COP 3223: C Programming
Spring 2009

Strings In C – Part 1

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cop3223/spr2009/section1

School of Electrical Engineering and Computer Science
University of Central Florida

Fundamentals of Strings And Characters In C

- So far in this course, we've done very little with characters, although we have used them a little bit in a few of the programs in the notes. The reason for this is that, by themselves, characters are not terribly useful in many programs.
- However, strings of characters, or simply strings, are quite useful in many applications.
- A string is a series of characters treated as a single unit.
- Some programming languages, such as Java, define a special string type, C does not have a special string data type.
- A string in C is an array of characters ending with the null character (`'\0'` – used to mark the end of the string). A string is accessed via a pointer to the first character in the string.

Fundamentals of Strings And Characters In C

- Thus, in C, it is appropriate to say that a string is a pointer. In fact, it is a pointer to the string's first character. The first character is in index position 0. In this sense, strings are like arrays, because an array is also a pointer to its first element.
- In C it is possible to define both string literals and string variables.
- A string literal (also called a string constant) is a sequence of characters enclosed within double quotes, such as "`Please enter an integer value:`".
- We've used string literals quite a lot in our programs this semester. String literals commonly appear as format strings in calls to `printf` or `scanf`.
- What is actually being passed to `scanf` or `printf` when a string literal is sent to the function?

Fundamentals of Strings And Characters In C

- String literals are treated as arrays by the C compiler, so when the C compiler encounters a string literal of length `n` in a program, it sets aside `n+1` contiguous bytes of memory for the string. The `n` locations contain the characters in the string and the `n+1` location contains the null character (`'\0'`).

The null character ('\0') is a byte whose bits are all zero (i.e., 00000000), so it is represented by the \0 escape sequence.

Do not confuse the null character ('\0') with the zero character ('0'). The null character has the ASCII code 0, while the zero character has the ASCII code 48.

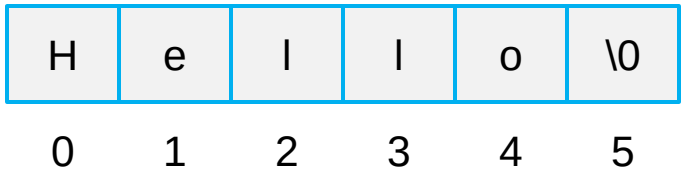
Equivalents: ASCII Name	C escape sequence	meaning
nul	\0	null byte
bel	\a	bell character
bs	\b	backspace
ht	\t	horizontal tab
np	\f	form feed
nl	\n	new line
cr	\r	carriage return

7 bit ASCII Table ($2^7 = 128$)

