CSE 1320

Week of 02/18/2019

Instructor : Donna French

Strings

A string is a sequence of characters from the underlying character set.

A string in C is terminated by a null character, `'\0'`

A string is accessed via a pointer to its first character.

A string is like an array of characters – both are stored in contiguous memory.

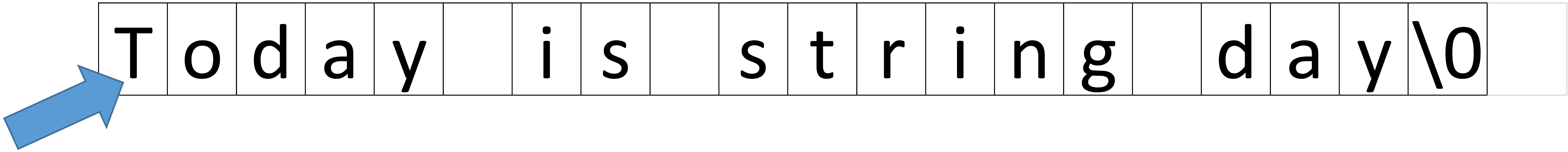
Arrays do not require a null character.

strings must have a null character at the end

Strings

When the compiler sees a sequence of characters enclosed in double quotes, it stores the sequence and appends a terminating `'\0'` to the end of the character sequence.

```
"Today is string day"
```



T	o	d	a	y		i	s		s	t	r	i	n	g		d	a	y	\0	

The compiler then associates the string constant with the address of the memory location of the first character in the string.

Strings

The first parameter to `printf()` is a string constant.

This memory location/string constant stored by the compiler is the parameter used by `printf()` to output the string. It stops outputting characters when it finds the null character (`'\0'`).

```
printf("Today is string day");
```

```
Today is string day
```

Strings

What happens when you put a null character in the middle of a string?

```
printf("Please don't interrupt\0 me while I am printing");
```

```
Please don't interrupt me while I am printing
```
✗

```
Please don't interrupt
```
✓

Strings

The compiler associates the string constant with the address of the memory location of the first character in the string.

Which means we can store that address in a pointer.

```
char *StringPtr = "Today is string day";
printf(StringPtr);
```

```
Breakpoint 1, main () at string1Demo.c:8
8               char *StringPtr = "Today is string day";
(gdb) step
11              printf(StringPtr);
(gdb) p StringPtr
$1 = 0x4008b8 "Today is string day"
(gdb) p *StringPtr
$2 = 84 'T'
```

string1Demo.c

string1Demo.c

Strings

```
char *StringPtr = "Today is string day";
```

C allows the use of the array indexing syntax to access the individual elements of a string.

```
for (i = 0; i < 20; i++)
    printf("%c", StringPtr[i]);

for (i = 0; StringPtr[i] != '\0'; i++)
    printf("%c", StringPtr[i]);
```

T	o	d	a	y		i	s		s	t	r	i	n	g		d	a	y	\0

Strings

C allows the use of pointer arithmetic and dereferencing to access the individuals characters in a string.

```
char *StringPtr = "Today is string day";


printf("The first  character of the string is %c\n", *StringPtr);
printf("The second character of the string is %c\n", *(StringPtr+1));
printf("The third  character of the string is %c\n", *(StringPtr+2));
```

T	o	d	a	y		i	s		s	t	r	i	n	g		d	a	y	\0

string1Demo.c

Strings

%s

%s signals the output of a string to **printf()** which then expects the corresponding parameter to be the address of the first character in a string. It starts outputting the character found at that address and outputs subsequent characters until it finds a null character.

```
char *StringPtr = "Today is string day";
printf("%s", StringPtr);
```

```
Today is string day
```

string1Demo.c

Strings

%s

`%s` signals the input of a string to `scanf()` which then expects the parameter to be the address of an array with enough space to handle the input. `scanf()` will put the first sequence of characters that does not contain a whitespace character into the given variable.

```
Enter a string: three two one
Printing the string with %s - three
```

```
char Array[50];
printf("Enter a string: ");
scanf("%s", Array);
printf("Printing the string %%s - %s");
```

