CS214 Recitation Sec.7

Oct.10, 2017

Topics

1. HW3 answers

2. A question about using pointers

3. HW4: Implementing “ls” command in C

HW3 - Q0

0. What are the differences between strlen and sizeof a string in C? Why?

HW3 - Answer to Q0

	sizeof	strlen
what is it	operator	function in <string.h>
when to be computed	preprocessing	when it is running
function	get the size that could hold the biggest possible size of the object	return the length of a string
parameter	array,pointer,type,function(return value),struct	only char*
need to notice	can’t use for dynamic allocated memory	length not include ‘\0’
example	char arr[10] = "What?"; int len_two = sizeof(arr); 10	char arr[10] = "What?"; int len_one = strlen(arr); 5

HW3 - Q1

1. Write the function: replace(char string[], char from[], char to[])

which finds the string from in the string string and replaces it with the string to. You may assume that from and to are the same length. For example, the code

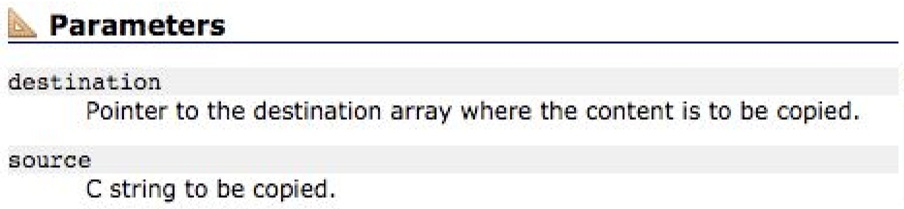
```
    char string[] = "recieve";
    replace(string, "ie", "ei");
```

should change string to "receive".

HW3 - Q1

functions in <string.h>

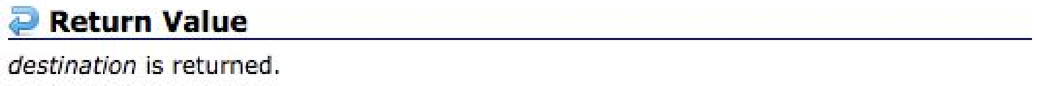
strcpy(): char * strcpy (char * destination, const char * source);



Parameters

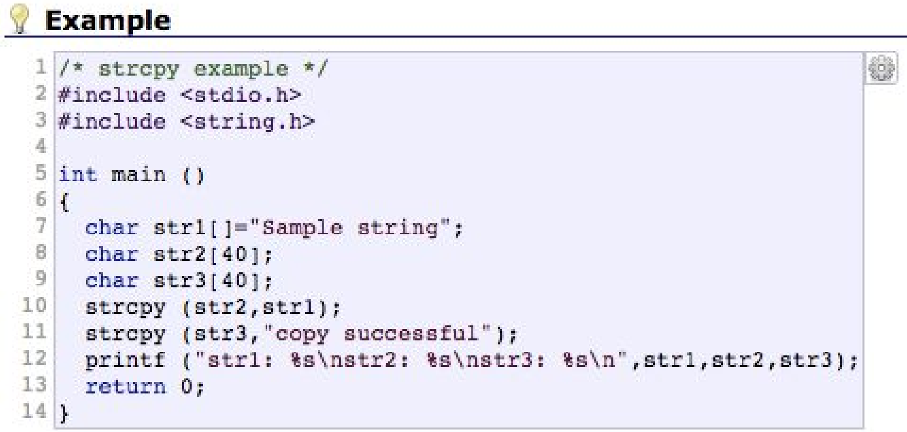
`destination`
: Pointer to the destination array where the content is to be copied.

`source`
: C string to be copied.



Return Value

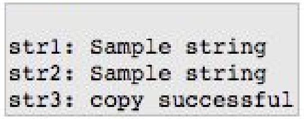
destination is returned.



Example

```c
/* strcpy example */
#include <stdio.h>
#include <string.h>

int main ()
{
  char str1[]="Sample string";
  char str2[40];
  char str3[40];
  strcpy (str2,str1);
  strcpy (str3,"copy successful");
  printf ("str1: %s\nstr2: %s\nstr3: %s\n",str1,str2,str3);
  return 0;
}
```

Output:

