When you read a string into a program, you always should use the address of previously allocated memory. This address can be in the form of an array name or of a pointer that has been initialized using `new`.

Next, notice what the following code accomplishes:

```cpp
ps = animal; // set ps to point to string
...
cout << animal << " at " << (int *) animal << endl;
cout << ps << " at " << (int *) ps << endl;
```

It produces the following output:

```
fox at 0x0065fd30
fox at 0x0065fd30
```

Normally, if you give `cout` a pointer, it prints an address. But if the pointer is type `char *`, `cout` displays the pointed-to string. If you want to see the address of the string, you have to type cast the pointer to another pointer type, such as `int *`, which this code does. So, `ps` displays as the string "fox", but `(int *) ps` displays as the address where the string is found. Note that assigning `animal` to `ps` does not copy the string, it copies the address. This results in two pointers (`animal` and `ps`) to the same memory location and string.

To get a copy of a string, you need to do more. First, you need to allocate memory to hold the string. You can do this by declaring a second array or by using `new`. The second approach enables you to custom fit the storage to the string:

```cpp
ps = new char[strlen(animal) + 1]; // get new storage
```

The string "fox" doesn't completely fill the `animal` array, so we're wasting space. This bit of code uses `strlen()` to find the length of the string; it adds 1 to get the length including the null character. Then, the program uses `new` to allocate just enough space to hold the string.

```
ps = new char[strlen(animal) + 1]; // get new storage
```

Next, you need a way to copy a string from the `animal` array to the newly allocated space. It doesn't work to assign `animal` to `ps`, for that just changes the address stored in `ps` and