CSE 1320

Week of 03/06/2023

Instructor : Donna French

Passing Parameters to Functions

In C

all parameters are passed by value

the ability to pass by reference does not exist

Pass by reference can be simulated

- pass the address of the variable
- address cannot be modified
- contents of address can be modified

```
int main(void)
{
  int MyMainNum = 0;

  printf("Before PassByValue call\tMyMainNum = %d\n", MyMainNum);
  PassByValue(MyMainNum);
  printf("After  PassByValue call\tMyMainNum = %d\n", MyMainNum);

  printf("Before PassByRef   call\tMyMainNum = %d\n", MyMainNum);
  PassByRef(&MyMainNum);
  printf("After  PassByRef   call\tMyMainNum = %d\n", MyMainNum);

  return 0;
}
```

simrefDemo.c

A copy is passed

```
int PassByValue(int MyNum)
{
      MyNum += 100;
      printf("Inside PassByValue\tMyNum     = %d\n", MyNum);
}
```

The address of the actual variable is passed

```
int PassByRef(int *MyNumPtr)
{
      *MyNumPtr += 100;
      printf("Inside PassByRef\tMyRefNum  = %d\n", *MyNumPtr);
}
```

simrefDemo.c

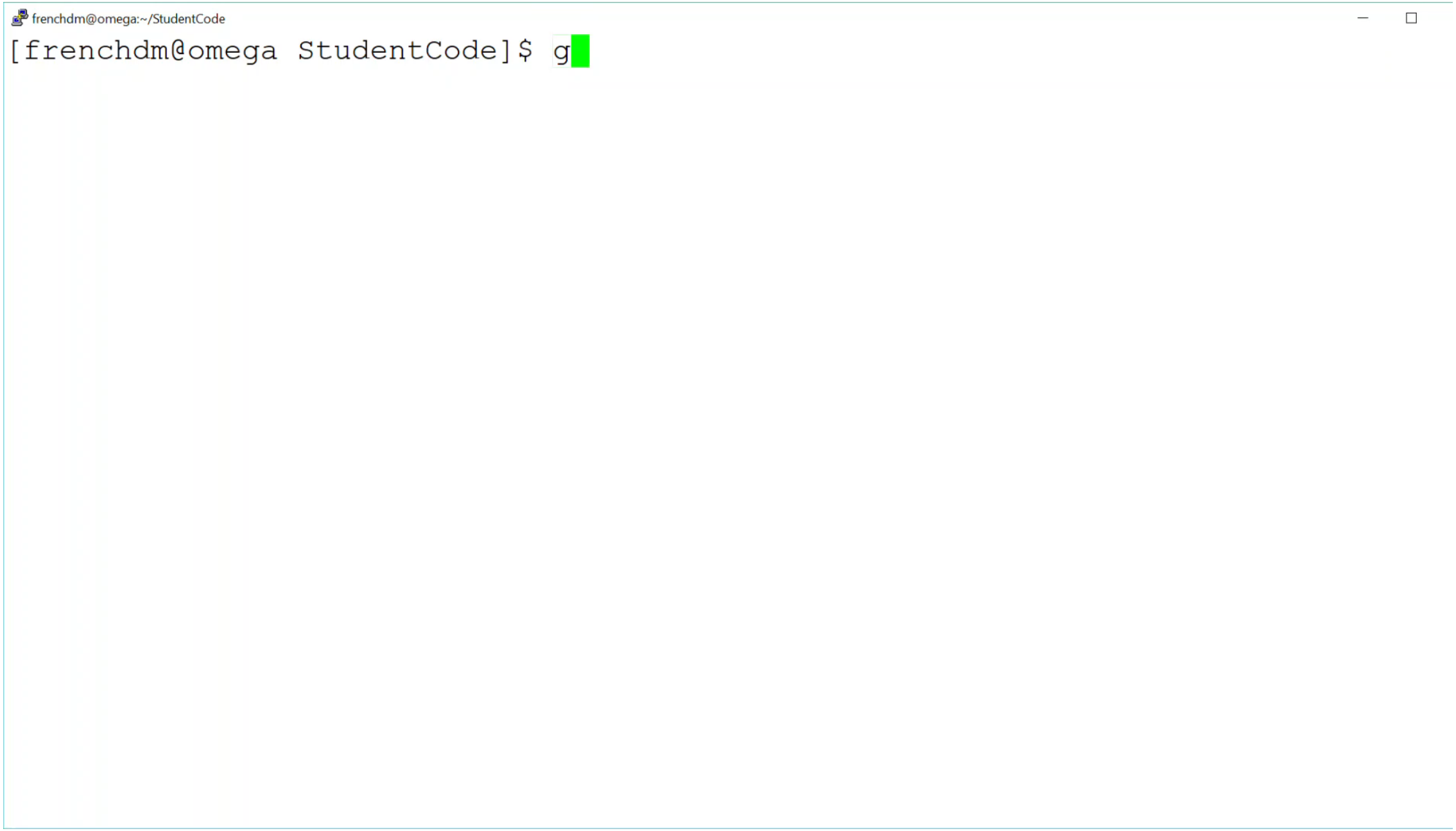
```
int MyMainNum = 0;

printf("Before PassByValue call\tMyMainNum = %d\n", MyMainNum);
PassByValue(MyMainNum);
printf("After  PassByValue call\tMyMainNum = %d\n", MyMainNum);

int PassByValue(int MyNum)
{
     MyNum += 100;
     printf("Inside PassByValue\tMyNum     = %d\n", MyNum);
}
```