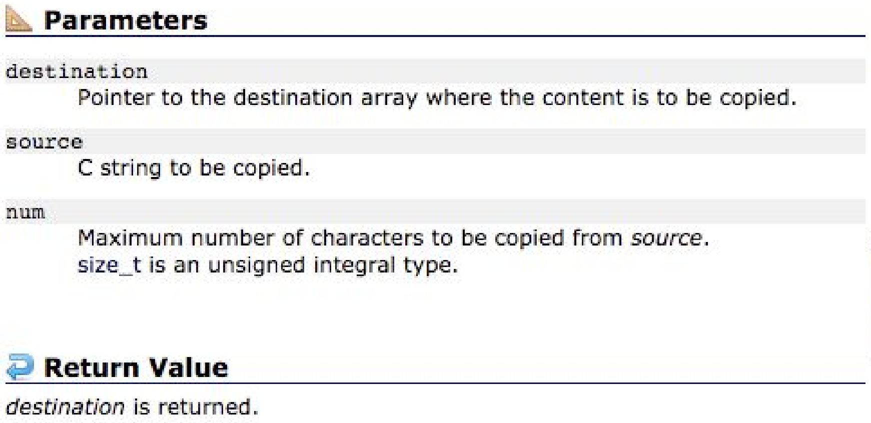


```
str1: Sample string
str2: Sample string
str3: copy successful
```

HW3 - Q1

functions in <string.h>

strncpy(): char * strncpy (char * destination, const char * source, size_t num);



Parameters

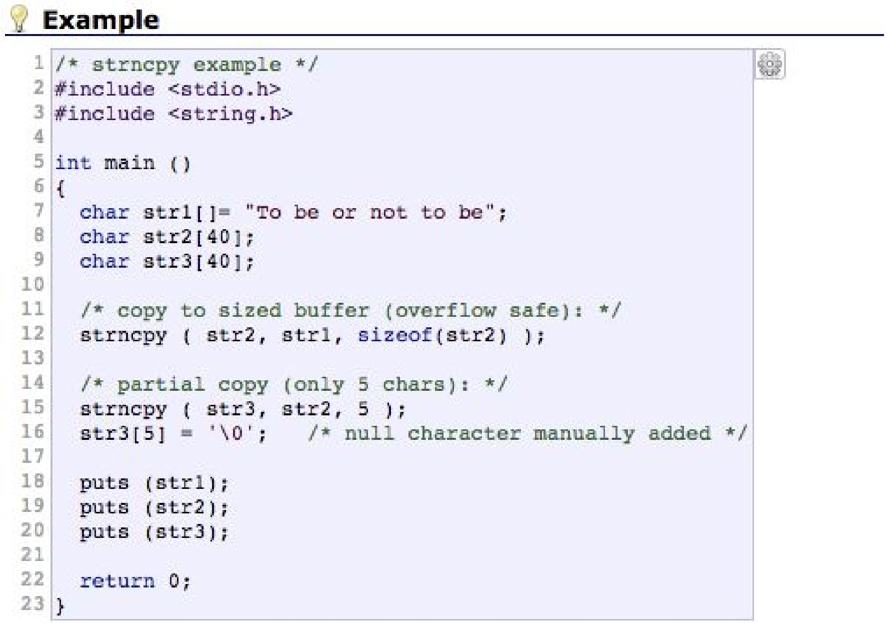
destination
: Pointer to the destination array where the content is to be copied.

source
: C string to be copied.

num
: Maximum number of characters to be copied from *source*.
size_t is an unsigned integral type.

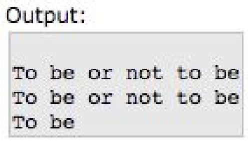
Return Value

destination is returned.



Example

```c
/* strncpy example */
#include <stdio.h>
#include <string.h>

int main ()
{
  char str1[]= "To be or not to be";
  char str2[40];
  char str3[40];

  /* copy to sized buffer (overflow safe): */
  strncpy ( str2, str1, sizeof(str2) );

  /* partial copy (only 5 chars): */
  strncpy ( str3, str2, 5 );
  str3[5] = '\0';   /* null character manually added */

  puts (str1);
  puts (str2);
  puts (str3);

  return 0;
}
```



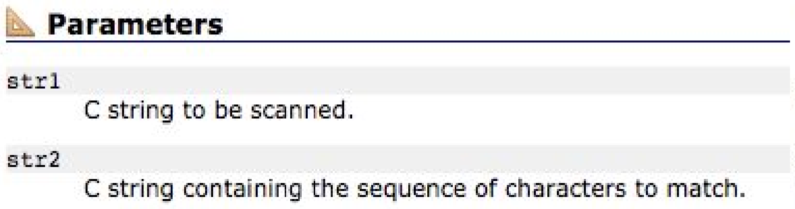
Output:

```
To be or not to be
To be or not to be
To be
```

