## CPE 101 slides based on UW course

Lecture 19: Strings





## Overview

Concepts this lecture
- String constants
- Null-terminated array representation
- String library <strlib.h>
- String initializers
- Arrays of strings



## Chapter 9

Read Sections 9.1, 9.2, and 9.4:

9.1: String Basics

Table 9.1 for summary of common functions

9.2: String Assignment

9.4: String Comparison



## Character Data in Programs

Names, messages, labels, headings, etc.

All of these are common in computer applications

All involve characters: usually multiple characters

So far, our ability to handle these things in C is very limited



## Characters and Strings

Character constants (literals): single quotes

'a', 'A', '0', '1', '\n', ' ', 'B', 'i', 'l' , '\0'

null character



## Characters and Strings

Character constants (literals): single quotes

'a', 'A', '0', '1', '\n', ' ', 'B', 'i', 'l' , '\0'

null character

*String constants* (literals): double quotes

"Turner is a madman!"

"The answer is %.2f. \n"



## String Representation (1)

Strings are stored in char arrays

Programming convention: a null character '\0' is stored at the end

| string | representation |
|---|---|
| "sample" | s a m p l e \0 |



## String Representation (2)

'\0' included in string constants automatically

Programmer must take pains to be sure it is present elsewhere when needed

s a m p l e \0

## String Representation (3)

Character arrays holding strings must have room for '\0' following the actual data

The empty string "" occupies 1 char

Character and string constants are not the same:
'x' and "x" are different. How?

s a m p l e \0

## Why the Null ?

This '/0' is an interesting solution to the reading or writing to the character array problem, how?

(Think about the old sentinel controlled loops….)

N-10

## String Operations

Common needed operations:
- Copy (assignment)
- Compare
- Find length
- Concatenate (combine strings)
- I/O

Unfortunately...

| s | a | m | p | l | e | \0 |
|---|---|---|---|---|---|---|

## What You Can't Do

Always remember: Strings *are arrays*

They have the limitations of arrays

Can't assign one string to another with =

Can't compare strings with ==, <=

But there are library functions to help do such things

| s | a | m | p | l | e | \0 |
|---|---|---|---|---|---|---|

## String Library: <string.h>

Standard C includes a library of string functions
- use *#include <string.h>*

Library functions:
- Require proper null-terminated ('\0') strings as arguments
- Produce null-terminated strings as results (usually!!!)

| s | a | m | p | l | e | \0 |
|---|---|---|---|---|---|---|

## String Length: *strlen*

*strlen* returns the length of its string argument
- Does not count the null '\0' at the end (why?)

Examples:
- The length of "A" is 1
- The length of "" is 0

```
k = strlen("null-terminated string");
```

stores 22 in *k*

N-14

## Size (of array) vs. Length of strings

What is the diff? Remember this!

length is # of chars not incl. '\0'

size of array includes (at least) the '\0' in addition to that.

N-15

## A *strlen* implementation

```
/*
 *  return the length of string  s, i.e.,
 *   number of characters before terminating '\0',
 *   or equivalently, index of first '\0'.
 */
int strlen( char s[ ] )
{
    int n = 0;
    while ( s[n]  != '\0')
       n = n + 1 ;
    return n;
}
```

N-16

## String Assignment: *strcpy*

```
strcpy(dest, source);
```

Copies characters from *source* to *dest*
- Copies up to, and including the first '\0' found
- Be sure that *dest* is large enough to hold the result!

N-17

## String Assignment: Examples

```
#include <string.h>
...
char medium[21] ;
char big[1000] ;
char small[5] ;
strcpy(medium, "Four score and seven" ) ;
```

medium: `Four score and seven\0`

N-18

## String Assignment: Examples

```
char medium[21 ];
char big[1000] ;
char small[5] ;

strcpy(big, medium) ;
strcpy(big, "Bob") ;
```



## String Assignment: Examples

```
char medium[21 ];
char big[1000] ;
char small[5] ;

strcpy(big, medium) ;
strcpy(big, "Bob") ;
```

big: `Four score and seven\0?????...`



## String Assignment: Examples

```
char medium[21 ];
char big[1000] ;
char small[5] ;

strcpy(big, medium) ;
strcpy(big, "Bob") ;
```

big: `Four score and seven\0?????...`

big: `Bob\0 score and seven\0?????...`

## String Assignment Dangers

```
char medium[ 21];
char big[1000] ;
char small[5] ;

strcpy(small, big) ;
strcpy(small, medium) ;    /* looks like trouble... */
```



## String Assignment Dangers

```
char medium[ 21];
char big[1000] ;
char small[5] ;

strcpy(small, big) ;
strcpy(small, medium) ;    /* looks like trouble... */
```

small: `Bob\0?`



## String Assignment Dangers

```
char medium[ 21];
char big[1000] ;
char small[5] ;

strcpy(small, big) ;
strcpy(small, medium) ;    /* looks like trouble... */
```

small: `Bob\0?`

small: `Four score and seven\0`



## What went wrong?

NOTE AND EMPHASIZE -

- writing off the end of the array is NOT a compiler (syntax) error!
- It does NOT necessarily give a runtime error either, the program may run with corrupted data!
- (subtle and very difficult bugs may result in your program output!)



## A *strcpy* implementation

```
/* copy source string into dest, stopping with '\0' */
void strcpy(char dest[ ], char source[ ])
{
    int i = 0;
    while (source[ i ]  != '\0') {
       dest[ i ] = source[ i ] ;
       i ++;
    }
    dest[ i ] = '\0' ;
}
/* why write the '\0' outside the loop? */
```



## String Concatenation: *strcat*

To append means to place one string directly after another

"geek" appended to "csc" should result in "cscgeek"

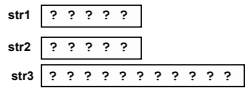
*strcat(dest, source);*

Appends characters from *source* to *dest*

- Copy is stored starting at first '\0' in *dest*
- Copies up to, and including the first '\0' in *source*
- Be sure that *dest* is large enough!