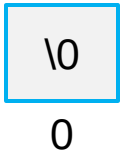
Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex
(nul)	0	0000	0x00	(sp)	32	0040	0x20	@	64	0100	0x40	`	96	0140	0x60
(soh)	1	0001	0x01	!	33	0041	0x21	A	65	0101	0x41	a	97	0141	0x61
(stx)	2	0002	0x02	"	34	0042	0x22	B	66	0102	0x42	b	98	0142	0x62
(etx)	3	0003	0x03	#	35	0043	0x23	C	67	0103	0x43	c	99	0143	0x63
(eot)	4	0004	0x04	$	36	0044	0x24	D	68	0104	0x44	d	100	0144	0x64
(enq)	5	0005	0x05	%	37	0045	0x25	E	69	0105	0x45	e	101	0145	0x65
(ack)	6	0006	0x06	&	38	0046	0x26	F	70	0106	0x46	f	102	0146	0x66
(bel)	7	0007	0x07	'	39	0047	0x27	G	71	0107	0x47	g	103	0147	0x67
(bs)	8	0010	0x08	(	40	0050	0x28	H	72	0110	0x48	h	104	0150	0x68
(ht)	9	0011	0x09	)	41	0051	0x29	I	73	0111	0x49	i	105	0151	0x69
(nl)	10	0012	0x0a	*	42	0052	0x2a	J	74	0112	0x4a	j	106	0152	0x6a
(vt)	11	0013	0x0b	+	43	0053	0x2b	K	75	0113	0x4b	k	107	0153	0x6b
(np)	12	0014	0x0c	,	44	0054	0x2c	L	76	0114	0x4c	l	108	0154	0x6c
(cr)	13	0015	0x0d	-	45	0055	0x2d	M	77	0115	0x4d	m	109	0155	0x6d
(so)	14	0016	0x0e	.	46	0056	0x2e	N	78	0116	0x4e	n	110	0156	0x6e
(si)	15	0017	0x0f	/	47	0057	0x2f	O	79	0117	0x4f	o	111	0157	0x6f
(dle)	16	0020	0x10	0	48	0060	0x30	P	80	0120	0x50	p	112	0160	0x70
(dc1)	17	0021	0x11	1	49	0061	0x31	Q	81	0121	0x51	q	113	0161	0x71
(dc2)	18	0022	0x12	2	50	0062	0x32	R	82	0122	0x52	r	114	0162	0x72
(dc3)	19	0023	0x13	3	51	0063	0x33	S	83	0123	0x53	s	115	0163	0x73
(dc4)	20	0024	0x14	4	52	0064	0x34	T	84	0124	0x54	t	116	0164	0x74
(nak)	21	0025	0x15	5	53	0065	0x35	U	85	0125	0x55	u	117	0165	0x75
(syn)	22	0026	0x16	6	54	0066	0x36	V	86	0126	0x56	v	118	0166	0x76
(etb)	23	0027	0x17	7	55	0067	0x37	W	87	0127	0x57	w	119	0167	0x77
(can)	24	0030	0x18	8	56	0070	0x38	X	88	0130	0x58	x	120	0170	0x78
(em)	25	0031	0x19	9	57	0071	0x39	Y	89	0131	0x59	y	121	0171	0x79
(sub)	26	0032	0x1a	:	58	0072	0x3a	Z	90	0132	0x5a	z	122	0172	0x7a
(esc)	27	0033	0x1b	;	59	0073	0x3b	[	91	0133	0x5b	{	123	0173	0x7b
(fs)	28	0034	0x1c	<	60	0074	0x3c	\	92	0134	0x5c	|	124	0174	0x7c
(gs)	29	0035	0x1d	=	61	0075	0x3d	]	93	0135	0x5d	}	125	0175	0x7d
(rs)	30	0036	0x1e	>	62	0076	0x3e	^	94	0136	0x5e	~	126	0176	0x7e
(us)	31	0037	0x1f	?	63	0077	0x3f	_	95	0137	0x5f	(del)	127	0177	0x7f

Fundamentals of Strings And Characters In C

- Consider the string literal "`Hello`". This will be stored in an array of six characters as shown:



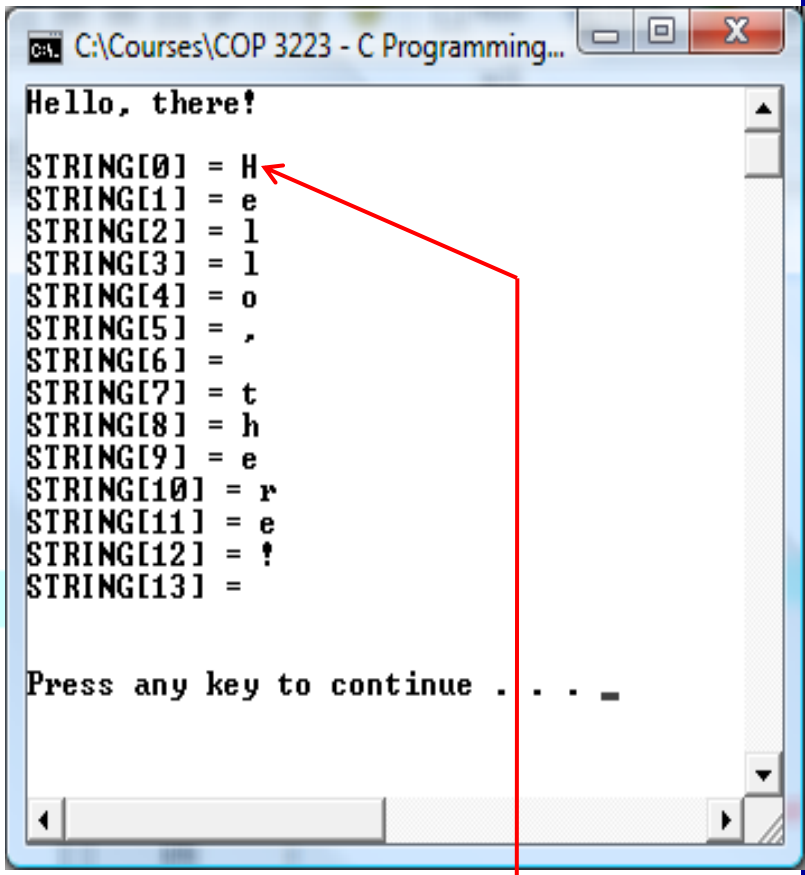
- An empty string literal, denoted as: "", will be stored as a single null character.