HW3 - Q1

functions in <string.h>

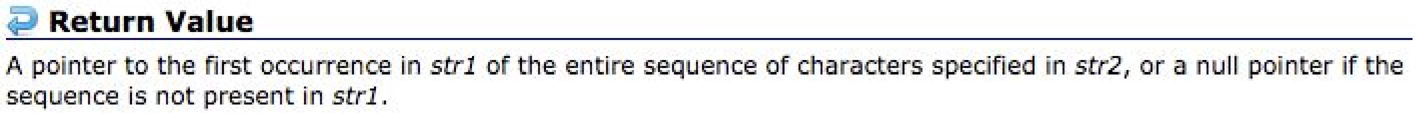
strstr(): char * strstr (char * str1, const char * str2);



Parameters

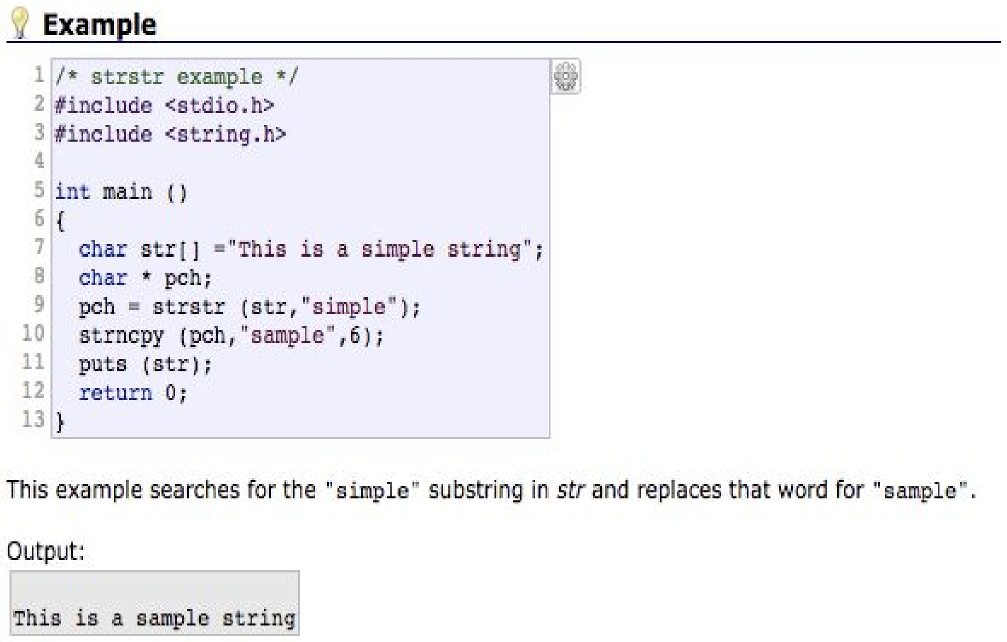
`str1`
: C string to be scanned.

`str2`
: C string containing the sequence of characters to match.



Return Value

A pointer to the first occurrence in *str1* of the entire sequence of characters specified in *str2*, or a null pointer if the sequence is not present in *str1*.



Example

```c
/* strstr example */
#include <stdio.h>
#include <string.h>

int main ()
{
  char str[] ="This is a simple string";
  char * pch;
  pch = strstr (str,"simple");
  strncpy (pch,"sample",6);
  puts (str);
  return 0;
}
```

This example searches for the "`simple`" substring in *str* and replaces that word for "`sample`".

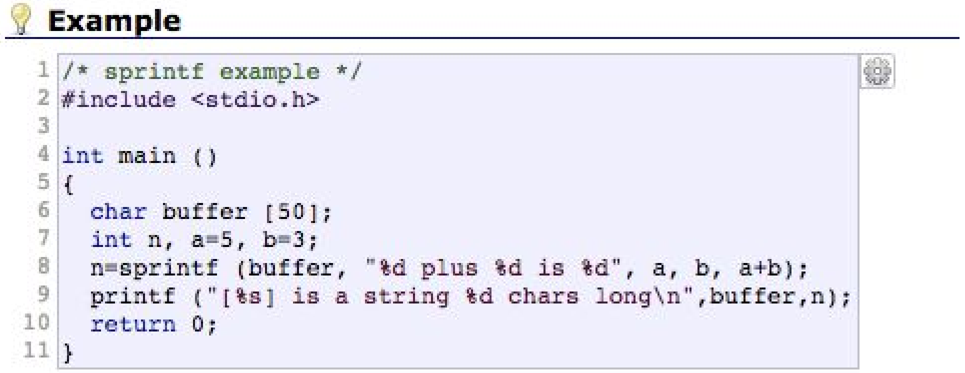
Output:

```
This is a sample string
```

