String Copying | Coursera

coursera.org/learn/pointers-arrays-recursion/supplement/rqww1/string-copying

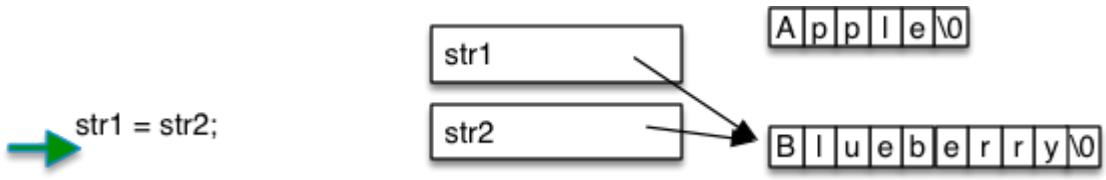
String Copying

String Copying

Take a moment to look at the situation depicted in the figure below. In this figure, the execution arrow is immediately before the assignment statement ;. What do you think will happen when this assignment statement is executed?

str1=str2

The short answer is: we will follow the same rules we always have. The right side of the assignment statement (*str2*) evaluates to an arrow pointing at the second string ("*Blueberry*"). We will take that value (**the arrow**) and copy it into the box named by the left side of the assignment statement (in this case, *str1*'s box). The result is that both *str1* and *str2* point at the same memory location, as depicted in the figure below which shows the state of the program after executing the above line of code. If these strings are pointing at modifiable memory locations, and we later change the contents of one (*e.g.*, we execute *str1[0]='x';*) then if we "look at" that memory location through its other name (*str2[0]*), we will "see" the change.



We may, however, want to actually copy the contents of the string from one location to another. As with comparing for equality, doing this copy yourself requires iterating through the characters of the string and copying them one by one to the destination. In doing so, we must be careful that the destination has sufficient space to receive the string being copied into it.

The C library has a function, *strncpy* which performs this task for us—it copies a string from one location to another, and takes a parameter (*n*) telling it the maximum number of characters it is allowed to copy. If the length of the source string is greater than or equal to *n*, then the destination is *not* null terminated—a situation which the programmer must typically rectify before using the string for any significant purpose.

Note that there is a similarly named function, *strcpy* (the previous one had an **n** in the middle of its name, this one does not). The *strcpy* function is more dangerous, as there is no way to tell it how much space is available in the destination. If insufficient space is available, then *strcpy* will simply overwrite whatever follows it in memory, creating a variety of problems. Some of these problems may result in security hazards. There is another function, *strdup* which allocates space for a copy of the string, and copies it into that space. However, to understand how *strdup* works, we need to discuss dynamic allocation first (which is one of the major topics of Course 4).