

COP 3223: C Programming Spring 2009

Strings In C – Part 2

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



Alternatives To `scanf` And `printf` For Strings

- The `scanf` function ignores leading white space and terminates its read with the first white space character it finds (`scanf` always appends a null character to the end of the string that has been read). Thus, a string read using `scanf` will never contain white space. This makes `scanf` unsuitable for reading in several words of text into a string.
- The `gets` function (also found in `stdio.h`), works in much the same way as `scanf` with two notable exceptions:
 1. `gets` does not skip leading white space before starting to read the string.
 2. `gets` reads until it finds a new-line character. `gets` discards the new-line character and replaces it with the null character.



Alternatives To `scanf` And `printf` For Strings

CAUTION

As `scanf` and `gets` read characters into an array, they have no way of determining when the array is full. As a result, they may attempt to store characters past the end of the array, causing unpredictable behavior.

`scanf` can be made safer by using the conversion specifier `%ns`, where `n` is an integer indicating the maximum number of characters to be stored, with subsequent characters simply being ignored.

Unfortunately, `gets` is inherently unsafe in this regard, so it is the responsibility of the programmer to ensure that the number of characters read by `gets` is always 1 less than the size of the array being used.



Alternatives To `scanf` And `printf` For Strings

- The `printf` function writes characters from the string one by one until it encounters a null character. (In most C environments, if the null character is missing, `printf` will continue printing characters past the end of the string until it eventually finds a null character, i.e., a byte containing all zeros, somewhere in the memory.)
- Like a number, a string can be printed within a field. The conversion specifier `%ms` will print a string right-justified in a field width of size `m`. (A string with more than `m` characters will be printed in full and not truncated.)
- Using the precision specifier, `%m.ps`, will cause the first `p` characters of the string to be printed in a field width of size `m`.
- The example program on the next page illustrates some of these features.



```
1 //Strings in C - Part 2 - Options for scanf and printf with strings
2 //March 18, 2009      Written by: Mark Llewellyn
3
4 #include <stdio.h>
5 #define MAX_LENGTH 10
6
7 int main()
8 {
9     char aString[MAX_LENGTH]; //a string
10    char bString[MAX_LENGTH]; //another string
11
12    printf("Enter a string:\n");
13    scanf("%9s", aString);
14    printf("\nThe string you entered was: %s\n", aString);
15    printf("\nThe first 4 characters of the string you entered are: %.4s\n", aString);
16
17    printf("\n\n");
18    printf("Enter another string:\n");
19    scanf("%9s", bString);
20    printf("\nThe first 9 characters of the string you entered are: %.9s\n", bString);
21
22    printf("\n\n");
23    system("PAUSE");
24    return 0;
25 } //end main function
26
```

Using the `%ns` conversion specifier to limit number characters read into the array by `scanf`

Using the `%.ps` conversion specifier to limit number characters printed from the string `printf`



The %ns conversion specifier limited the string to 9 characters from the input

The %.ps conversion specifier limited the printing of the string to the first 4 characters.

```
Enter a string:  
thiswordistoolong
```

```
The string you entered was: thiswordi
```

```
The first 4 characters of the string you entered are: this
```

```
Enter another string:
```

```
The first 9 characters of the string you entered are: stoolong
```

Since the input string was longer than the input read (9 characters), and no white space has yet been encountered, the characters already in the input buffer are used to fill the second string!.