HW3 - Q1

functions in <stdio.h>

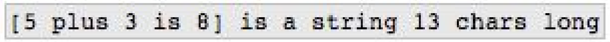
sprintf(): int sprintf (char * str, const char * format, ...);



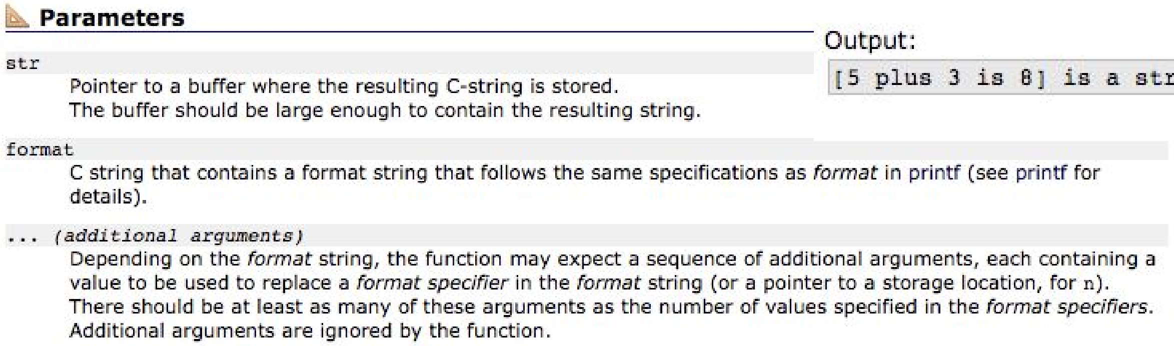
Example

```
/* sprintf example */
#include <stdio.h>

int main ()
{
  char buffer [50];
  int n, a=5, b=3;
  n=sprintf (buffer, "%d plus %d is %d", a, b, a+b);
  printf ("[%s] is a string %d chars long\n",buffer,n);
  return 0;
}
```

Output:



```
[5 plus 3 is 8] is a string 13 chars long
```

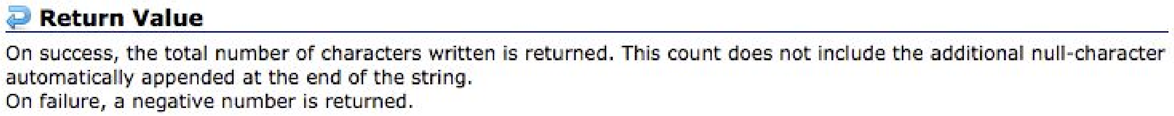


Parameters

`str`
: Pointer to a buffer where the resulting C-string is stored.
The buffer should be large enough to contain the resulting string.

`format`
: C string that contains a format string that follows the same specifications as *format* in printf (see printf for details).