- Since a string literal is stored as an array, the compiler treats it as a pointer of type `char *`. Both `printf` and `scanf`, expect a value of type `char *` as their first argument.

read a character.c | string literal example.c | mystrlib[1].c

```
1  //Strings In C - Part 1 - sample using string literal
2  //March 17, 2009   Written by: Mark Llewellyn
3
4  #include <stdio.h>
5  #define STRING "Hello, there!"
6
7  int main()
8  {
9      int i;  //loop control
10
11     printf(STRING);
12     printf("\n\n");
13     for (i = 0; i < 14; ++i){
14         printf("STRING[%d] = %c\n", i, STRING[i]);
15     }
16
17     printf("\n\n");
18     system("PAUSE");
19     return 0;
20 }//end main function
21
```

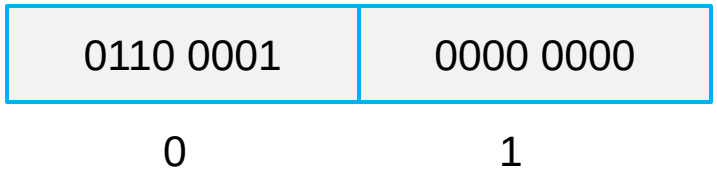


The address of `STRING[0]` is passed to `printf` when the call is made.

Fundamentals of Strings And Characters In C

- It is important to understand the difference between a string literal and a character constant.
- In C, `'a'` and `"a"`, are two very different beasts.
- `'a'` is a character constant, which occupies one byte in memory. The value stored in the memory location is $(97)_{10} = (141)_8 = (01100001)_2$.

- `"a"` is a string literal, which will be stored in an array containing two locations, the first containing the ASCII code for the 'a' character and the second containing the null character.



String Variables In C

- While string literals are an important concept in C and understanding how they work is important, from an application point of view, string variables are much more important and interesting.

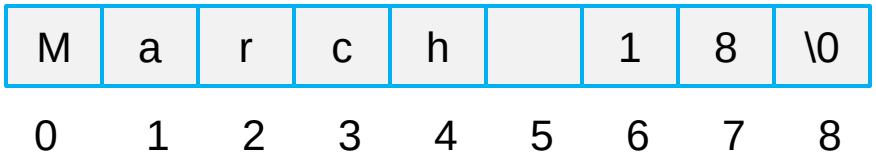
Since character strings always terminate with the null character, this means that you must declare the size of arrays that are used to hold strings to be one larger than the longest possible string of characters that the array might contain in order to allow room for the terminating null character. You'll forget this so I'll remind you again later ☺ !

- The length of a string of characters is determined by the position of the null character and not the size of the allocation of the array. This means that there is no quicker way to determine the length of a string than a character by character search for the null character.