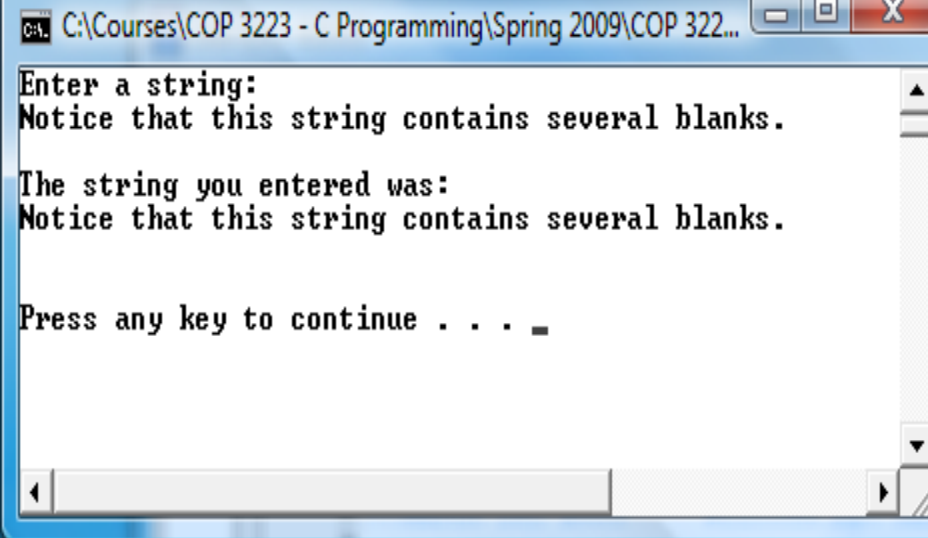


Alternatives To `scanf` And `printf` For Strings

- The `printf` function is not the only function that can write strings.
- The `stdio.h` library also includes a `puts` function, which takes as a single argument, the string to be printed.
- The `puts` function works in much the same way as `printf` with one notable exception:
 1. `puts` always writes a new-line character and the end of the string, thus advancing to the beginning of the next output line.
- The program on the following page illustrates using `gets` and `puts` to read and write strings.
- Some of the functions in `<stdio.h>` are shown in the table on page 9. To see all the functions in `<stdio.h>` you can go to: <http://en.wikipedia.org/wiki/Stdio.h>



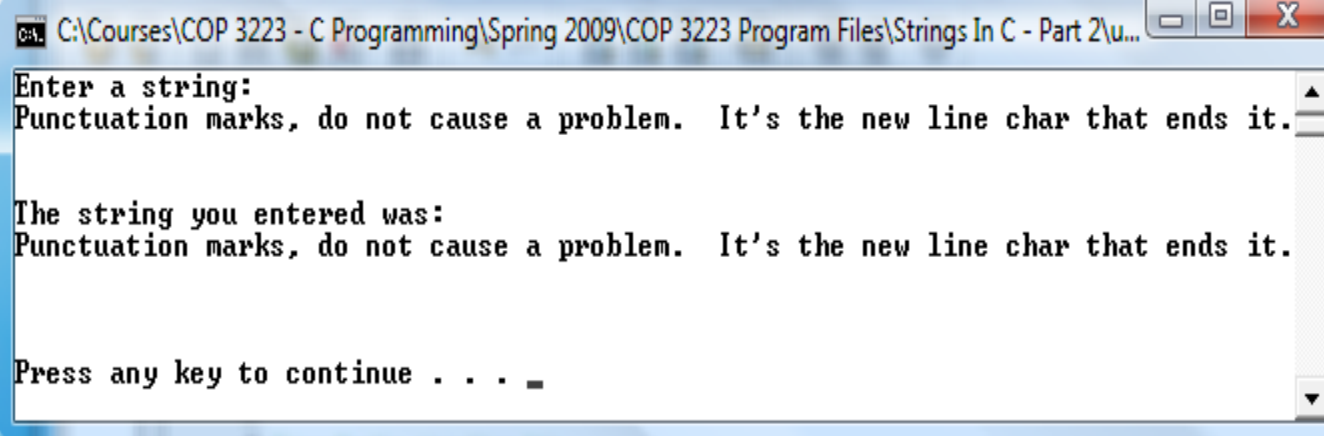
```
1 //Strings in C - Part 2 - Using gets and puts for reading and writing strings
2 //March 18, 2009    Written by: Mark Llewellyn
3
4 #include <stdio.h>
5 #define MAX_LENGTH 81
6
7 int main()
8 {
9     char aString[MAX_LENGTH]; //a string
10
11     printf("Enter a string:\n");
12     gets(aString);
13     printf("\nThe string you entered was:\n");
14     puts(aString);
15
16     printf("\n\n");
17     system("PAUSE");
18     return 0;
19 } //end main function
20
```



```
C:\Courses\COP 3223 - C Programming\Spring 2009\COP 322...
Enter a string:
Notice that this string contains several blanks.

The string you entered was:
Notice that this string contains several blanks.

Press any key to continue . . . _
```



```
C:\Courses\COP 3223 - C Programming\Spring 2009\COP 3223 Program Files\Strings In C - Part 2\u...
Enter a string:
Punctuation marks, do not cause a problem. It's the new line char that ends it.

The string you entered was:
Punctuation marks, do not cause a problem. It's the new line char that ends it.

Press any key to continue . . . _
```