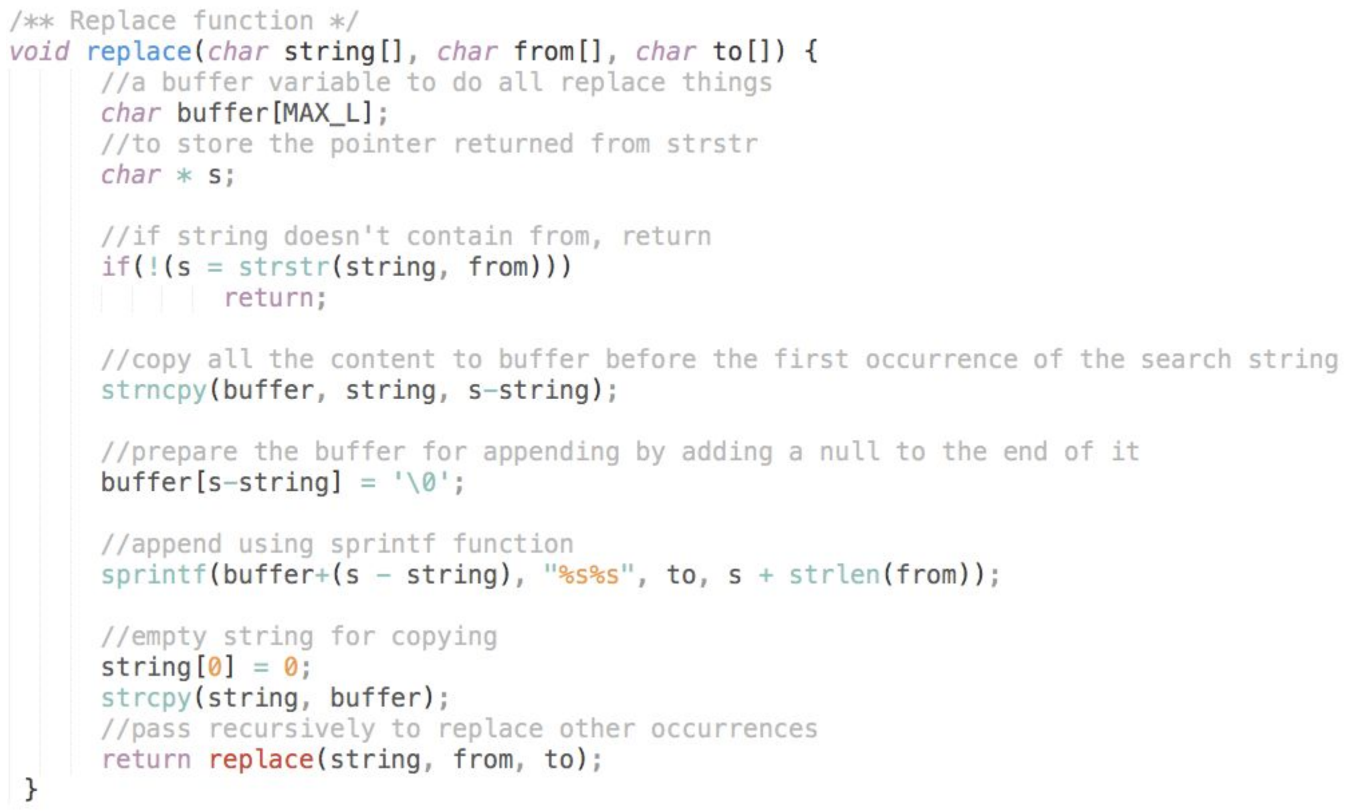
`... (additional arguments)`
: Depending on the *format* string, the function may expect a sequence of additional arguments, each containing a value to be used to replace a *format specifier* in the *format* string (or a pointer to a storage location, for `n`).
There should be at least as many of these arguments as the number of values specified in the *format specifiers*. Additional arguments are ignored by the function.



Return Value

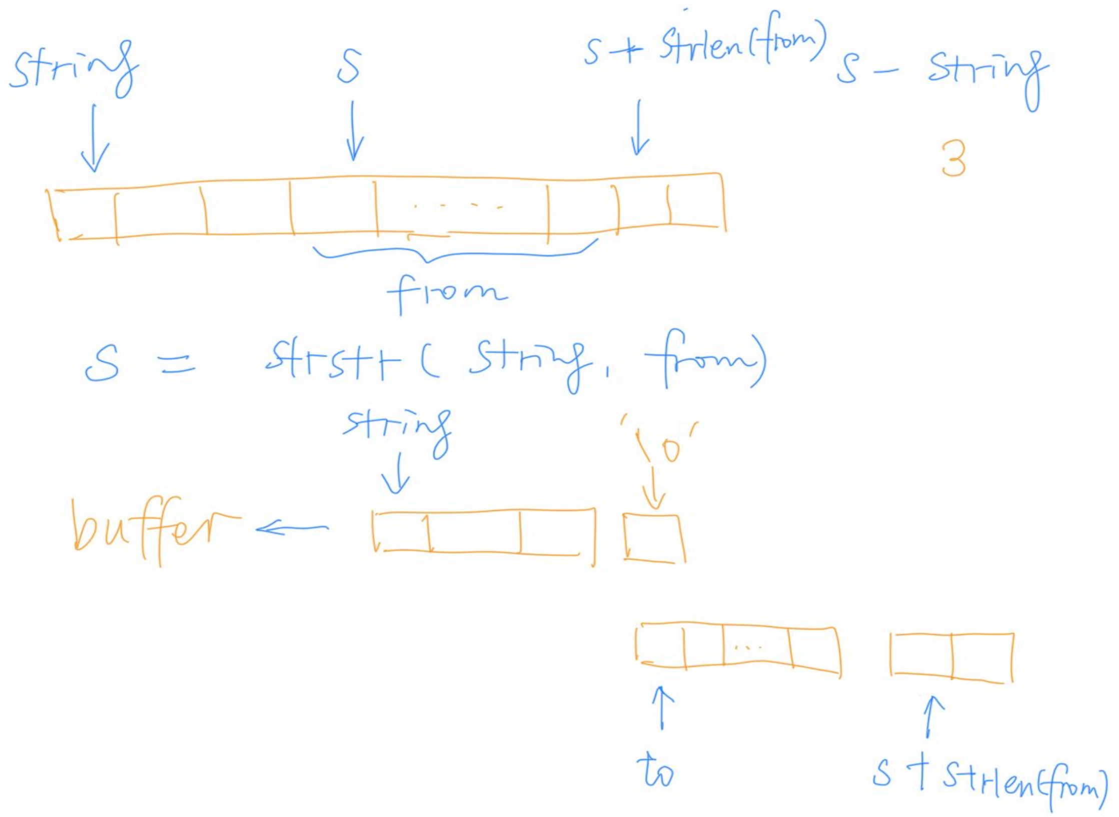
On success, the total number of characters written is returned. This count does not include the additional null-character automatically appended at the end of the string.
On failure, a negative number is returned.