Why isn't this using `&Array`?

string1Demo.c

Strings

`scanf()` adds a null terminator to the array

```
Breakpoint 2, main () at string1Demo.c:45
45                scanf("%s", Array);
(gdb) step
Enter a string three two one
47                printf("Printing the string with %%s - %s", Array);
(gdb) p Array
$1 = "three\000\303\000\027\b@", '\000' <repeats 13 times>"\300,
\313!\311>\000\000\000\340\a@", '\000' <repeats 13 times>"\220, ",
<incomplete sequence \350>
```

string1Demo.c

Strings

```
Breakpoint 2, main () at string1Demo.c:44
44                printf("Enter a string ");
(gdb) ptype Array
type = char [5]
(gdb) step
45                scanf("%s", Array);
(gdb)
Enter a string encyclopedia
47                printf("Printing the string with %%s - %s", Array);
(gdb) p Array
$3 = "encyc"
(gdb) p *Array@20
$4=   "encyclopedia\000\000\000\000\220\350\377\377"
(gdb) step
Printing the string with %s - encyclopedia
```

What happens if the array is not large enough to hold the input?

string1Demo.c

Arrays and String Manipulation

A string can be stored in an array at the time the array is declared.

```
char StringArray[80] = "This is my string in my StringArray\n";
```

```
printf(StringArray);
This is my string in my StringArray

printf("%s", StringArray);
This is my string in my StringArray

printf("%p", StringArray);
0x7fffffffe750
```

```
(gdb) p StringArray
$1 = "This is my string in my
StringArray\n", '\000' <repeats
43 times>

(gdb) p *StringArray
$2 = 84 'T'
(gdb) p &StringArray
$3 = (char (*)[80])
0x7fffffffe750
```

string2Demo.c

Arrays and String Manipulation

A string can be stored in an array at the time the array is declared.

```
char StringArray[80] = “This is my string in my StringArray\n”;

*(StringArray + 8) = *(StringArray + 8) ^ 32;
*(StringArray + 11) ^= 32;
*(StringArray + 18) = '\0';

printf(StringArray);
```

```
This is My String
```

T	h	i	s		i	s		M	y
	S	t	r	i	n	g		\0	n
	m	y		S	t	r	i	n	g
A	r	r	a	y	\n	\0			

Variable Strings

To input a variable length string

- create an array large enough to hold the max possible length
- store user input in array one character at a time
- when newline is entered, replace it with null character to terminate the string
- `%s` can then be used with `printf()` to print the string

Variable Strings

```
char String[80];

int i = 0, StringLength = 80;

while (i < StringLength)
{
    String[i] = getchar();

    if (String[i] == '\n')
    {
      String[i] = '\0';
      break;
    }

    i++;
}
```

```
/* user typed in more than 80 characters */
if (i == StringLength)
{
   printf("Truncating input string\n");
   String[i] = '\0';
}
```

Using `\n` as a signal to while loop for when to quit reading from `stdin`

string3Demo.c

Input and Output of Strings

`fgets(inbuff, n, fp)`

accepts input of a string of maximum length **`n-1`** from one line of the file `fp`

Why n-1?

Parameters

- `inbuff` the address of the buffer that will hold the string
- `n` an `int` representing the maximum length of the buffer
- `fp` a `FILE *` representing the open file from which input is to come

Return value

- a `char*` value (the address of `inbuff`) or `NULL` in case of error or end-of-file

```c
#include <stdio.h>

#define MAX_INPUT 40

int main(void)
{
  char MyString[MAX_INPUT];
  char *MyStringPtr;

  printf("\nEnter a line of text (max of %d)\n\n", MAX_INPUT-1);

  /* fgets() will terminate the string with \0 after the newline */
  MyStringPtr = fgets(MyString, MAX_INPUT, stdin);

  printf("\nThe string you entered is\n\n\"%s\"\n", MyString);

  return 0;
}
```

fgets1Demo.c

Input and Output of Strings

`fgets()` **must be given an array to write into**

```
char MyString[MAX_INPUT];
char *MyStringPtr;
MyStringPtr = fgets(MyString, MAX_INPUT, stdin);
```

