CSE / ENGR 142 Programming I

Strings

Chapter 9

Read Sections 9.1, 9.2, and 9.4:

- **9.1: String Basics**
- **9.2: String Assignment**
- **9.4: String Comparison**

Characters and Strings

- **Character constants (single quote):**
  - **'a', 'A', '0', '1', '\n', ' ', '\0', ...**
- **String constants (double quote):** null
  - **"Mary had a little %c%c%c%c. \n"**
- **Character variables:**

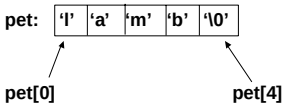
```
char va = 'I', vb = 'a', vc = 'm', vd = 'b';
printf("Mary had a little %c%c%c%c.\n",va,vb,vc,vd);
```


Strings

- **Strings: arrays of char**

```
char pet[5] = { 'I', 'a', 'm', 'b', '\0' };
printf("Mary had a little %s . \n", pet);
```

- **More accurate: *null-terminated array of char***



Initializing Strings

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

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

char pet[5] = "lamb" ;

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

all equivalent

But Not:

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


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*)**

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';
```


String Assignment with *strcpy*

```
/* strcpy is defined in string.h:
   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' ;
}
```


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... */
```


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`

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) ;
}
```


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		

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") ;
```


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.  */
```


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 ? ?

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'**

String Comparison Errors

str1 = str2; **Syntax error**

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

if (str1 < str2)… **Likewise**

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:
strcmp **is not a Boolean function!**

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];
scanf ("%s", in_string);
```

no &

- *printf* with "%s"

Do-It-Yourself Whole Line Input, With Length Checking

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


Arrays of strings

```
char month[12][10] = {
    "January",
    "February",
    ...
    "September",      /* longest month: 9 letters */
    ...
    "December" } ;

printf ("%s is hot \n", month[7] );     /* August */
```


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]) ;

}
```

no &

Many Functions in *<string.h>*

strcat, strncat	concatenation
strcmp, strncmp	comparison
strtod, strtol, strtoul	conversion

Lots of others: see Table 9.1 and Appendix B.

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

Strings Summary

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