## Using *strcat* (1)

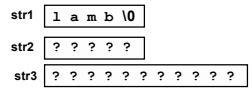
```
#include <string.h>
...
char str1[5] , str2[5] , str3[11];

strcpy(str1, "lamb");
strcpy(str2, "chop");
```






## Using *strcat* (1)

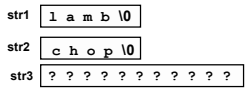
```
#include <string.h>
...
char str1[5] , str2[5] , str3[11];

strcpy(str1, "lamb");
strcpy(str2, "chop");
```






## Using *strcat* (1)

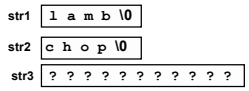
```
#include <string.h>
...
char str1[5] , str2[5] , str3[11];

strcpy(str1, "lamb");
strcpy(str2, "chop");
```






## Using *strcat* (2)

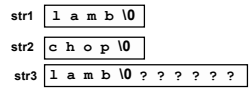
```
strcpy(str3, str1);
strcat(str3, str2);
```






## Using *strcat* (2)

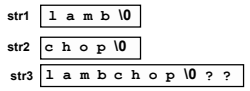
```
strcpy(str3, str1);
strcat(str3, str2);
```






## Using *strcat* (2)

```
strcpy(str3, str1);
strcat(str3, str2);
```






## String Comparison: *strcmp*



## String Comparison: *strcmp*

***strcmp(s1, s2);***

Compares *s1* to *s2* and *returns an int* describing the comparison

*Negative* if *s1* is less than *s2*
*Zero* if *s1* equals *s2*
*Positive* if *s1* is greater than *s2*
*(why does this make sense?)*



## Comparing Strings

***strcmp*** compares corresponding characters until it finds a mismatch.

- "lamb" is less than "wolf"
- "lamb" is less than "lamp"
- "lamb" is less than "lambchop"
- '\0' ought to come first or last?
- (How natural is all this in C? Recall ascii … and also Unicode.)

## Using *strcmp* (1)

***Don't treat the result of strcmp as a Boolean*!**

**Test the result as an integer**

```
if (strcmp(s1,s2) == 0)
        printf("same\n");
```

**(what integers represent True and False in C? Now you see the problem? See next slide.)**



## Using *strcmp* (2)

**If you treat the result of *strcmp* as a Boolean, it probably won't do what you want**

```
if (strcmp(s1,s2))
        printf("yikes!");
```

**prints *yikes* if s1 and s2 are *different!***



## String I/O

***scanf* and *printf* can read and write C strings**
- **Format code is %s**
- **'\0' termination handled properly**

**Be sure there's enough space for data plus '\0' on input!**

```
#define MAX_INPUT 2000
char buffer [MAX_INPUT];
…
scanf("%s", buffer);
/* no '&' needed here, why? */
```



## Security

**Notice that buffer could be overwritten, scanf does not check bounds.**

**An attacker can write his (her!) own code in the memory locations after buffer and run their own code on your system.**



## Many Functions in <*string.h*>

| | |
|---|---|
| ***strcat, strncat*** | **concatenation** |
| ***strcmp, strncmp*** | **comparison** |
| ***strtod, strtol, strtoul*** | **conversion** |

**Lots of others: look in Appendix B.**

**Related useful functions in <*ctype.h*>**
- **operations on a single char:**
- **convert case, check category, etc.**

**See a textbook or reference manual**



## Many Functions in <*string.h*>

| | |
|---|---|
| ***strcat, strncat*** | **concatenation** |
| ***strcmp, strncmp*** | **comparison** |
| ***strtod, strtol, strtoul*** | **conversion** |

**Lots of others: check your favorite reference.**

**Related useful functions in <*ctype.h*>**
- **operations on a single char:**
- **convert case, check category, etc.**



## Using Libraries of Functions

**To use strings effectively in C, use functions from string.h**

**Using libraries is very typical of C programming**
- **ANSI C standard libraries such as stdio.h, string.h, ctype.h**
- **Application-specific libraries: (thousands of them exist)**

**You can't be an effective programmer without being able to quickly master new libraries of functions, but ….**



## String functions note

**Be sure you understand how to use functions before utilizing them in your programs (labs, etc.)**



## Bonus: String Initializers

## Bonus: String Initializers

```
char pet[5] = { 'l', 'a', 'm', 'b', '\0' } ;

char pet[5] ;
pet[0] = 'l' ;  pet[1] = 'a' ;  pet[2] = 'm' ;
      pet[3] = 'b' ; pet[4] = '\0' ;

char pet[5] = "lamb" ; /* careful with length! */
```

all equivalent

But not:
```
char pet[5];
pet = "lamb" ;     /* No array assignment in C */
```
Remember that initializers are not assignment statements!

And, remember the array length for "lamb" is 5, need the null character at the end!

N-46

## Bonus: Arrays of Strings

```
char month[12][10] = {
    "January",
    "February",
    ...
    "September",          /* longest month: 9 letters */
    ...
    "December" } ;
...
printf ("%s is hot \n", month[7] );     /* August */
```

N-47

## Strings Summary

Definition: Null-terminated array of char

Strings are not a unique type in C (a special kind of an array…)
They share most limitations of arrays
*scanf/printf*: %s
<string.h> library functions
- Assignment: *strcpy*
- Length: *strlen*
- *strcat* and many others

Major Pitfall: overrunning available space

N-48