Some Of The Functions In `<stdio.h>`

Function Prototype	Function Description
<code>int getchar (void);</code>	Inputs the next character from the standard input and returns it as an integer.
<code>char *gets (char *s);</code>	Inputs characters from the standard input into the array <code>s</code> until a newline or end-of-file character is encountered. A terminating null character is appended to the array. Returns the string inputted into <code>s</code> . An error will occur if <code>s</code> is not large enough to hold the string.
<code>int putchar (int c);</code>	Prints the character stored in <code>c</code> and returns it as an integer.
<code>int puts (const char *s);</code>	Prints the string <code>s</code> followed by a newline character. Returns a non-zero integer if successful, or EOF if an error occurs.
<code>int sprintf (char *s, const char *format, ...);</code>	Equivalent to <code>printf</code> , except the output is stored in the array <code>s</code> instead of printed on the screen. Returns the number of characters written into <code>s</code> , or EOF if an error occurs.
<code>int sscanf (char *s, const char *format, ...);</code>	Equivalent to <code>scanf</code> , except the input is read from the array <code>s</code> rather than from the keyboard. Returns the number of items successfully read by the function, or EOF if an error occurs.



Reading Strings Character By Character

- As we've seen in the previous few examples, due to the manner in which `scanf` and `gets` read characters into strings, they are often not the ideal way to enter string data into an application program.
- Quite often a programmer will simply write their own function for reading strings using a character by character approach. This gives the programmer a greater degree of flexibility and control than using the standard input functions.
- However, things aren't quite as simple as they may seem, because now you must consider several issues when designing your own string input functions.



Reading Strings Character By Character

- Among the issues to consider are:
- What character causes the function to stop reading characters from the input? Will it be the new-line character, any white space character, or some special character?
- Is the character that triggered the end of the string stored in the string or discarded?
- Should the function skip any leading white space characters or include them in the string?
- What should the function do if the input string is longer than the array in which the string is to be stored? Should it discard the remaining characters, or leave them to be read by the next operation?