vs

```
char *MyStringPtr;
char *MyOtherStringPtr;
MyStringPtr = fgets(MyOtherStringPtr, MAX_INPUT, stdin);
```

```
#include <stdio.h>

#define MAX_INPUT 40

int main(void)
{
  char *MyStringPtr;
  char *MyOtherStringPtr;

  printf("\nEnter a line of text (max of %d)\n\n", MAX_INPUT-1);

  MyStringPtr = fgets(MyOtherStringPtr, MAX_INPUT, stdin);

  return 0;
}
```

```
[frenchdm@omega ~]$ gcc fgets2Demo.c
[frenchdm@omega ~]$ a.out

Enter a line of text (max of 40)

The quick fox jumps over the brown dog
Segmentation fault
[frenchdm@omega ~]$
```

Why?

Because it wrote into memory that was not allocated to the process

fgets2Demo.c

Common Coding Mistake

```
char MyString[MAX_INPUT];
char *MyStringPtr;

printf("\nEnter a line of text (max of %d)\n\n", MAX_INPUT-1);

MyString = fgets(MyString, MAX_INPUT, stdin);
```

```
[frenchdm@omega ~]$ gcc fgets1Demo.c
fgets1Demo.c: In function ‘main’:
fgets1Demo.c:16: error: incompatible types in assignment
```

Short Version of `fgets()`

```
char MyString[MAX_INPUT];
char *MyStringPtr;

MyStringPtr = fgets(MyString, MAX_INPUT, stdin);
```

```
char MyString[MAX_INPUT];

fgets(MyString, MAX_INPUT, stdin);
```

Input and Output of Strings

`fgets()` vs `gets()`

```
char InputArray[MAX_INPUT];

fgets(InputArray, MAX_INPUT, stdin);

gets(InputArray);
```

```
[frenchdm@omega ~]$ gcc getsDemo.c
/tmp/ccGWXitm.o: In function `main':
getsDemo.c:(.text+0x30): warning: the `gets' function is dangerous
and should not be used.
```

Conclusion – `gets()` cannot be safely used; therefore, do not use it.

Using `gets()` in a Coding Assignment in this class will result in an automatic 0

The Common String Library Functions

`strlen()` - calculates the length of a string

`strcpy()` - makes a copy of a string

`strcat()` - concatenates two strings

`strncat()` - concatenates two strings for a specified number of characters

`strcmp()` - compares two strings

`strncmp()` - compare two strings for a specified number of characters

`strchr()` - searches for a character in a string

`strstr()` - searches for a string in a string

`strpbrk()` - finds the first occurrence of any of a set of characters in a given string

`strtok()` - divides a string into tokens

The Common String Library Functions

`strlen(string)`

calculates the length of `string`

Parameters `string` a null-terminated string

Return value the length of `string` not including the terminating `‘\0’`, of type `size_t`

`size_t` is an unsigned integer type

stringtoolsDemo.c

strlen()

Why MAX_INPUT – 1?

```
char Buffer[MAX_INPUT];
char UserString[MAX_INPUT];

printf("\nEnter a line of text (max of %d)\n\n", MAX_INPUT-1);
fgets(UserString, MAX_INPUT, stdin);

printf("\nYou entered %s", UserString);
printf("\nThe length of your string is %d\n", strlen(UserString));
```

stringtoolsDemo.c

Removing the \n from an input string

Two methods

```
printf("\nEnter a line of text (max of %d)\n\n", MAX_INPUT-1);
fgets(UserString, MAX_INPUT, stdin);
```

Replace \n with a blank

```
    UserString[strlen(UserString)-1] = ' ';
```

Replace \n with \0

```
    UserString[strlen(UserString)-1] = '\0';
```

The Common String Library Functions

`strcpy(buffer, string)`

copies `string` into `buffer`

Parameters	`buffer`	is the address of a memory buffer in the program
	`string`	a null-terminated string
Return value	the address of `buffer`, a `char *`	

stringtoolsDemo.c

strcpy()

```
char Buffer[MAX_INPUT];
char UserString[MAX_INPUT];

printf("\nEnter a line of text (max of %d)\n\n", MAX_INPUT-1);
fgets(UserString, MAX_INPUT, stdin);

strcpy(Buffer, UserString);

printf("Buffer is %s", Buffer);
```