HW3 - Q1

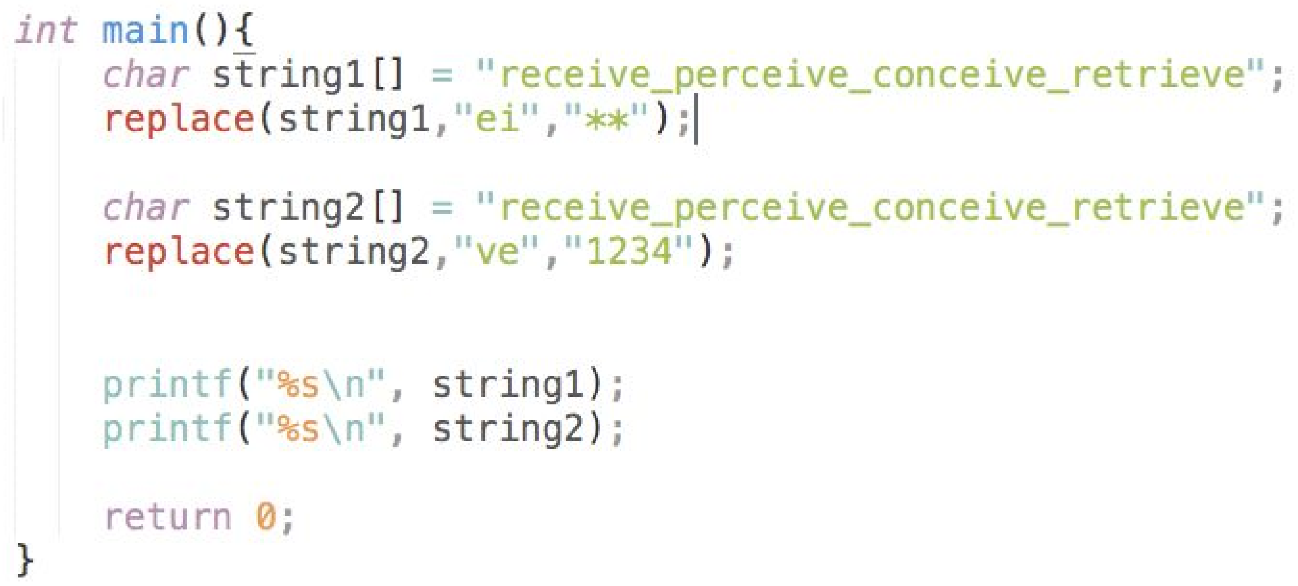


```c
/** Replace function */
void replace(char string[], char from[], char to[]) {
    //a buffer variable to do all replace things
    char buffer[MAX_L];
    //to store the pointer returned from strstr
    char * s;

    //if string doesn't contain from, return
    if(!(s = strstr(string, from)))
            return;

    //copy all the content to buffer before the first occurrence of the search string
    strncpy(buffer, string, s-string);

    //prepare the buffer for appending by adding a null to the end of it
    buffer[s-string] = '\0';

    //append using sprintf function
    sprintf(buffer+(s - string), "%s%s", to, s + strlen(from));

    //empty string for copying
    string[0] = 0;
    strcpy(string, buffer);
    //pass recursively to replace other occurrences
    return replace(string, from, to);
}
```

HW3 - Q1


string
s
s + strlen(from)
s - string
3
from
s = strstr(string, from)
string
'\0'
buffer
to
s + strlen(from)