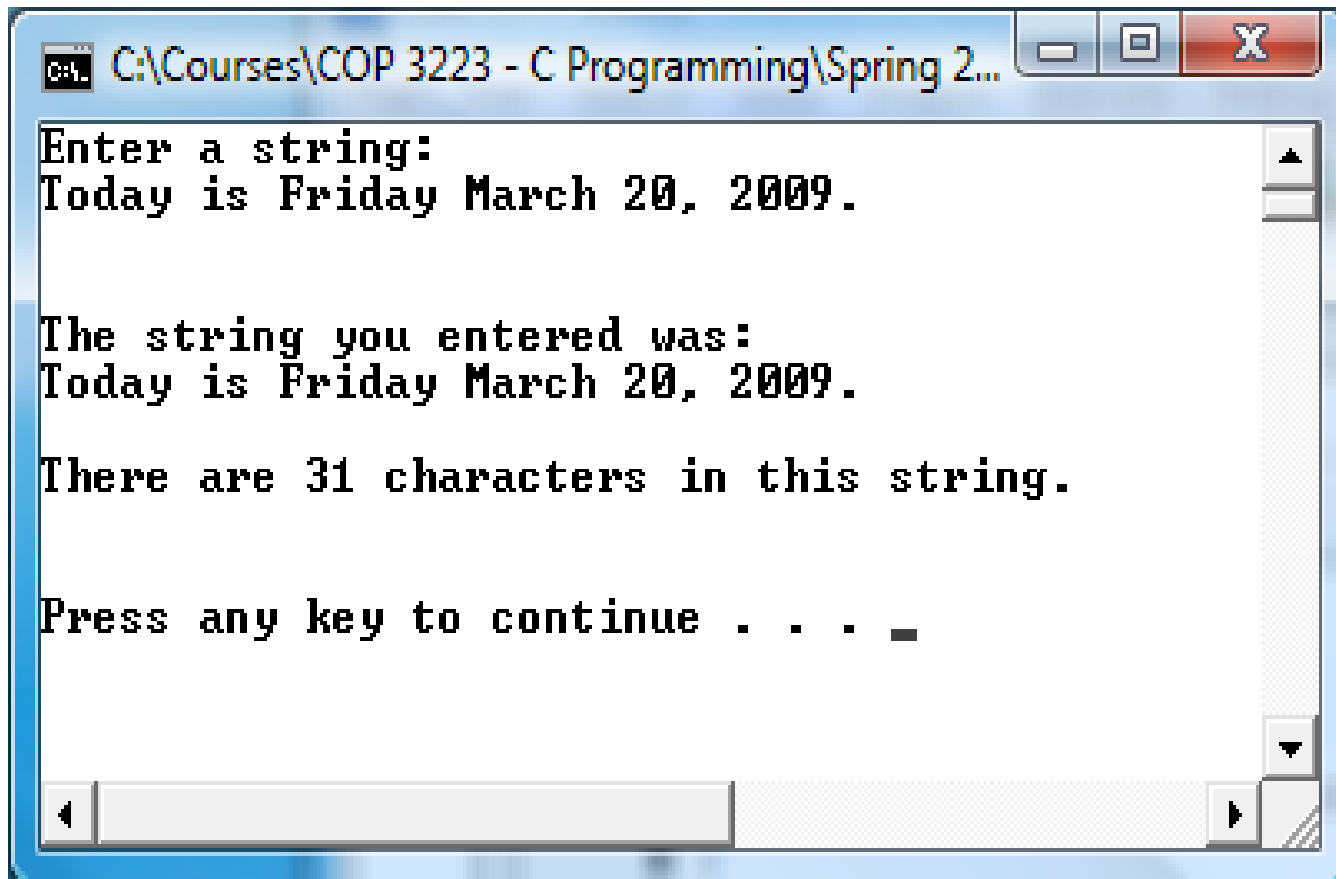
Reading Strings Character By Character

- When writing a character by character string input function all of these questions need to be answered by the programmer.
- Suppose we wanted to write such a function with the following conditions:
 - The function should stop reading characters at the first new-line character it encounters.
 - The new-line character is not to be stored in the string.
 - Leading white space characters are not ignored.
 - If the input string is too long for the array, the extra characters are discarded.
- The function on the following page satisfies these criteria.



```
4 #include <stdio.h>
5 #define MAX_LENGTH 81
6
7 int readString(char aString[], int maxNumChar)
8 {
9     int ch; //a character from the input
10    int i = 0; //loop control
11
12    while ((ch = getchar()) != '\n') {
13        if (i < maxNumChar) { //still inside valid string length
14            aString[i++] = ch; //assign array position the input character
15        } //end if stmt
16    } //end while stmt
17    aString[i] = '\0'; //put null character on end of string
18    return i; //return actual number of characters in the string.
19 } //end readString function
20
21 int main()
22 {
23     char sentence[MAX_LENGTH]; //a character string
24     int numberOfCharacters; //actual length of the string
25
26     printf("Enter a string:\n");
27     numberOfCharacters = readString(sentence, MAX_LENGTH);
28     printf("\n\n");
29     printf("The string you entered was:\n");
30     puts(sentence);
31     printf("\nThere are %d characters in this string.\n", numberOfCharacters);
32 }
```





A screenshot of a Windows command prompt window. The title bar shows the path "C:\Courses\COP 3223 - C Programming\Spring 2...". The window contains the following text:

```
Enter a string:  
Today is Friday March 20, 2009.  
  
The string you entered was:  
Today is Friday March 20, 2009.  
  
There are 31 characters in this string.  
  
Press any key to continue . . . -
```



