Tip 17: Define a string type

Ben Klemens

4 November 2011

level: still basic
purpose: slightly less confusing declarations

Here's a line to paste into the unified header you wrote to include at the head of all your programs back in Tip 6:

```
typedef char* string;
```

Pretty simple, but it will make your code more readable.

The sample code from last time is pretty short, so I'll reprint the whole program. You may find it to be hard to read (I was using it as an example of a point I was calling obscure), but we'll try to fix that:

```
#include <stdio.h>

int main(){
    char *list[] = {"first", "second", "third", NULL};
    for (char **p=list; *p != NULL; p++){
        printf("%s\n", *p);
    }
}
```

Do the declarations communicate to you that `char *list[]` is a list of strings, and that `*p` is a string?

Now use typedef to replace `char *` with `string`. There are fewer stars floating around; `p` is more clearly a pointer and `*p` the data at the pointer:

```
#include <stdio.h>
typedef char* string;

int main(){
    string list[] = {"first", "second", "third", NULL};
    for (string *p=list; *p != NULL; p++){
        printf("%s\n", *p);
    }
}
```

The declaration line for `list` is now as easy as C gets, and clearly indicates that it is a list of strings, and the snippet `string *p` should indicate to you that `p` is a pointer-to-string, so `*p` is a string.

In the end, you'll still have to remember that a string is a pointer-to-`char`; for example, `NULL` is a valid value.

I find that most folks (myself included) have no serious problem with pointers until they get to pointers-to-pointers and further depth beyond that. With strings as `char*`s, you hit that multiple-star wall much earlier than there's any reason to. And since Tips 10 (Entry #060) through 12 already went over how to deal with strings without ever calling `malloc`, we might as well start using a non-pointer type name for them.

To do:
Try declaring a 2-D array of strings, using the typedef above plus

```
typedef stringlist string*
```

Is it easier to work out how to allocate its parts?