller than the string. In this case, the function copies the rest of the string into the memory bytes immediately following the array, which can overwrite other memory your program is using. To avoid that problem, use `strncpy()` instead. It takes a third argument: the maximum number of characters to be copied. Be aware, however, that if this function runs out of space before it reaches the end of the string, it doesn't add the null character. Thus, you should use the function like this:

```c
strncpy(food, "a picnic basket filled with many goodies", 19);
```