The Common String Library Functions

```
strcat(buffer, string)
```

concatenates `string` onto the end of the current string in `buffer`

Parameters	`buffer`	the address of a memory buffer in the program that contains a null-terminated string
	`string`	a null-terminated string
Return value	the address of `buffer`, a `char *`	

stringtoolsDemo.c

strcat()

```
printf("\nEnter line1 of text (max of %d)\n\n", MAX_INPUT-1);
fgets(UserString1, MAX_INPUT, stdin);
printf("\nEnter line2 of text (max of %d)\n\n", MAX_INPUT-1);
fgets(UserString2, MAX_INPUT, stdin);

printf("\nString1 = %s",   UserString1);
printf("\nString2 = %s\n", UserString2);

UserString1[strlen(UserString1)-1] = '\0';
UserString2[strlen(UserString2)-1] = '\0';

strcat(UserString1, UserString2);

printf("String1 = %s\n", UserString1);
printf("String2 = %s\n", UserString2);
```

```
Enter line1 of text (max of 99)

Hello there.

Enter line2 of text (max of 99)

How are you?

String1 = Hello there.

String2 = How are you?

String1 = Hello there.How are you?
String2 = How are you?
```

The Common String Library Functions

`strncat(buffer, string)`

concatenates `string` onto the end of the current string in `buffer`

Parameters	`buffer`	the address of a memory buffer in the program that contains a null-terminated string
	`string`	a null-terminated string
	`n`	an `int` indicating the number of characters to concatenate

Return value the address of `buffer`, a `char *`

The Common String Library Functions

`strcmp(string1, string2)`

compares the contents of `string1` with that of `string2`

Parameters	`string1`	a null-terminated string
	`string2`	a null-terminated string

Return value a value of type `int`

`0`	`string1` and `string2` are identical
positive	`string1` would occur[*] after `string2`
negative	`string1` would occur[*] before `string2`

[*] in the ordering given by the ASCII character set

Question

What are the values returned by `strcmp()` – other than just positive or negative?

Return value	a value of type `int`
`0`	`string1` and `string2` are identical
positive	`string1` would occur after `string2`
negative	`string1` would occur before `string2`

Answer

Never rely on the exact return value of `strcmp()` (other than `0`, of course). The only guarantee is that the return value will be negative if the first string is "smaller", positive if the first string is "bigger" or `0` if they are equal. The same inputs may generate different results on different platforms with different implementations of `strcmp()`.

Bottom line – do not try to use the return value of `strcmp()` as anything other than a test for `> 0`, `< 0` or `0`.

stringtoolsDemo.c

strcmp()

```
printf("\nEnter a line of text (max of %d)\n\n", MAX_INPUT-1);
fgets(UserString1, MAX_INPUT, stdin);
printf("\nEnter a line of text (max of %d)\n\n", MAX_INPUT-1);
fgets(UserString2, MAX_INPUT, stdin);
printf("\n\n");

UserString1[strlen(UserString1)-1] = '\0';
UserString2[strlen(UserString2)-1] = '\0';

if (strcmp(UserString1, UserString2) == 0)
   printf("Strings are identical\n");
else if (strcmp(UserString1, UserString2) > 0)
   printf("%s\n>\n%s\n", UserString1, UserString2);
else if (strcmp(UserString1, UserString2) < 0)
   printf("%s\n<\n%s\n", UserString1, UserString2);
else
   printf("strcmp failed\n");
```

The Common String Library Functions

`strncmp(string1, string2, n)`

compares the first `n` characters of `string1` to the first `n` characters of `string2`

Parameters	`string1`	a null-terminated string
	`string2`	a null-terminated string
	`n`	an `int` indicating the number of characters to compare