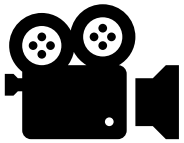
```
Before PassByValue call MyMainNum = 0
Inside PassByValue      MyNum     = 100
After  PassByValue call MyMainNum = 0
```

simrefDemo.c

```
int MyMainNum = 0;

printf("Before PassByRef   call\tMyMainNum = %d\n", MyMainNum);
PassByRef(&MyMainNum);
printf("After  PassByRef   call\tMyMainNum = %d\n", MyMainNum);

int PassByRef(int *MyNumPtr)
{
     *MyNumPtr += 100;
     printf("Inside PassByRef\tMyNumPtr  = %d\n", *MyNumPtr);
}

Before PassByRef   call MyMainNum = 0
Inside PassByRef        MyRefNum  = 100
After  PassByRef   call MyMainNum = 100
```

simrefDemo.c


frenchdm@omega:~/StudentCode
[frenchdm@omega StudentCode]$ g



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.

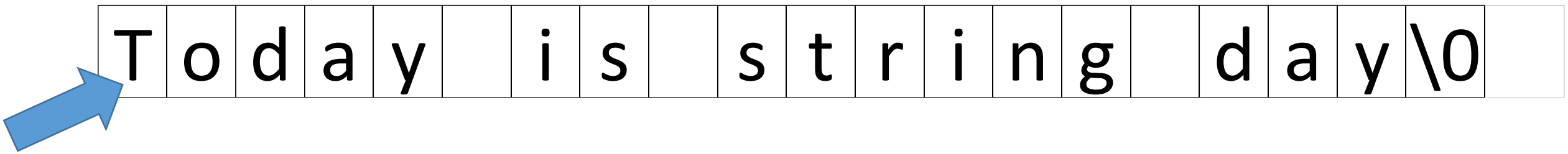
Arrays do not require a null character.

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"`



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'
```

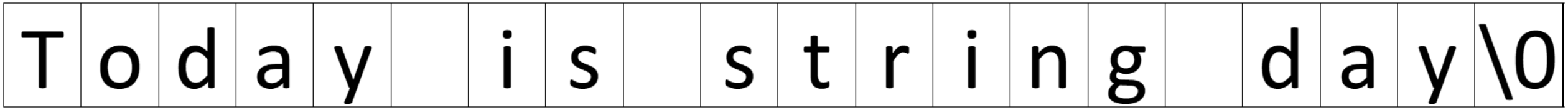
string1Demo.c

Strings

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

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

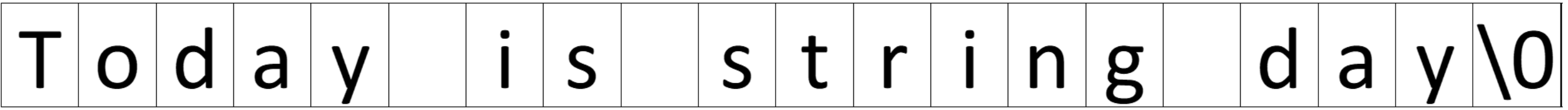
```
for (i = 0; i < 20; i++)
    printf("%c", StringPtr[i]);

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



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));
```



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", Array);
```

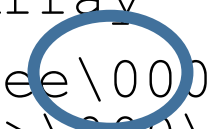
Why isn't this using &Array?

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
```

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);
```

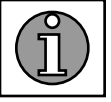
```
This is My String
```

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';
}
```

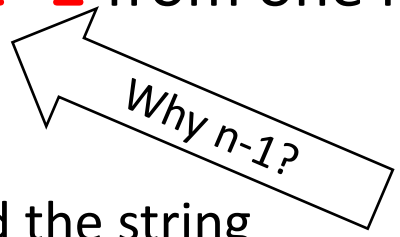
Using `\n` as a signal to while loop for when to quit reading from `stdin`

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`

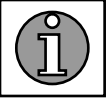


Parameters

- `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-1, stdin);

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

   return 0;
}
```

fgets1Demo.c

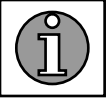
Input and Output of Strings

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

```
char MyString[MAX_INPUT];
char *MyStringPtr;
MyStringPtr = fgets(MyString, MAX_INPUT-1, stdin);
```

vs

```
char *MyStringPtr;
char *MyOtherStringPtr;
MyStringPtr = fgets(MyOtherStringPtr, MAX_INPUT-1, stdin);
```