Access To The Characters In A String

- Since strings are stored as arrays in C, as we've already seen in the first section of the notes on strings, you can use the array subscripting notation to access the individual characters in a string. (See page 7 of String In C – Part 1).
- So, just as we can read in strings character by character, we can also move through the string character by character. A simple loop can accomplish this task for us.
- The program on the next page uses a function named `countBlanks` simply returns the number of blank characters in an input string passed to it as a parameter.



```
21 int countBlanks(const char aString[])
22 {
23     int counter = 0; //the count of blank characters in the string
24     int i; //loop control
25
26     for ( i = 0; aString[i] != '\0'; ++i) {
27         if (aString[i] == ' ') { //found a blank character
28             counter++; //increment blank counter
29         } //end if stmt
30     } //end for stmt
31     return counter; //return number of blank characters in the string.
32 } //end countBlanks function
33
34 int main()
35 {
36     char sentence[MAX_LENGTH]; //a character string
37     int howManyBlanks; //number of blank characters in the string
38     int numberOfCharacters; //actual length of the string
39
40     printf("Enter a string:\n");
41     numberOfCharacters = readString(sentence, MAX_LENGTH);
42     printf("\n\n");
43     printf("The string you entered was:\n");
44     puts(sentence);
45     howManyBlanks = countBlanks(sentence);
46     printf("\nThere are %d blank characters in this string.\n", howManyBlanks);
47
48     printf("\n\n");
```




```
C:\Courses\COP 3223 - C Programming\Spring 2009\...
Enter a string:
Today is Friday March 20, 2009.

The string you entered was:
Today is Friday March 20, 2009.

There are 5 blank characters in this string.

Press any key to continue . . . _
```

```
C:\Courses\COP 3223 - C Programming\Spring 2009\COP 3223 Program Files\Strings In ...
Enter a string:
  There are three blanks before the word there in this sentence.

The string you entered was:
  There are three blanks before the word there in this sentence.

There are 13 blank characters in this string.

Press any key to continue . . .
```



Using The String Handling Library In C `<string.h>`

- The string handling library `<string.h>` includes many useful functions for manipulating strings.
- Functions included in `<string.h>` handle copying strings, concatenating strings, comparing strings, searching strings for characters and substrings, tokenizing strings (separating strings into logical pieces), and determining the length of strings.
- The next page list some of the more commonly used functions in `<string.h>`, but for a complete listing you can go to: <http://en.wikipedia.org/wiki/Memcpy#Functions>
- A series of small programs begins on page 20 that illustrate some of these more commonly used functions from `<string.h>`.



Some Of The Functions In <string.h>

Function Prototype	Function Description
<code>char *strcpy (char *s1, const char *s2);</code>	Copies string s2 into array s1. The value of s1 is returned (i.e., a pointer to s1 is returned). s2 is not modified.
<code>char *strcat (char *s1, const char *s2);</code>	Appends string s2 to array s1. The first character of s2 overwrites the terminating null character in s1. The value of s1 is returned (i.e., a pointer to s1 is returned). s2 is not modified.
<code>int strcmp (const char *s1, const char *s2);</code>	Compares string s1 with s2. The function returns 0 if s1 is equal to s2; returns a negative value if s1 less than s2; returns a positive value if s1 is greater than s2. Neither s1 nor s2 is modified.
<code>char *strchr (const char *s, int c);</code>	Locates the first occurrence of character c in string s. If c is found a pointer to c in s is returned. Otherwise, a NULL pointer is returned. The string s1 is not modified.
<code>char *strstr (const char *s1, const char *s2);</code>	Locates the first occurrence in string s1 of string s2. (That is to say that s2 is a substring of s1). If the string is found, a pointer to the string in s1 is returned. Otherwise, a NULL pointer is returned. Neither s1 nor s2 is modified.



```
1 //Strings In C - Part 2 - Copying strings using strcpy - version 1 with pointers
2 //March 20, 2009    Written by: Mark Llewellyn
3
4 #include <stdio.h>
5 #include <string.h>
6 #define MAX_LENGTH 81
7
8 int main()
9 {
10     char sentenceOne[MAX_LENGTH]; //a string
11     char sentenceTwo[MAX_LENGTH]; //another string
12     char *ptr1, *ptr2; //two pointers to characters
13
14     printf("Enter a string of 80 characters or less . . . \n");
15     gets(sentenceOne);
16     printf("\n\nYou entered the string:\n");
17     puts(sentenceOne);
18     ptr1 = sentenceOne;
19     ptr2 = sentenceTwo;
20     ptr2 = strcpy(ptr2, ptr1);
21     printf("\n\nAfter copying sentence1 into sentence2, sentence2 contains:\n");
22     puts(sentenceTwo);
23
24     printf("\n\n");
25     system("PAUSE");
26     return 0;
27 }//end main function
28 |
29
```