HW3 - Q1



```
int main(){
    char string1[] = "receive_perceive_conceive_retrieve";
    replace(string1,"ei","**");

    char string2[] = "receive_perceive_conceive_retrieve";
    replace(string2,"ve","1234");


    printf("%s\n", string1);
    printf("%s\n", string2);

    return 0;
}
```

HW3 - Q1

1. Write the function: replace(char string[], char from[], char to[])

which finds the string from in the string string and replaces it with the string to. You may assume that from and to are the same length. For example, the code

```
char string[] = "recieve";
replace(string, "ie", "ei");
```

How to improve?

should change string to "receive".

why professor mentions the same length here?   how about to = “ie” or “cie”?

segment fault

HW3 - Q2

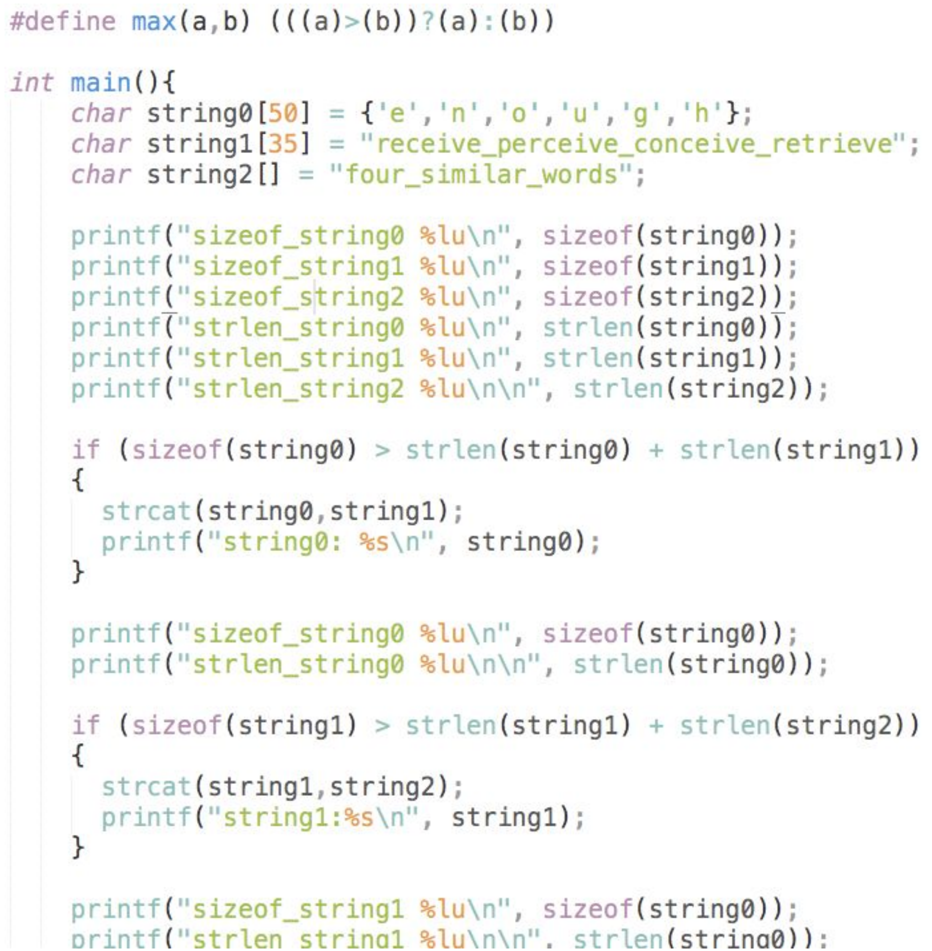
Write a short program to read two lines of text, and concatenate them using strcat. Since strcat concatenates in-place, you'll have to make sure you have enough memory to hold the concatenated copy.

For now, use a char array which is twice as big as either of the arrays you use for reading the two lines. Use strcpy to copy the first string to the destination array, and strcat to append the second one.