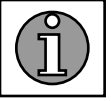
```
#include <stdio.h>

#define MAX_INPUT 41

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-1, 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?

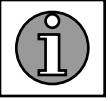
Because it wrote into memory that was not allocated to the process

fgets2Demo.c



Common Coding Mistake

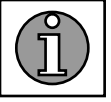
```
char MyString[MAX_INPUT];
char *MyStringPtr;

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

MyString = fgets(MyString, MAX_INPUT-1, 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-1, 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-1, stdin);

gets(InputArray);
```

```
student@maverick:/media/sf_VM/CSE1320$ gcc getsDemo.c
getsDemo.c: In function ‘main’:
getsDemo.c:8:2: warning: implicit declaration of function ‘gets’; did you
mean ‘fgets’? [-Wimplicit-function-declaration]
    8 |  gets(buffer);
      |  ^~~~
      |  fgets
/usr/bin/ld: /tmp/ccCWXoax.o: in function `main':
getsDemo.c:(.text+0x3f): 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

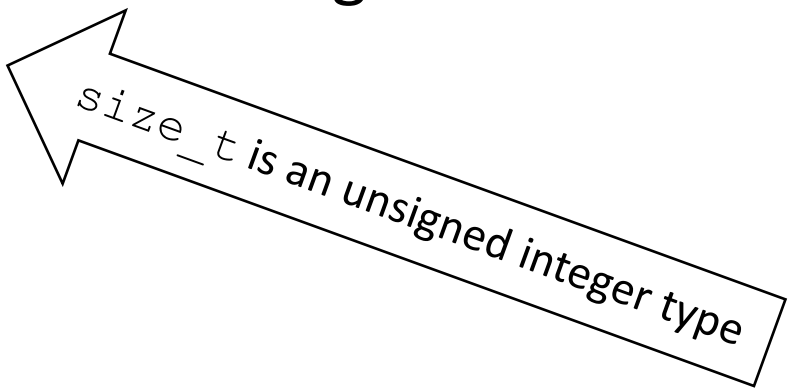
The Common String Library Functions

`strlen(string)`

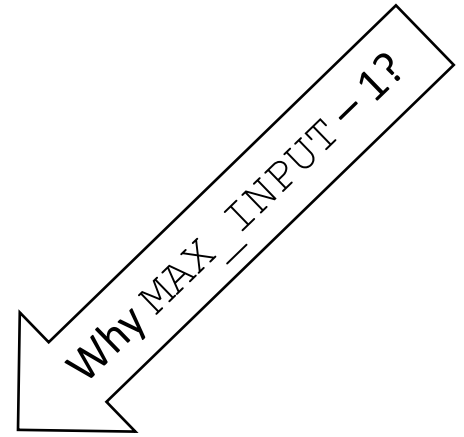
calculates the length of `string`

Parameters `string` a null-terminated string

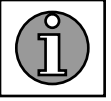
Return value the length of `string` not including the terminating `'\0'`, of type `size_t`



strlen()




```
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-1, 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

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

Replace \n with \0

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

```c
#include <stdio.h>

#define MAX 100

int main(void)
{
   char First[MAX] = {};
   char Middle[MAX] = {};
   char Last[MAX] = {};

   printf("Please enter your first name ");
   fgets(First, MAX-1, stdin);

   printf("Please enter your middle name ");
   fgets(Middle, MAX-1, stdin);

   printf("Please enter your last name ");
   fgets(Last, MAX-1, stdin);

   printf("Your name is %s %s %s", First, Middle, Last);

   return 0;
}
```

```
[frenchdm@omega ~]$ a.out
Please enter your first name Alan
Please enter your middle name Mathison
Please enter your last name Turing
Your name is Alan
 Mathison
 Turing
```

```
[frenchdm@omega ~]$ gcc RemNL.c -g
[frenchdm@omega ~]$ gdb a.out
GNU gdb (GDB) Red Hat Enterprise Linux (7.0.1-45.el5)
Copyright (C) 2009 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show
copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/f/fr/frenchdm/a.out...done.
(gdb) break main
Breakpoint 1 at 0x400573: file RemNL.c, line 9.
(gdb) run
Starting program: /home/f/fr/frenchdm/a.out
warning: no loadable sections found in added symbol-file system-supplied
DSO at 0x2aaaaaaab000
```

```
Breakpoint 1, main () at RemNL.c:9
9                   char First[MAX] = {};
(gdb) step
10                  char Middle[MAX] = {};
(gdb)
11                  char Last[MAX] = {};
(gdb)
13                  printf("Please your first name ");
(gdb)
14                  fgets(First, MAX-1, stdin);
(gdb)
Please enter your first name Alan
16                  printf("Please enter your middle name ");
(gdb) p First
$1 = "Alan\n", '\000' <repeats 94 times>
```