This version of the string copy example uses explicit pointers. A pointer is declared and assigned to each character array (string). When the parameter are passed to `strcpy`, pointer values are sent and returned.



```
C:\Courses\COP 3223 - C Programming\Spring 2009\COP 3223 Program Fi...
Enter a string of 80 characters or less . . .
Today is Friday March 20, 2009.

You entered the string:
Today is Friday March 20, 2009.

After copying sentence1 into sentence2, sentence2 contains:
Today is Friday March 20, 2009.

Press any key to continue . . .
```

```
C:\Courses\COP 3223 - C Programming\Spring 2009\COP 3223 Program ...
Enter a string of 80 characters or less . . .
Today is not Thursday March 19, 2009.

You entered the string:
Today is not Thursday March 19, 2009.

After copying sentence1 into sentence2, sentence2 contains:
Today is not Thursday March 19, 2009.

Press any key to continue . . .
```



```
1 //Strings In C - Part 2 - Copying strings - version 2 without explicit pointers
2 //March 20, 2009   Written by: Mark Llewellyn
3
4 #include <stdio.h>
5 #include <string.h>
6 #define MAX_LENGTH 81
7
8 int main()
9 {
10     char sentenceOne[MAX_LENGTH]; //a string
11     char sentenceTwo[MAX_LENGTH]; //another string
12     char *discard; //this pointer is returned by strcpy but we won't use it
13
14     printf("Enter a string of 80 characters or less . . . \n");
15     gets(sentenceOne);
16     printf("\n\nYou entered the string:\n");
17     puts(sentenceOne);
18     discard = strcpy(sentenceTwo, sentenceOne);
19
20     printf("\n\nAfter copying sentence1 into sentence2, sentence2 contains:\n");
21     puts(sentenceTwo);
22
23     printf("\n\n");
24     system("PAUSE");
25     return 0;
26 } //end main function
27
```

This version of the string copy example does not use explicit pointers. However, since the `strcpy` function returns a pointer, we must declare one for the return value, but we simply discard it (i.e., never use it).



```
C:\Courses\COP 3223 - C Programming\Spring 2009\COP 3223 Program Fil...  
Enter a string of 80 characters or less . . .  
I'm getting really tired of typing the same sentence!  
  
You entered the string:  
I'm getting really tired of typing the same sentence!  
  
After copying sentence1 into sentence2, sentence2 contains:  
I'm getting really tired of typing the same sentence!  
  
Press any key to continue . . .
```



```
1 //Strings In C - Part 2 - example string concatenation using strcat -
2 //March 19, 2009      Written by: Mark Llewellyn
3
4 #include <stdio.h>
5 #include <string.h>
6 #define MAX_LENGTH 81
7
8 int main()
9 {
10     char sentence1[MAX_LENGTH]; //a string
11     char sentence2[MAX_LENGTH]; //another string
12     char *discard; //this pointer will be ignored
13
14     printf("Enter a string of no more than 40 characters:\n");
15     gets(sentence1);
16     printf("\nEnter another string of no more than 40 characters:\n");
17     gets(sentence2);
18     discard = strcat(sentence1, sentence2);
19     printf("\n\nAfter concatenation sentence1 contains:\n");
20     puts(sentence1);
21
22     printf("\n\n");
23     system("PAUSE");
24     return 0;
25 } //end main function
26 |
27
```



Since the second sentence did not start with a blank, the concatenated sentence does not have a space between the two sentences as does the second version.

```
C:\Courses\COP 3223 - C Programming\Spring 2009\COP 3223 Program ...
Enter a string of no more than 40 characters:
This is the first sentence.

Enter another string of no more than 40 characters:
This is the second sentence.

After concatenation sentence1 contains:
This is the first sentence.This is the second sentence.

Press any key to continue . . .
```

```
C:\Courses\COP 3223 - C Programming\Spring 2009\COP 3223 Program Fil...
Enter a string of no more than 40 characters:
This is the first sentence.

Enter another string of no more than 40 characters:
This is the second sentence.

After concatenation sentence1 contains:
This is the first sentence. This is the second sentence.

Press any key to continue . . .
```



Practice Problems

1. Redo Practice Problem #1 from Strings In C – Part 1 where you wrote a program that reads in two strings and then determines if the strings are the same or not. This time, use the `strcmp` from `<string.h>` rather than your own function.
2. Write a program that will have the user enter two strings and then determine if the second string is contained in the first string.

