

CSE 142 Programming I

Strings

© 2000 UW CSE

5/19/00

N-1

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

5/19/00

N-2

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

5/19/00

N-3

Characters and Strings

- **Character** constants (literals): single quotes
 - 'a', 'A', '0', '1', '\n', ' ', 'B', 'i', 'l', '\0' ← the null character
- **String** constants (literals): double quotes
 - "Bill"
 - "Mary had a little %c%c%c%c. \n"
- **Character** variables:

```
char va = 'l', vb = 'a', vc = 'm', vd = 'b';  
printf("Mary had a little %c%c%c%c.\n", va, vb, vc, vd);
```

5/19/00

N-4

Strings

- Strings: arrays of char

```
char pet[5] = { 'l', 'a', 'm', 'b', '\0' };  
printf("Mary had a little %s. \n", pet);
```
- More accurate: **null-terminated array of char**

```
pet: [ 'l' | 'a' | 'm' | 'b' | '\0' ]  
      ↑           ↑  
    pet[0]       pet[4]
```
- Strings are not quite a full-fledged type in C
- Programmer must take pains to ensure '\0' is present

5/19/00

N-5

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';
```

all equivalent

```
char pet[5] = "lamb";  
char pet[] = "lamb";
```

But Not:

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

5/19/00

N-6

Things You Can and Can't Do

- You **can't**
use = to assign one string variable to another
(use library functions *strcpy* etc.)
- You **can't**
use == to directly compare strings (use
library functions *strcmp* etc.)
- You **can't**
have a string as a function return type
- You **can**
directly *scanf* or *printf* strings (use `%s`)

N-7

Do-It-Yourself String Assignment

```
char str1[10], str2[ ] = "Saturday";
int i;
/* can't do: str1 = str2; */
/* can do: */
i = 0;
while (str2[i] != '\0') {
    str1[i] = str2[i]; i = i + 1;
}
str1[i] = '\0';
```

5/19/00

N-8

String Assignment with *strcpy*

```
/* strcpy is defined in string.h (although implementation details differ)
   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 = i + 1;
    }
    dest[i] = '\0';
}
```

5/19/00

N-9

String Assignment: Dangers

```
#include <string.h>
...
char medium[ ] = "Four score and seven";
char big[1000];
char small[5];
strcpy(big, medium);
strcpy(big, "Bob");
strcpy(small, big);
strcpy(small, medium); /* looks like trouble... */
```

5/19/00

N-10

strcpy results

```
medium: Four score and seven\0
big:    Four score and seven\0?????...
big:    Bob\0 score and seven\0?????...
small:  Bob\0?
small:  Four score and seven\0
```

5/19/00

N-11

String Length: *strlen* (in *string.h*)

```
/* * 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);
}
```

5/19/00

N-12

Length Examples

```
#include <string.h>    /* defn of strlen, strcpy*/
...
char pet[] = "lamb";
int len1, len2, len3, len4, len5;

len1 = strlen(pet);
len2 = strlen("wolf");
len3 = strlen("");
len4 = strlen("Help\n");
strcpy(pet, "cat");
len5 = strlen(pet);
```

0	1	2	3	4	5	6
l	a	m	b	\0		
w	o	l	f	\0		
\0						
H	e	l	p	\n	\0	
c	a	t	\0	\0		

5/19/00 N-13

Example Use of strlen

```
#include <string.h>    /* defn of strlen, strcpy*/
char small[5];
...
if ( strlen(medium) <= 4 )
    strcpy(small, medium);

else
    printf ("String is too long to copy.\n");
```

5/19/00 N-14

String Concatenation

```
#include <string.h>
...
char str1[] = "lamb", str2[] = "chop";
char str3[11];

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

/* strcat(s1, s2) – make a copy of s2 at the end of s1. */
```

5/19/00 N-15

strcat results

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

5/19/00 N-16

Comparing Strings

str_1 is less than *str_2* if *j* is the first position where they differ and *str_1[j] < str_2[j]*.

"lamb" is less than "wolf" *j* = 0, 'l' < 'w'

"lamb" is less than "lamp" *j* = 3, 'b' < 'p'

"lamb" is less than "lambchop" *j* = 4, '\0' < 'c'

5/19/00 N-17

String Comparison Errors

str1 = *str2*; **Syntax "error"**

if (*str1* == *str2*)... **No syntax error (but almost surely a logic error)**

if (*str1* < *str2*)... **Likewise**

5/19/00 N-18

Correct String Comparison

```
/* function strcmp in <string.h> */
int strcmp(char str_1[], char str_2[]);
```

The integer returned is:

negative if *str_1* less than *str_2*
zero if *str_1* equals *str_2*
positive if *str_2* less than *str_1*

Common errors: if (strcmp(str1, str2))... means "if they ARE equal"

5/19/00 N-19

String Input and Output

- **scanf** with "%s"
 - Skips initial whitespace
 - Inserts '\0' at next whitespace
 - **Danger:** no length check
 - a malicious user could cause harm

```
char in_string[10];
scanStatus = scanf ("%s", in_string);
```

no &

- **printf** with "%s"

5/19/00 N-20

Do-It-Yourself Whole Line Input

```
char line [LENGTH + 1];
int i, scanStatus;

/* read input characters into line until end of input line
   reached or all available space in line used */
i = 0;
scanStatus = scanf ("%c", &line[i]);
while (1 == scanStatus && i < LENGTH && line[i-1] != '\n') {
    i++;
    scanStatus = scanf ("%c", &line[i]);
}
line [i] = '\0'; /* is this a bug? */
```

5/19/00 N-21

Arrays of Strings

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

5/19/00 N-22

Reading and Printing Strings

```
char name [NUM_NAMES] [MAX_NAME + 1];
int age [NUM_NAMES], i;
for (i = 0; i < NUM_NAMES; i = i + 1)
{
    scanf ("%s %d", name[i], &age[i]);
    printf ("%s %d \n", name[i], age[i]);
}
```

5/19/00 N-23

Many Functions in <string.h>

<i>strcat, strncat</i>	concatenation
<i>strcmp, strncmp</i>	comparison
<i>strtod, strtol, strtoul</i>	conversion

Lots of others: see Appendix B.

Related useful functions in <ctype.h>
operations on a single char:
convert case, check category, etc.
See textbook Table 9.3 and Appendix B

5/19/00 N-24

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: *cpanel.h*, *GP142.h*, etc. (thousands of them exist)
- *You can't be an effective programmer without being able to quickly master new libraries of functions*

5/19/00 N-25

Strings Summary

- Definition: Null-terminated array of char
- A **convention**, not a first-class citizen
 - E.g., no string assignment or compare in the C language itself
- *scanf/printf*: %s
- <string.h> library functions
 - Assignment: *strcpy*, *strncpy*
 - Length: *strlen*
 - reminder: length of contents, not container
 - *strcat* and many others
- **Major Pitfall**: overrunning available space

5/19/00 N-26