```
(gdb) step
17              fgets(Middle, MAX-1, stdin);
(gdb)
Please enter your middle name Mathison
19              printf("Please enter your last name ");
(gdb) p Middle
$2 = "Mathison\n", '\000' <repeats 90 times>
(gdb) step
20              fgets(Last, MAX-1, stdin);
(gdb)
Please enter your last name Turing
22              printf("Your name is %s %s %s", First, Middle, Last);
(gdb) p Last
$3 = "Turing\n", '\000' <repeats 92 times>
```

```
(gdb) step
Your name is Alan
 Mathison
 Turing
24                return 0;
(gdb) step
25        }
(gdb)
0x0000003ec941d9f4 in __libc_start_main () from /lib64/libc.so.6
(gdb)
Single stepping until exit from function __libc_start_main,
which has no line number information.

Program exited normally.
(gdb) quit
[frenchdm@omega ~]$
```

```
(gdb) p First
$1 = "Alan\n", '\000' <repeats 94 times>

(gdb) p Middle
$2 = "Mathison\n", '\000' <repeats 90 times>

(gdb) p Last
$3 = "Turing\n", '\000' <repeats 92 times>
```

```
printf("Your name is %s %s %s", First, Middle, Last);
```

Your name is Alan
Mathison
Turing

Compiler Warning

```
[frenchdm@omega ~]$ gcc RemNL.c -g
RemNL.c: In function ‘main’:
RemNL.c:15: warning: incompatible implicit
declaration of built-in function ‘strlen’
```

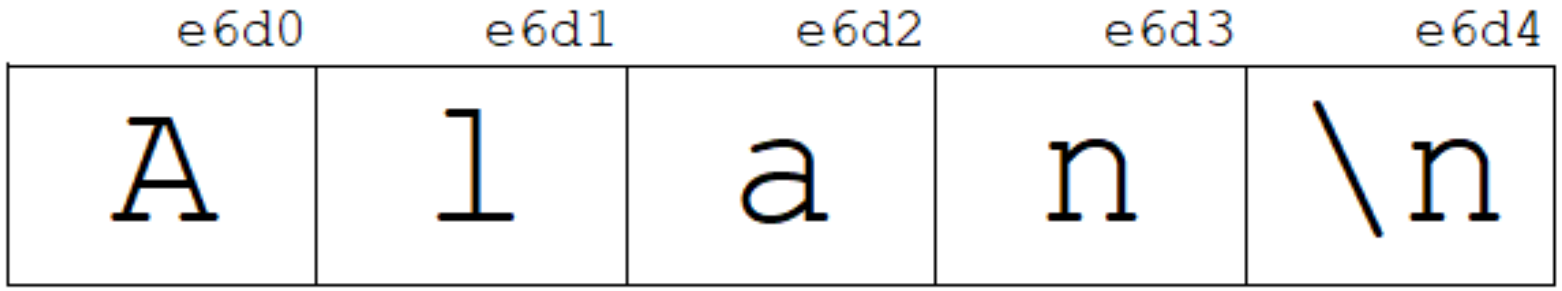
```
#include <string.h>
```

Removing the \n from an input string

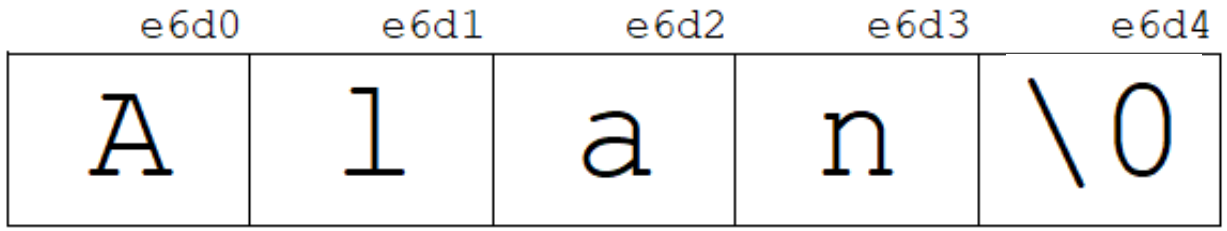
```
[frenchdm@omega ~]$ a.out
Please enter your first name Alan
Please enter your middle name Mathison
Please enter your last name Turing
Your name is Alan Mathison Turing
```

```
14      printf("Please enter your first name ");
15      fgets(First, MAX-1, stdin);
16      First[strlen(First) - 1] = '\0';
17
18      printf("Please enter your middle name ");
19      fgets(Middle, MAX-1, stdin);
20      Middle[strlen(Middle) - 1] = '\0';
21
22      printf("Please enter your last name ");
23      fgets(Last, MAX-1, stdin);
24      Last[strlen(Last) - 1] = '\0';
```

