C-String. C-String Run

CS 211

Adam Sweeney

October 27, 2017

Wichita State University, EECS

Introduction

- We have been dealing with arrays of characters
- Words and messages are extremely common in any program
- C++ gives us a couple methods for dealing with these “strings”
- We’ll be going over the older method today

Agenda

- Section 8.1
- Declaring
- C-Strings are different

Declaring C-Strings

First, What is even a C-String

- As the name implies, C-Strings are inherited from C
- C-Strings are a special array of characters
- What makes them special?
    - '\0'
    - This special character (a backslash and zero) is called the *null character*
    - An array of characters is **not** a C-String unless it ends with the null character

Now, the Declaring

- C-Strings are a null character terminated array of characters
- As it happens, unlike an `int` or `char` or `double`, there are actually "multiple" ways we can declare a C-String

```
char word[10] = "Hello";
char word[] = {'H', 'e', 'l', 'l', 'o', '\0'}
char word[] = "Hello";
char *word = "Hello";
```

- That last one is our first look at a pointer
- Just showing it because it is a legitimate way to declare a C-String
    - It may throw a warning on newer compilers, but it still works

How not to declare a C-String

```
char word[10];
word = "Hello";
```

- We cannot assign a value after declaration, **only** at initialization
- But what if we actually need to assign a new value, or assign a value later?
    - There is a function we can use
    - Must include library <cstring>

```
#include <cstring>

char word[10];
strcpy(word, "Hello");
```

- We can see that the second parameter should be a C-String, and it gets placed in our variable, which is the left parameter

It’s Not Quite That Simple

- The function `strcpy()` does absolutely **no** size checking
- This can be extremely dangerous!
- Ideally, we shouldn’t use this function
- But that leaves us in the same predicament
    - . . . Or does it?
- There is an improved version of `strcpy()`, and it is called `strncpy()`
- It looks like this

```
strncpy(word, "Hello", sizeof(word) - 1);
```

An Explanation

```
strncpy(word, "Hello", sizeof(word) - 1);
```

- The first two parameters are the same as `strcpy()`
- The third parameter is the maximum number of characters you will allow to be copied
- `sizeof()` is a function that returns the number of **bytes** of its argument
- Because a character is 1 byte in size, `sizeof()` reports back the total number of characters that a C-String can hold
- The "- 1" is necessary because we need to make sure that the null character written
  - The third parameter will not take that into account for us
  - We are **always** responsible that there is room for the null character

Other Things C-Strings Don't do Normally

- The equivalency operator == doesn't work as expected, either
- Although, in this situation, it's actually worse than assignment
- You won't get a compile error
- But it won't work as expected
- Again, the reason for this is that C-Strings are not variables, they are null character terminated arrays of characters

A Pattern Emerges

- If we want to compare two C-Strings, there is a function for that

```
char word[10] = "Hello";
char other[6] = "Hello";

if (strcmp(word, other))
    cout << "NOT EQUIVALENT\n";
else
    cout << "EQUIVALENT\n";
```

- We should observe that if strcmp returns `false`, then the two C-Strings are equivalent
- But why, though?

How `strcmp()` Works

- It utilizes a comparison method known as *lexicographical compare*
- We have hopefully observed by now that a `char` also has an integer value
- Lexicographical comparison involves looking at both C-Strings character-by-character until a mis-match is found
- At that point, one of the characters is “less-than” the other
- This leads into the next reason why `strcmp()` appears to behave weirdly
- `strcmp()` does not return a `bool`, but an integer

An Integer, You Say?

- If the integer returned is negative, then the mismatched character in the left argument is “less than” that corresponding character in the right argument
    - This because the difference between the integer values of the two characters results in a negative number if the left character is “smaller”
- The reverse holds true if `strcmp()` returns a positive value (the character in the left argument is larger than the corresponding character in the right argument)
- If the two C-Strings are equivalent, then a zero is returned
- When evaluating the equivalency to a Boolean, any non-zero value is treated as `true`, and zero is treated as `false`

Back to the Emerging Pattern

- `strcmp()` does have some safety built into it
- However, if you attempt to use it and have previously lost the null character in both strings, those safeties won't kick in
- Like strcpy, there is a safer version

```
char word[10] = "Hello";
char other[6] = "Hello";

if (strncmp(word, other, 5))
    cout << "NOT EQUIVALENT\n";
else
    cout << "EQUIVALENT\n";
```

- You could still use `sizeof()` here, but it is still your responsibility to know which string is smaller

A Few More Functions from <cstring>

Function	**Description**
`strlen(`*`srcString`*`)`	Returns integer equal to length of the C-String. The null character is **not** counted
`strcat(`*`target`*`, `*`source`*`)`	Concatenates the C-String value of *`source`* onto the end of *`target`*. **target must be large enough to hold both strings and the null character**
`strncat(`*`target`*`, `*`source`*`, `*`limit`*`)`	A safer version of `strcat()`. You are still responsible to know how many remaining characters can be written

C-Strings Really do Make Some Things Easier

- We can use `cin` to store user input directly into a C-String
    - We should be extra careful to make sure our C-String is absolutely large enough
    - It would be much safer to use `cin.get()` or `getline()`, because these functions allow us to specify a maximum size to read in
- We can `cout` a C-String directly
    - No loop necessary, it will only output the relevant part of the C-String automatically
- Convert a string to a number
    - This can serve as rudimentary way to validate input from a user, if we are expecting a number
    - Must include <`cstdlib`>
    - The functions are `atoi(`*string*`)` and `atof(`*string*`)`