CSE 142
Computer Programming I

Strings

… or, “the ltl prgrmr wh cld.”

Overview

Concepts this lecture

- String constants
- Null-terminated array representation
- String library <string.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
- 9.3: String Concatenation
- 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

String constants (literals): double quotes

"Bill is very rich"

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

String Representation

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

’\0’ in Strings

’\0’ is not included in strings automatically

’\0’ is included in string constants automatically

Programmer must take pains to be sure ’\0’ is present elsewhere when needed

s	a	m	p	l	e	\0

Leaving Room for ’\0’

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

What kind of error does this bode?

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

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

But… they don’t check bounds for you! Why not?

String Length: *strlen*

strlen returns the length of its string argument

Does not count the null ’\0’ at the end

Examples:

The length of "A" is 1

The length of "" is 0

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

stores 22 in ***k***

Why strlen?
str ➔ string
len ➔ length
programmers ➔ lazy

T-12

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;
     /* just a few statements are needed here...
     can you figure them out?
     */
     return n;
}
```

T-13

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;
}
```

T-14

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!

T-15

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`

T-16

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?????...`

T-17

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`

T-18

T

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' ;
}
```

Appending and Concatenation

To append means to place one string directly after another

"chop" appended to "lamb" should result in "lambchop"

Also referred to as concatenation

String Concatenation: *strcat*

<string.h> function:

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");
```

```
str1  l a m b \0
str2  c h o p \0
str3  ? ? ? ? ? ? ? ? ? ? ?
```

Using *strcat* (2)

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

```
str1  l a m b \0
str2  c h o p \0
str3  l a m b c h o p \0 ? ?
```

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*

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"

T-25

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");
```

T-26

Steve's note: I did this wrong ***this morning****!*
BEWARE!

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!*

T-27

String I/O

scanf and *printf* can read and write C strings
 Format code is %s

printf assumes '\0' is present

scanf will automatically insert '\0' at the end
Be sure the array has room for it!

T-28

Spot the Security Hole

```
#define MAX_INPUT 200
char buffer [MAX_INPUT];
…
scanf("%s", buffer);
```

Never happen? Doesn't matter?
Ever heard of the Internet Worm?

T-29

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 (to upper or lower)
check category (is char a number, etc.)
many others

T-30

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, math.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

T-31

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" ;
```

all equivalent

But not:

```
char pet[5];
pet = "lamb" ;      /* No array assignment in C */
```

Remember that initializers are not assignment statements!

T-32

Bonus: Arrays of Strings

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

T-33

Strings Summary

Definition: Null-terminated array of char

Strings are not fully a type of C
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

T-34

QOTD: Name the Player

Strings are often used for names of people and things.

As each player joins a game, *how would you choose a name for the player?*
Each player should have a different, pronounceable, memorable name.

T-35

T