```
(gdb) p First
$1 = "Alan\n", '\000' <repeats 94 times>

(gdb) p &First
$8 = (char (*)[100]) 0x7fffffffe6d0

(gdb) p &First[0]
$9 = 0x7fffffffe6d0

(gdb) p &First[1]
$10 = 0x7fffffffe6d1

(gdb) p &First[2]
$11 = 0x7fffffffe6d2

(gdb) p &First[3]
$12 = 0x7fffffffe6d3

(gdb) p &First[4]
$13 = 0x7fffffffe6d4
```



```
(gdb) p First
$1 = "Alan\n", '\000' <repeats 94 times>

(gdb) p strlen(First)
$2 = 5

(gdb) p First[4]
$3 = 10 '\n'

(gdb) p First[strlen(First) - 1]
$4 = 10 '\n'
```



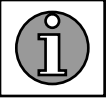

```
First[strlen(First) - 1] = '\0';

(gdb) p First
$5 = "Alan", '\000' <repeats 95 times>

(gdb) p First[4]
$6 = 0 '\000'
```

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-1, stdin);

strcpy(Buffer, UserString);

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

strcpy()

```
(gdb) p UserString1
$5 = "supercalifragilistexpialidocious\n\000\000\000\000\000

(gdb) p UserString2
$6 = "fly\n\000\000\000\000\227\a\000\000\001"

strcpy(UserString1, UserString2);

(gdb) p UserString1
$7 = "fly\n\000califragilistexpialidocious\n\000\000\000\000\000

(gdb) p UserString2
$8 = "fly\n\000\000\000\000\227\a\000\000\001"
```

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 *`	

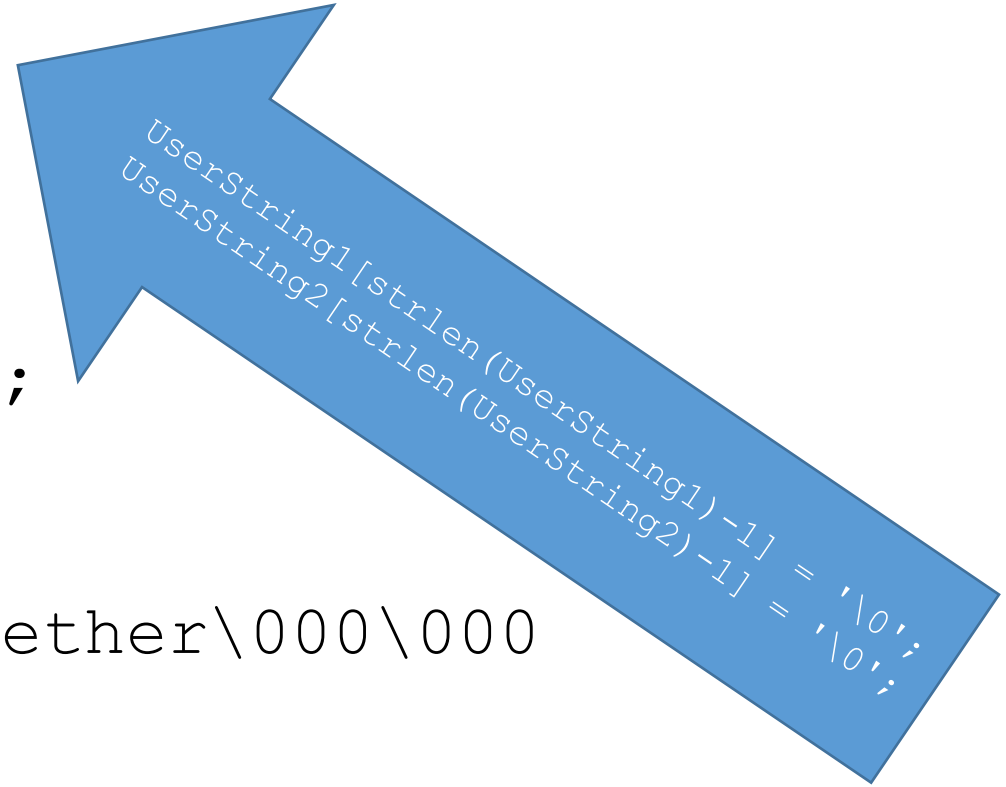
strcat()

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

strcat()

```
(gdb) p UserString1
$9 = "This string was glu\n\000\177\000\000\000\

(gdb) p UserString2
$10 = "ed together\n", '\000'

strcat(UserString1, UserString2);