Return value a value of type `int`

`0`	strings are identical
positive	`string2` would occur before `string1` in the ordering given by the ASCII character set
negative	`string1` would occur before `string2`

strncmp()

stringtoolsDemo.c

```
printf("\nEnter a line of text (max of %d)\n\n", MAX_INPUT-1);
fgets(UserString1, MAX_INPUT, stdin);
printf("\nEnter a line of text (max of %d)\n\n", MAX_INPUT-1);
fgets(UserString2, MAX_INPUT, stdin);

UserString1[strlen(UserString1)-1] = '\0';
UserString2[strlen(UserString2)-1] = '\0';

printf("Enter how many letters to compare ");
scanf("%d", &n);

if (strncmp(UserString1, UserString2, n) == 0)
   printf("Strings are identical for the first %d characters\n", n);
else if (strncmp(UserString1, UserString2, n) > 0)
   printf("%s\n>\n%s\n", UserString1, UserString2);
else if (strncmp(UserString1, UserString2, n) < 0)
   printf("%s\n<\n%s\n", UserString1, UserString2);
else
   printf("strncmp failed\n");
```

The Common String Library Functions

`strchr(string, ch)`

looks for the first occurrence of `ch` in `string`

Parameters	`string`	a null-terminated string
	`ch`	a character

Return value — a `char *` pointer to the first occurrence of `ch` in `string` or `NULL` if `ch` does not appear in string.

stringtoolsDemo.c

strchr()

```
char *FirstOccur;

printf("\nEnter a line of text (max of %d)\n\n", MAX_INPUT-1);
fgets(UserString, MAX_INPUT, stdin);
UserString[strlen(UserString)-1] = '\0';

printf("\nEnter a character\n\n");
scanf(" %c", &Ch);

FirstOccur = strchr(UserString, Ch);

while (FirstOccur != NULL)
{
   *FirstOccur = '-';
   FirstOccur = strchr(UserString, Ch);
}

printf("New version of String is\n\n%s", UserString);
```

The Common String Library Functions

`strstr(string1, string2)`

find the first occurrence of `string2` as a substring of `string1`

Parameters	`string1`	a null-terminated string
	`string2`	a null-terminated string
Return value	a `char *` pointer to the first occurrence of `string2` in `string1` or `NULL` if `string2` does not appear in `string1`.	

stringtoolsDemo.c

strstr()

```
printf("\nEnter a line of text (max of %d)\n\n", MAX_INPUT-1);
fgets(UserString1, MAX_INPUT, stdin);
printf("\nEnter world to search for (max of %d)\n\n", MAX_INPUT-1);
fgets(UserString2, MAX_INPUT, stdin);
UserString1[strlen(UserString1)-1] = '\0';
UserString2[strlen(UserString2)-1] = '\0';


if (strstr(UserString1, UserString2) != NULL)
   printf("%s\ncontains\n%s\n\n", UserString1, UserString2);
else
   printf("%s\ndoes not contain\n%s\n\n", UserString1, UserString2);
```

The Common String Library Functions

`strpbrk(string, char_set)`

find the first occurrence of any of a set of characters in `string`

Parameters	`string`	a null-terminated string
	`char_set`	a set of characters

Return value a `char *` pointer to the first occurrence of any character from `char_set` in `string` or `NULL` if no characters from `char_set` are found.

stringtoolsDemo.c

strpbrk()

```
printf("\nEnter a line of text (max of %d)\n\n", MAX_INPUT-1);
fgets(UserString, MAX_INPUT, stdin);
printf("\nEnter characters to replace with _\n\n");
fgets(Char_Set, MAX_INPUT, stdin);
Char_Set[strlen(Char_Set)-1] = '\0';

FirstOccur = strpbrk(UserString, Char_Set);

while (FirstOccur != NULL)
{
   *FirstOccur = '_';
   FirstOccur = strpbrk(UserString, Char_Set);
}

printf("Replacing all instances of\n\n\t%s\nwith _\n\n\n%s\n",
       Char_Set, UserString);
```

The Common String Library Functions

`strtok(buffer, delimiters)`

A “token” in `buffer` is defined to be a sequence of characters between any two occurrences of characters in delimiters. A call to `strtok()` places a null character at the end of the first “token” and returns the address of the first character of the “token”. Subsequent calls to `strtok()` with a NULL as the first parameter will find and isolate each “token” in `buffer`.

