# APS105: Absolutely Official Strings Factsheet

**Daniel Zingaro**

March 8, 2010

## Instructions

**Read once a day until April 2010. Then wrap it up and give it to someone for their birthday.**

Here is an outline of some important facts and confusions about strings. The exercises should be attempted as they are reached in the handout.

## Fact 1: Strings Don't Really Exist

Hmmm?

Well, in other programming languages, there is a real `string` type. In C, there is not. What happens is in C is:

1. We use `char` arrays to hold characters of our strings
2. We terminate them with a `'\0'` character
3. All of C's string-processing functions rely on this `'\0'` terminator to detect when the string ends

Therefore, a "string" in C is just a `char` array that ends with a `'\0'`. Everything that is true of arrays is therefore true of strings, including:

- The first character is at index 0
- Elements of strings are not initialized until you give them values
- You should never try to access a character past the end of the string
- You can initialize strings when you declare them

- A string's characters are placed in order in memory, with the address of each character increasing by 1 each time we move to the right
- An array that will be used as a string is not necessarily using all of its elements. The string ends at the first `'\0'` character.

# Confusion 1: Pointers into Strings

Remember: using an array name by itself gives us the address of its first element. Adding an integer `k` to a pointer makes it point `k` elements to the right. We can use both of these facts to access portions of strings. For example, let's say we declare:

```
char g[] = "hello";
```

(C will reserve space for six characters here.) I can print the entire string:

```
printf ("%s\n", g);
```

I can also print starting from the `'e'`:

```
printf ("%s\n", g + 1);
```

This works because `g + 1` is a pointer to the `'e'`. When we tell `printf` to output the string starting there, it finds `"ello"` before hitting the `'\0'` character.

# Exercise 1

What would be printed if we used

```
printf ("%s\n", g + 5);
```

What would be printed if we used

```
printf ("%s\n", &g[1]);
```

In other words: starting `k` positions from the beginning of a string gives us another string (with `k` fewer characters).

# Exercise 2

Consider the following code:

```
char s[] = "funny";
int i = strcmp (s + 2, "nny");
```

What is the value of `i`?

## Fact 2: We can Implement Built-in String Functions

Many of C's built-in string functions do little more than process a character array that ends with `'\0'`. We can therefore gain considerable practice with strings by trying to implement them ourselves. Here is an implementation of `strlen`; it works by counting characters until the `'\0'` is found. We also include a `main` to demonstrate a call of this function:

```
#include <stdio.h>

int myStrlen (const char *s) {
  int i = 0;
  while (s[i] != '\0')
    i++;
  return i;
}

int main(void) {
  char a[] = "gimme";
  printf ("%d\n", myStrlen (a));
  return 0;
}
```

What if we wanted to implement `strcpy` using `strcat`? We could do it like this (again with a small `main` function for testing):

```
#include <string.h>
#include <stdio.h>

void myStrcpy (char *s1, const char *s2) {
  s1[0] = '\0';
  strcat (s1, s2);
}

int main (void) {
char first[20] = "once upon ";
char second[] = "a time";
myStrcpy (first, second);
printf ("First: %s\n", first);
printf ("Second: %s\n", second);
return 0;
}
```

## Exercise 3

Here is a function that returns a pointer to a string's `'\0'` terminator. (In other words, it returns the address of the `'\0'` at the end of a string. It does not return `'\0'`; the return type is `char *`, not `char`.)

```
char *endOfString (const char *s) {
  while (*s != '\0')
    s++;
  return s;
}
```

Use this function to rewrite `myStrlen` in one line. (Hint: use pointer subtraction to determine how many elements come between the start of a string and its `'\0'` terminator.)

## Exercise 4

Implement `strlen` by using a pointer to step through the characters of `s`, rather than by using an integer to access each element. Your loop will terminate when your pointer is pointing to the `'\0'`.

## Confusion 2: Strings and Characters

A single character in single quotes is not a string. Similarly, a string cannot be treated like a character in single quotes. Consider this code:

```
char s[] = "tell";
```

We cannot try to print the `t` like this:

```
printf ("%c\n", s);
```

`s` here is not a character; it's a pointer to a character. The following, then, is legal:

```
printf ("%c\n", *s);
```

## Example 1: Vowels in a String

Let's write a program that tells us the total number of vowels in a string. We'll use integer `i` to loop through each index in the string, and compare each character to the vowels:

```
#include <stdio.h>

int numVowels (const char *s) {
  int i = 0, total = 0;
  while (s[i] != '\0') {
    if (s[i] == 'a' || s[i] == 'e' || s[i] == 'i' || s[i] == 'o'
     || s[i] == 'u')
      total++;
    i++;
  }
  return total;
}

int main (void) {
  printf ("%d\n", numVowels ("abcda"));
  return 0;
  }
```

# Exercise 5

This solution involves a tedious expression that tests for each vowel. Rewrite the solution so that it uses a string constant initialized to the five characters `aeiou`. Then, to check whether each character in the string is a vowel, use `strchr`. (You should make only one call of `strchr` on each iteration.)