(gdb) p UserString1
$11 = "This string was glued together\000\000

(gdb) p UserString2
$12 = "ed together", '\000'
```



The Common String Library Functions

`strncat(buffer, string, n)`

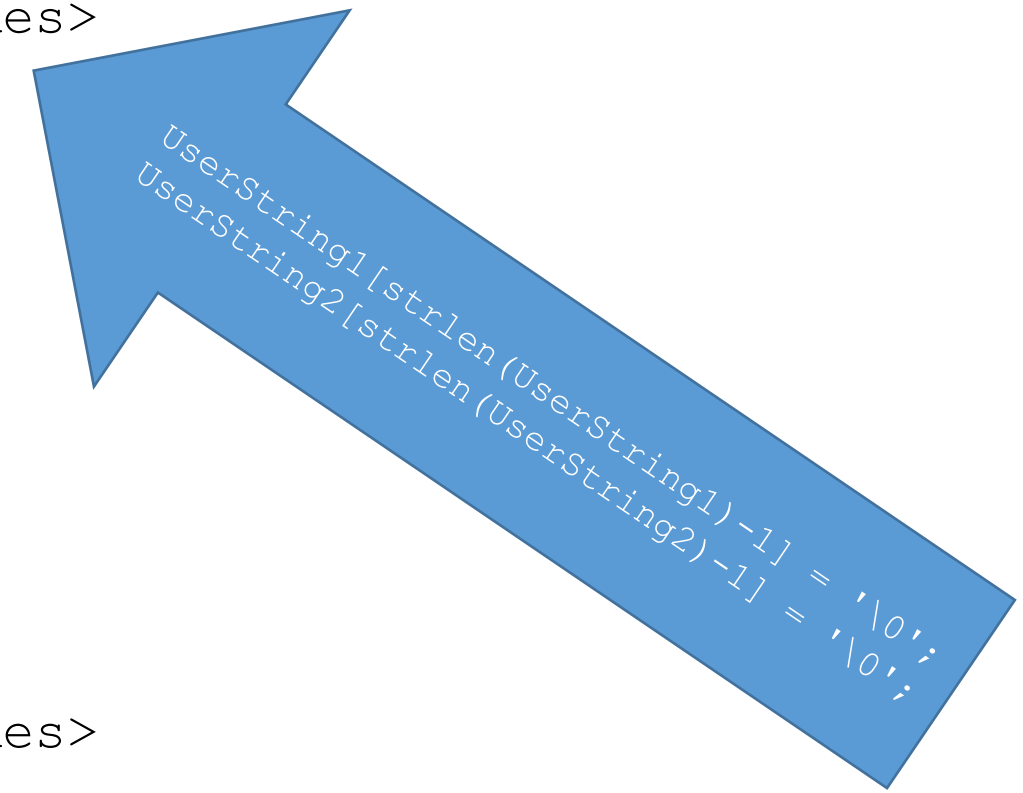
concatenates `string` onto the end of the current string in `buffer`

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 *`

strncat()

```
strncat(UserString1, UserString2, n);

(gdb) p UserString1
$1 = "This string was glu\000\000\177\000

(gdb) p UserString2
$2 = "ed together", '\000' <repeats 13 times>

(gdb) p n
$3 = 1
(gdb) step

(gdb) p UserString1
$4 = "This string was glue\000\177\000

(gdb) p UserString2
$5 = "ed together", '\000' <repeats 13 times>
```



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

strcmp()

stringtoolsDemo.c

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

```
Enter a line of text (max of 99)

apple

Enter a line of text (max of 99)

Apple


apple
>
Apple
```

```
Enter a line of text (max of 99)

Pear

Enter a line of text (max of 99)

PEar


Pear
>
PEar
```

```
Enter a line of text (max of 99)

Banana

Enter a line of text (max of 99)

Banana


Strings are identical
```

```
Enter a line of text (max of 99)

Zebra

Enter a line of text (max of 99)

apple


Zebra
<
apple
```

```
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");
```

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`.

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

```
Enter a line of text (max of 99)

apple

Enter a line of text (max of 99)

Apple

Enter how many letters to compare 2


apple
>
Apple
```

```
Enter a line of text (max of 99)

Pear

Enter a line of text (max of 99)

PEar

Enter how many letters to compare 1


Strings are identical for the first 1
characters
```

```
Enter a line of text (max of 99)

Banana

Enter a line of text (max of 99)

BanaNA

Enter how many letters to compare 4

Strings are identical for the first
4 characters
```

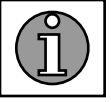
```
Enter a line of text (max of 99)

Zebra

Enter a line of text (max of 99)

apple

Enter how many letters to compare 2

Zebra
<
apple
```

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-1, 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);
```

```
Enter a line of text (max of 99)

encyclopedia

Enter a character

e


New version of String is

-ncyclop-dia
```

```
FirstOccur = strchr(UserString, Ch);

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

What would happen if I took out this line?

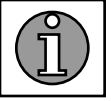
```
while (FirstOccur != NULL)
{
    FirstOccur = strchr(UserString, Ch);
}
```

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`.	



strstr()

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

```
Enter a line of text (max of 99)

supercalifragilisticexpialidocious

Enter word to search for (max of 99)

li


supercalifragilisticexpialidocious
contains
li

supercaXXfragiXXsticexpiaXXdocious
```

So how did this

```
supercalifragilisticexpialidocious
```

become this?