Parameters	`buffer`	a null-terminated string
	`delimiters`	a null-terminated string. The characters in the string mark the beginning and end of “tokens” in `buffer`.
Return value	The address of the next “token” in `buffer`	

stringtoolsDemo.c

strtok()

```
printf("\nEnter a line of text (max of %d) using Delimiters %s\n\n",
       MAX_INPUT-1, Delimiters);
fgets(Buffer, MAX_INPUT, stdin);
Buffer[strlen(Buffer) - 1] = '\0';

Token = strtok(Buffer, Delimiters);

while (Token != NULL)
{
   printf("Token = %s\n", Token);
   Token = strtok(NULL, Delimiters);
}
```

`strtok()`

How does `strtok()` work?

Does it use the `NULL` it inserts to track where to start?

If so, then why does it go into an infinite loop if you give it the string (now containing a `NULL`) instead of `NULL` inside the while loop?

strtok()

```
Token = strtok(UserString, Delimiters);
while (Token != NULL)
{
    printf("Token = %s\n", Token);
    Token = strtok(NULL, Delimiters);
}
```

`fgets` **has been called to get** `UserString` **and** `Delimiters`

```
(gdb) p UserString
$1 =
"hello,there,how,are,you\000\000\313!\311>\000\000\000\340\006@\000\000\000\0
00\000k\004@\000\000\000\000\000\001\000\000\000\000\000\303\000\027\a@",
'\000' <repeats 13 times>"\300, \313!\311>\000\000\000\340\006@", '\000'
<repeats 13 times>"\220, \350\377\377"
(gdb) p Delimiters
$2 = ",\000\000\377\001\000\000\000\340\366\252\252\252*\000\000`𢿥
\252*\000\000\020\350\377\377\377\177\000\000\000Ч\252*\000\000a\003@\000\000
\000\000\000v", '\000' <repeats 39 times>"\377,
\265\360\000\000\000\000\000\302\000\000"
```

strtok()

After this step is executed

```
Token = strtok(UserString, Delimiters);
```

We can look at `UserString`

```
(gdb) p UserString
$5 =
"hello\000there,how,are,you\000\000\313!\311>\000\000\0
00\340\006@\000\000\000\000\000k\004@\000\000\000\000\0
00\001\000\000\000\000\000\303\000\027\a@", '\000'
<repeats 13 times>"\300,
\313!\311>\000\000\000\340\006@", '\000' <repeats 13
times>"\220, \350\377\377"
```

strtok()

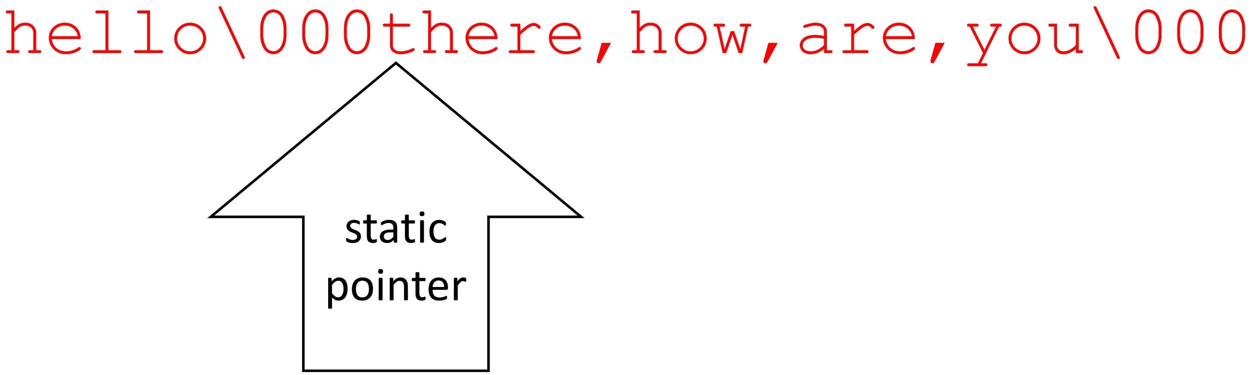
strtok() replaces the delimiter with a NULL so that when you use/print the pointer to start of that string, it only prints to the NULL and you only get "hello".

```
(gdb) p UserString
$5 =
"hello\000there,how,are,you\000\000\313!\311>\000\000\0
00\340\006@\000\000\000\000\000k\004@\000\000\000\000\0
00\001\000\000\000\000\000\303\000\027\a@", '\000'
<repeats 13 times>"\300,
\313!\311>\000\000\000\340\006@", '\000' <repeats 13
times>"\220, \350\377\377"
```