Initializing A String Variable

- A string variable can be initialized at the same time it is declared, just as with other variables in C.
- Thus, the declaration: `char date[9] = "March 18";`

  would produce:



- Although the declaration and initialization makes "March 18" appear as a string literal, it's not. Instead, C views it as an abbreviation for an array initializer, much in the same way that we have done for integer arrays. In other words, we could have written the declaration as:

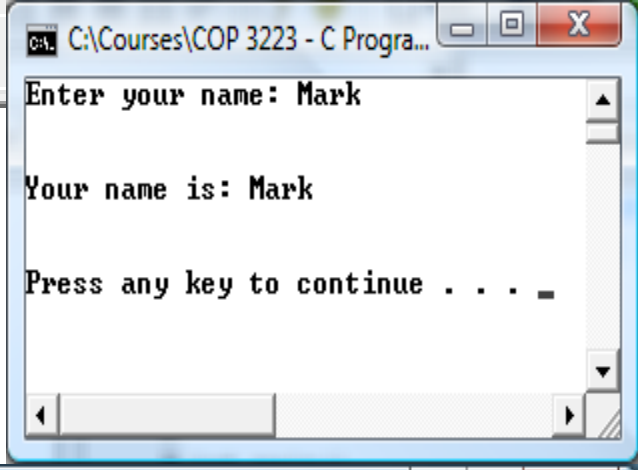
```
char date[9] = {'M', 'a', 'r', 'c', 'h', ' ', '1', '8', '\0'};
```

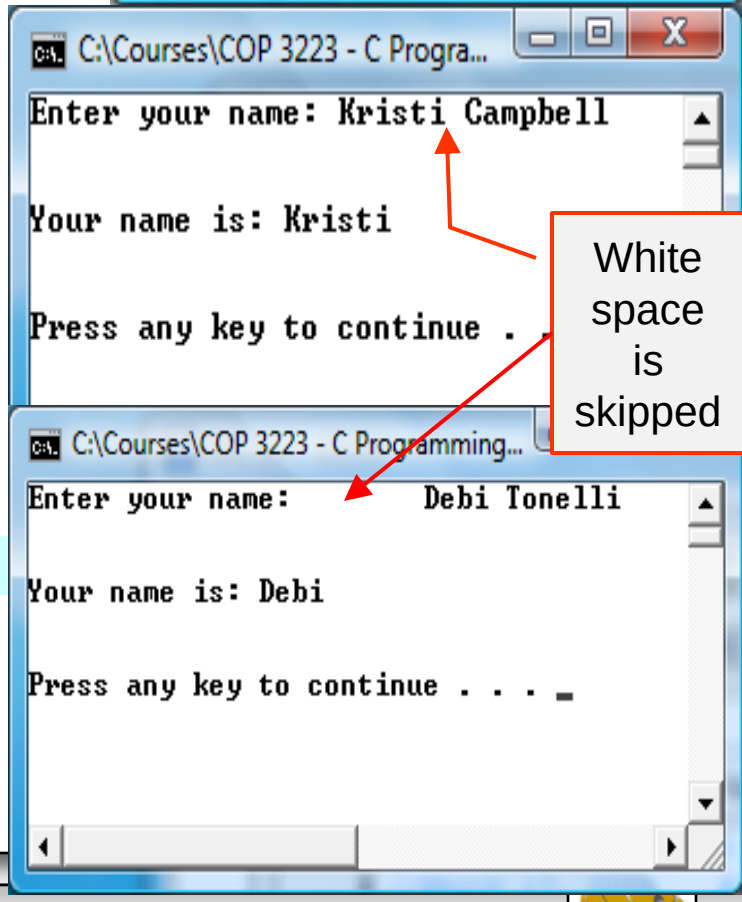

Reading Strings Using `scanf`

- Strings can be read into a character array using `scanf` in much the same way that integers can be read into an integer array using `scanf`.
- The conversion specifier for strings in C is `%s`.
- Assuming we had declared `char name[10];`. We could do the following: `scanf("%s", name);`
- Since `name` is a character array and hence a pointer, the address operator (&) is not needed on the `name` variable.
- The `scanf` function ignores leading white-space and the input string is terminated by any white-space.
- The program on the following page illustrates using scanf to read in a string of characters and print them out.

read a character.c | string literal example.c | input and reversing a name.c | using scanf to read strings.c

```c
//Strings in C - Part 1 - using scanf to read a string
//March 17, 2009    Written by: Mark Llewellyn

#include <stdio.h>
#define MAX 10

int main()
{
    char name[MAX]; //string to hold a name

    printf("Enter your name: ");
    scanf("%s", name);
    printf("\n\nYour name is: %s\n", name);

    printf("\n\n");
    system("PAUSE");
    return 0;
}//end main function
```