```
supercaXXfragiXXsticexpiaXXdocious
```

```
FirstOccur = strstr(UserString1, UserString2);

while (FirstOccur != NULL)
{
      distance = abs(UserString1 - FirstOccur);
      for (i = 0; i < strlen(UserString2); i++)
      {
            UserString1[distance + i] = 'X';
      }
      FirstOccur = strstr(UserString1, UserString2);
}

printf("%s\n", UserString1);
```

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.	

strpbrk()

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

```
Enter a line of text (max of 99)

supercalifragilisticexpialidocious

Enter characters to replace with _

aeiou



Replacing all instances of

        aeiou
with _

s_p_rc_l_fr_g_l_st_c_xp__l_d_c___s
```

s_percalifragilisticexpialidocious	s_p_rc_l_fr_g_l_st_c_xpialidocious
s_p_rcalifragilisticexpialidocious	s_p_rc_l_fr_g_l_st_c_xp_alidocious
s_p_rc_lifragilisticexpialidocious	s_p_rc_l_fr_g_l_st_c_xp__lidocious
s_p_rc_l_fragilisticexpialidocious	s_p_rc_l_fr_g_l_st_c_xp__l_docious
s_p_rc_l_fr_gilisticexpialidocious	s_p_rc_l_fr_g_l_st_c_xp__l_d_cious
s_p_rc_l_fr_g_listicexpialidocious	s_p_rc_l_fr_g_l_st_c_xp__l_d_c_ous
s_p_rc_l_fr_g_l_sticexpialidocious	s_p_rc_l_fr_g_l_st_c_xp__l_d_c__us
s_p_rc_l_fr_g_l_st_cexpialidocious	s_p_rc_l_fr_g_l_st_c_xp__l_d_c___s

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`	

strtok()

```
{Austin|817-DOG-1234|10}
{Jenny|867-5309|40}
{Prof French|817-272-0161|162}
{Fake Name|123-456-7890|-1}
```

strtok()

Enter the phrase like {Name|Phone|Age} to be tokenized **{Austin|817-DOG-1234|10}**
Hello Austin - your phone number is 817-DOG-1234 and you are 10 years old

Enter the phrase like {Name|Phone|Age} to be tokenized **{Jenny|867-5309|40}**
Hello Jenny - your phone number is 867-5309 and you are 40 years old

Enter the phrase like {Name|Phone|Age} to be tokenized **{Prof French|817-272-0161|162}**
Hello Prof French - your phone number is 817-272-0161 and you are 162 years old

Enter the phrase like {Name|Phone|Age} to be tokenized **{Fake Name|123-456-7890|-1}**
Hello Fake Name - your phone number is 123-456-7890 and you are -1 years old

```
char *Delimiters = "{}|";
char *Token = NULL;
char TokenPhrase[PHRASELEN] = {};
char Name[PHRASELEN] = {};
char Phone[PHRASELEN] = {};
int age = 0;

printf("Enter the phrase like {Name|Phone|Age} to be tokenized ");
fgets(TokenPhrase, PHRASELEN-1, stdin);
TokenPhrase[strlen(TokenPhrase)-1] = '\0';

Token = strtok(TokenPhrase, Delimiters);
strcpy(Name, Token);

Token = strtok(NULL, Delimiters);
strcpy(Phone, Token);

Token = strtok(NULL, Delimiters);
age = atoi(Token);

printf("Hello %s - your phone number is %s and you are %d years old\n",
        Name, Phone, age);
```

strtok()

Enter the phrase like {Name|Phone|Age} to be tokenized **{Austin|817-DOG-1234|10}**
Hello Austin - your phone number is 817-DOG-1234 and you are 10 years old

Enter the phrase like {Name|Phone|Age} to be tokenized **{Jenny|867-5309|40}**
Hello Jenny - your phone number is 867-5309 and you are 40 years old

Enter the phrase like {Name|Phone|Age} to be tokenized **{Prof French|817-272-0161|162}**
Hello Prof French - your phone number is 817-272-0161 and you are 162 years old

Enter the phrase like {Name|Phone|Age} to be tokenized **{Fake Name|123-456-7890|-1}**
Hello Fake Name - your phone number is 123-456-7890 and you are -1 years old

```c
#include <stdio.h>
#include <string.h>

#define MAX 80

int main(void)
{
    char *Token = NULL;
    char Buffer[MAX] = {};

    printf("Enter 3 words separated by commas ");
    fgets(Buffer, MAX-1, stdin);

    Token = strtok(Buffer, ",");
    printf("The first word is %s\n", Token);
    Token = strtok(NULL, ",");
    printf("The second word is %s\n", Token);
    Token = strtok(NULL, ",");
    printf("The third word is %s\n", Token);

    return 0;
}
```

`strtok()` returns a `char` pointer so we need a variable to hold that address