`strtok()`

`strtok()` does not use that `NULL` to start its search the next time it is called.

`strtok()` stores an internal static pointer that points to the first character in the string past where it overwrote the delimiter with `\0`.



`hello\000there,how,are,you\000`

When you pass it `NULL` on the second call in the `while` loop, you are signaling to it to use that static pointer as its starting point for looking for the next delimiter.

If you pass the string in the second call in the `while` loop, then you are signaling to `strtok()` that you are starting over with a new string and that it should not use the static pointer it had from the previous call.

#include <string.h>

```
[frenchdm@omega ~]$ gcc Code4_1000074079.c
Code4_1000074079.c: In function ‘main’:
Code4_1000074079.c:19: warning: incompatible implicit
declaration of built-in function ‘strcpy’
Code4_1000074079.c:23: warning: incompatible implicit
declaration of built-in function ‘strpbrk’
```

`string.h`

must be included in your program to use the C string library.

More Tools for Our Toolbox

`islower(ch)`	tests if `ch` is a lowercase alphabetic character
`isupper(ch)`	tests if `ch` is an uppercase alphabetic character
`isalpha(ch)`	tests if `ch` is an alphabetic character
`isalnum(ch)`	tests if `ch` is an alphanumeric character
`isdigit(ch)`	tests if `ch` is a decimal digit
`ispunct(ch)`	tests if `ch` is punctuation character
`isspace(ch)`	tests if `ch` is a whitespace character

```c
int main(void)
{
  char ch;

  printf("Enter a character ");
  scanf("%c", &ch);

  if (islower(ch)) printf("islower\n");
  if (isupper(ch)) printf("isupper\n");
  if (isalpha(ch)) printf("isalpha\n");
  if (isalnum(ch)) printf("isalnum\n");
  if (isdigit(ch)) printf("isdigit\n");
  if (ispunct(ch)) printf("ispunct\n");
  if (isspace(ch)) printf("isspace\n");

  return 0;
}
```

```
Enter a character a
islower
isalpha
isalnum

Enter a character A
isupper
isalpha
isalnum

Enter a character 1
isalnum
isdigit

Enter a character !
ispunct

Enter a character  
isspace
```

a space

ischDemo.c

More Tools for Our Toolbox

`tolower(ch)` returns the lowercase version of `ch`

`toupper(ch)` returns the lowercase version of `ch`

```
char ch;
char chUP;
char chLOW;

printf("Enter a character ");
scanf("%c", &ch);

chUP = toupper(ch);
printf("ch %c has been changed to %c\n", ch, chUP);

printf("ch %c has been changed to %c\n", ch, tolower(ch));
```

touplowDemo.c

```
Breakpoint 1, main () at touplowDemo.c:10
10              printf("Enter a character ");
(gdb) step
11              scanf("%c", &ch);
(gdb)
Enter a character a
13              chUP = toupper(ch);
(gdb) p ch
$4 = 97 'a'
(gdb) step
14              printf("ch %c has been changed to %c\n", ch, chUP);
(gdb) p ch
$5 = 97 'a'
(gdb) p chUP
$6 = 65 'A'
(gdb) step
ch a has been changed to A
16              printf("ch %c has been changed to %c\n", ch, tolower(ch));
(gdb) step
ch a has been changed to a
18              return 0;
```