Character Arrays (Strings) Versus Character Pointers

- Consider the following two declarations:

```
char date[] = "March 18";

char *date = "March 18";
```

- The first declares `date` to be an array of characters (a string). The second declares `date` to be a pointer.
- Since arrays and pointers are so closely related in C, either version of `date` can be used as a string. Any function that is expecting to be passed a character array or character pointer will accept either version of `date` as an argument.
- Do not make the mistake of thinking that both versions of date are interchangeable. They are not!

Character Arrays (Strings) Versus Character Pointers

- There are significant differences between these two declarations:

```
char date[] = "March 18";

char *date = "March 18";
```

- In the array version (the first one), the characters stored in `date` can be modified, like the elements in any array. In the pointer version (the second one), `date` points to a string literal, i.e., constant and as such cannot be modified.
- In the array version, `date` is an array name. In the pointer version, `date` is a variable that can be made to point to other strings during the execution of a program.

Character Arrays (Strings) Versus Character Pointers

- The declaration:

```
char *ptr;
```

causes the compiler to set aside enough space for a pointer that will reference a character, not a string of characters. If we want `ptr` to reference a string of characters, we'll have to do it explicitly as in:

```
char aString[10], *ptr;

. . .

ptr = aString;
```

once the second line is executed, `ptr` will now point to (contain the address of) the first character in `aString`.

Character Arrays (Strings) Versus Character Pointers

- Is this ok?

```
char *ptr;
ptr[0] = 'H';
ptr[1] = 'i';
ptr[2] = '\0';
```

Answer: No! Since `ptr` has not been initialized, it is basically not pointing to any location in memory. The behavior of such an operation will be unpredictable, but not correct.

Fundamentals of Strings And Characters In C

- We'll be examining many different string handling functions in standard libraries in C over the next few days, as well as developing some of our own string handling functions. For now, we'll continue to use `scanf` and `printf` for strings.
- The example program on the next page uses a function to count the number of valid characters in a string read in from the keyboard.