```c
char *Token = NULL;
```

`strtok()` takes 2 parameters

First parameter is a string

We declared the array `Buffer` and used `fgets()` to make a string (null terminated array)

Second parameter is the string of delimiters

this can a quoted string

or

it can be a null terminated array (string)

After the `fgets()`

```
(gdb) p Buffer
$1 = "a,b,c\n", '\000' <repeats 73 times>

(gdb) p &Buffer
$3 = (char (*)[80]) 0x7fffffffe6d0
```

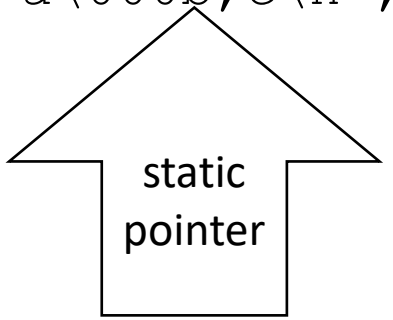
```
Token = strtok(Buffer, ",");
```

`strtok()` will find the first occurrence of the delimiter in `Buffer`. It will replace the delimiter with `NULL`.

```
(gdb) p Buffer
$5 = "a\000b,c\n", '\000' <repeats 73 times>
```

`strtok()` then creates an internal static pointer that points to the address of the character just past the newly inserted `NULL`. We cannot see/access this pointer.

```
(gdb) p Buffer
$5 = "a\000b,c\n", '\000' <repeats 73 times>
```



This static pointer is separate from the `NULL` the delimiter was replaced with previously.

```
(gdb) p Buffer
$5 = "a\000b,c\n", '\000' <repeats 73 times>

(gdb) p &Buffer
$1 = (char (*)[80]) 0x7fffffffe6d0
```

`strtok()` returns the pointer to the token

```
(gdb) p Token
$4 = 0x7fffffffe6d0 "a"
```

For the first call to `strtok()`, this address is same as the address of the string being tokenized.

```
Token = strtok(Buffer, ",");
```

We can this print the token

```
printf("The first word is %s\n", Token);
```

Remember that `printf()` with `%s` is looking for a pointer/address of a null terminated string.

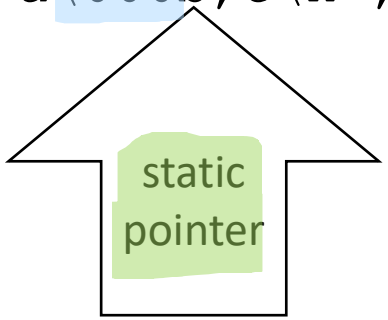
The second time we call `strtok()` to get the next token, we call it with a first parameter of `NULL` instead of the string (like we did the first time).

```
Token = strtok(NULL, ",");
```

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

This usage of `NULL` is not referring to the `NULL` that `strtok()` put in place of your delimiter in the string itself.

```
(gdb) p Buffer
$5 = "a\000b,c\n", '\000' <repeats 73 times>
```

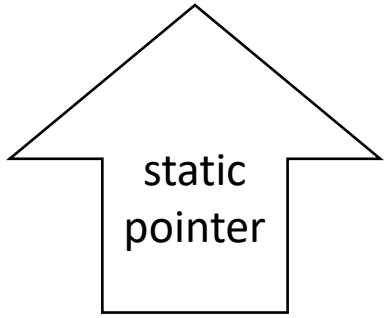


The second time we call `strtok()` to get the next token, we call it with a first parameter of `NULL` instead of the string (like we did the first time).

```
Token = strtok(NULL, ",");
```

This replaces the second delimiter with a `NULL` and moves the static pointer to the next character after the newly inserted `NULL`.

```
(gdb) p Buffer
$2 = "a\000b\000c\n", '\000' <repeats 73 times>
```



The address of the second token is returned by `strtok()` and stored in `Token`.

If you pass the string in any call to `strtok()` after the first one, 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.

```
Token = strtok(Buffer, ",");
printf("The first word is %s\n", Token);
Token = strtok(Buffer, ",");
printf("The second word is %s\n", Token);
Token = strtok(Buffer, ",");
printf("The third word is %s\n", Token);

[frenchdm@omega ~]$ a.out
Enter 3 words separated by commas a,b,c
The first word is a
The second word is a
The third word is a
[frenchdm@omega ~]$
```

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);
}
```

```
#include <stdio.h>
#include <string.h>

#define MAX 80

int main(void)
{
  char *Token
  char Buffer[

  printf("Ente
  fgets(Buffer

  // strtok()

  return 0;
}
```

```
Token = strtok(Buffer, ",");
printf("The first word is %s\n", Token);
Token = strtok(NULL, ",");
printf("The second word is %s\n", Token);
Token = strtok(NULL, ",");
printf("The third word is %s\n", Token);
```