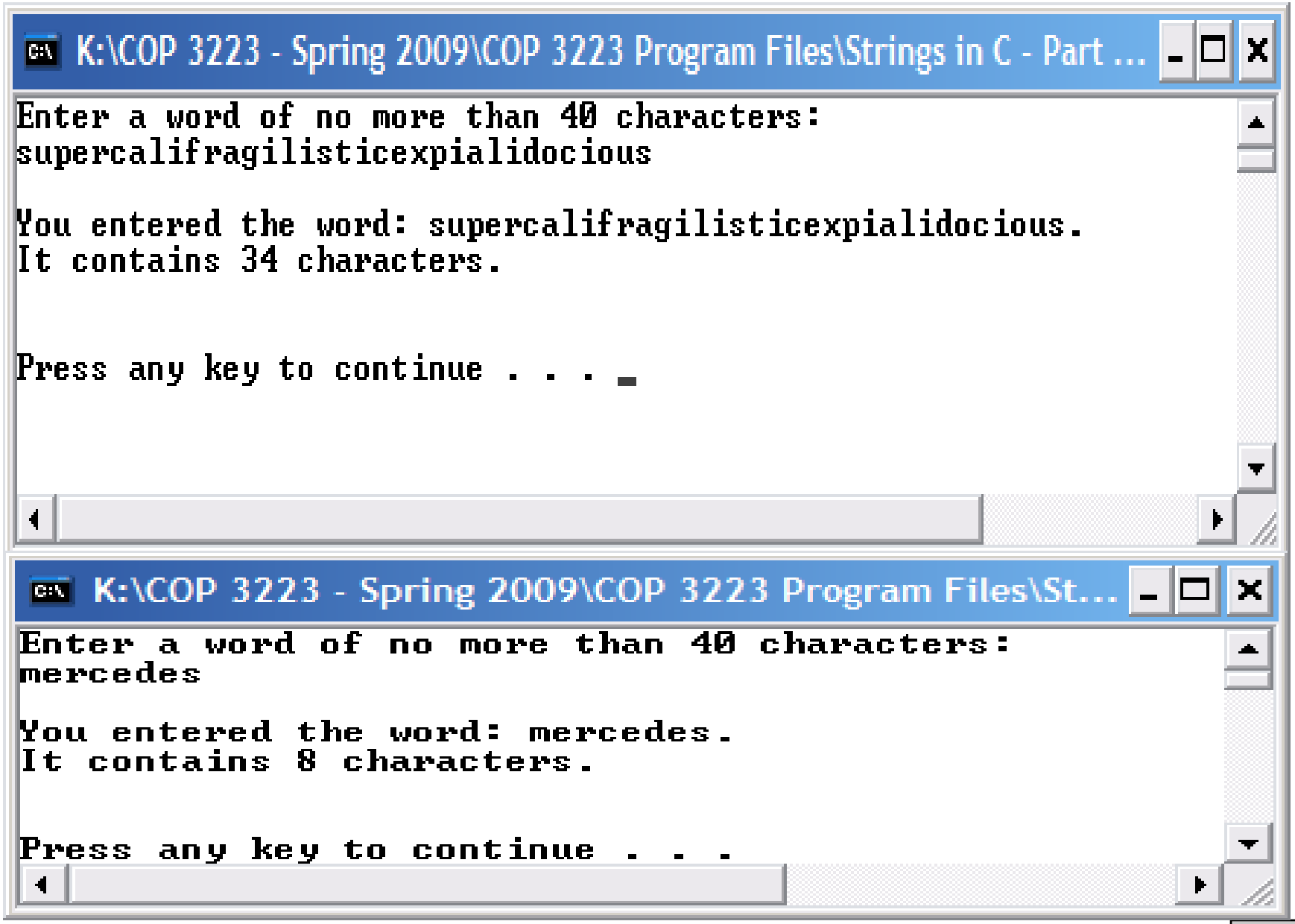
As an almost total aside, the longest non-coined, non-technical word in the English language is: antidisestablishmentarianism which contains 28 letters. The longest word in the Oxford dictionary is: Pseudopseudohypoparathyroidism at 30 letters. Although its Welsh not English, one of the longest names of a place in the world is the 58-character name:
Llanfairpwllgwyngyllgogerychwyrndrobwllllantvsiliogogogoch
which is the famous name of a town on Anglesey, and island of Wales. The longest technical word is: methionylthreon…isoleucine, the largest known protein (consisting of 34,350 amino acids) more commonly known at Titin at 189,819 letters.

determining the length of a word.c

```c
#include <stdio.h>
#define MAX_LENGTH 40

//function stringLength determines the number of valid characters in a string
int stringLength(char *aString)
{
    int index = 0;  //a counter

    while (aString[index] != '\0') {
          ++index;
    }//end while stmt
    return index;
}//end stringLength function

int main()
{
    char word[MAX_LENGTH];  //a word entered by the user
    int length;  //the number of valid characters in the string

    printf("Enter a word of no more than 40 characters:\n");
    scanf("%s", word);
    printf("\n");
    length = stringLength(word);
    printf("You entered the word: %s.\nIt contains %d characters.\n", word, length);

    printf("\n\n");
    system("PAUSE");
    return 0;
}//end main function
```

Resources | Compile Log | Debug | Find Results

Insert | Ready.


K:\COP 3223 - Spring 2009\COP 3223 Program Files\Strings in C - Part ...
Enter a word of no more than 40 characters:
supercalifragilisticexpialidocious
You entered the word: supercalifragilisticexpialidocious.
It contains 34 characters.
Press any key to continue . . .
K:\COP 3223 - Spring 2009\COP 3223 Program Files\St...
Enter a word of no more than 40 characters:
mercedes
You entered the word: mercedes.
It contains 8 characters.
Press any key to continue . . .

Practice Problems

1. Write a program that reads in two strings and then determines if the strings are the same or not.

2. Write a program that uses the string length function from the example program on page 18 in conjunction with another function which reverses the order of the characters in the string. Thus the input string: `hello` would be returned as `olleh`.