HW3 - Q2

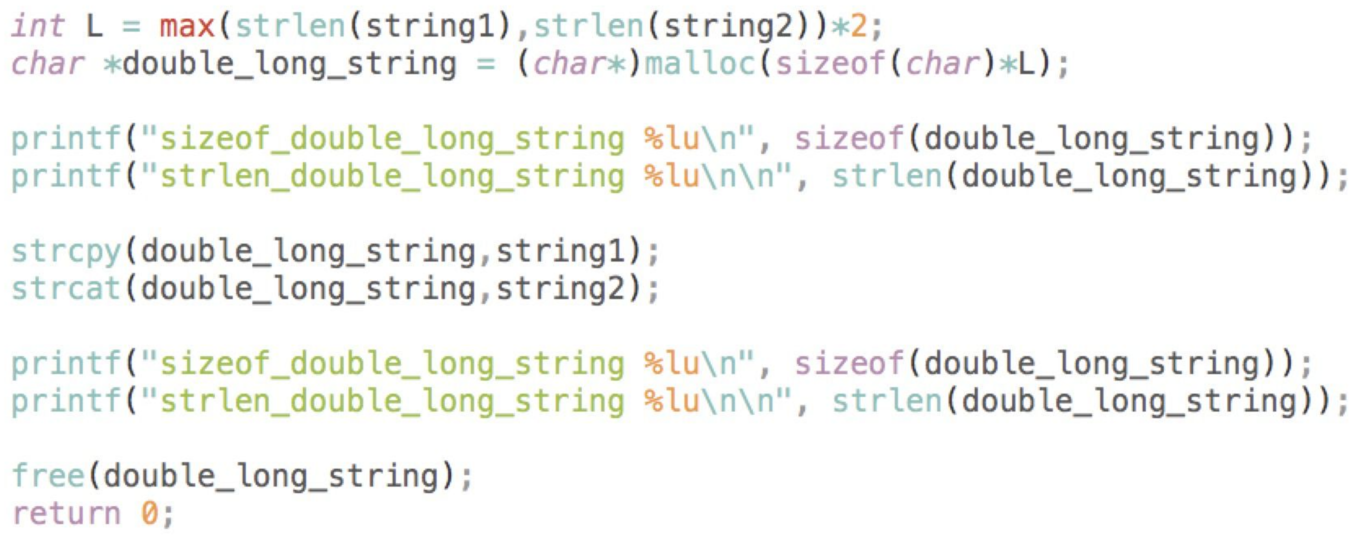
Write a short program to read two lines of text, and concatenate them using strcat. Since strcat concatenates in-place, you'll have to make sure you have enough memory to hold the concatenated copy.

For now, use a char array which is twice as big as either of the arrays you use for reading the two lines. Use strcpy to copy the first string to the destination array, and strcat to append the second one.



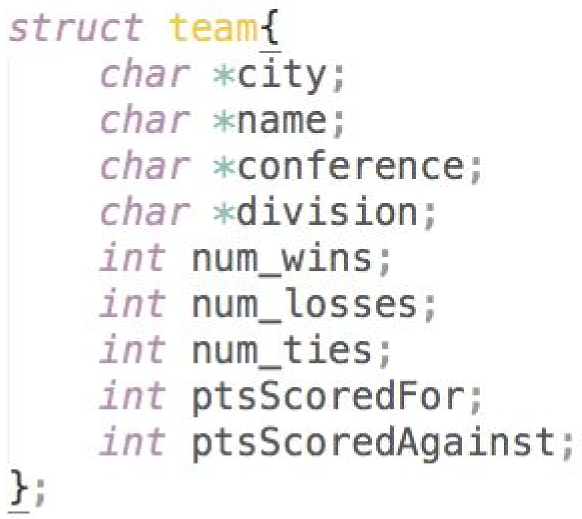
```c
#define max(a,b) (((a)>(b))?(a):(b))

int main(){
    char string0[50] = {'e','n','o','u','g','h'};
    char string1[35] = "receive_perceive_conceive_retrieve";
    char string2[] = "four_similar_words";

    printf("sizeof_string0 %lu\n", sizeof(string0));
    printf("sizeof_string1 %lu\n", sizeof(string1));
    printf("sizeof_string2 %lu\n", sizeof(string2));
    printf("strlen_string0 %lu\n", strlen(string0));
    printf("strlen_string1 %lu\n", strlen(string1));
    printf("strlen_string2 %lu\n\n", strlen(string2));

    if (sizeof(string0) > strlen(string0) + strlen(string1))
    {
     strcat(string0,string1);
     printf("string0: %s\n", string0);
    }

    printf("sizeof_string0 %lu\n", sizeof(string0));
    printf("strlen_string0 %lu\n\n", strlen(string0));

    if (sizeof(string1) > strlen(string1) + strlen(string2))
    {
     strcat(string1,string2);
     printf("string1:%s\n", string1);
    }

    printf("sizeof_string1 %lu\n", sizeof(string0));
    printf("strlen_string1 %lu\n\n", strlen(string0));
```

HW3 - Q2

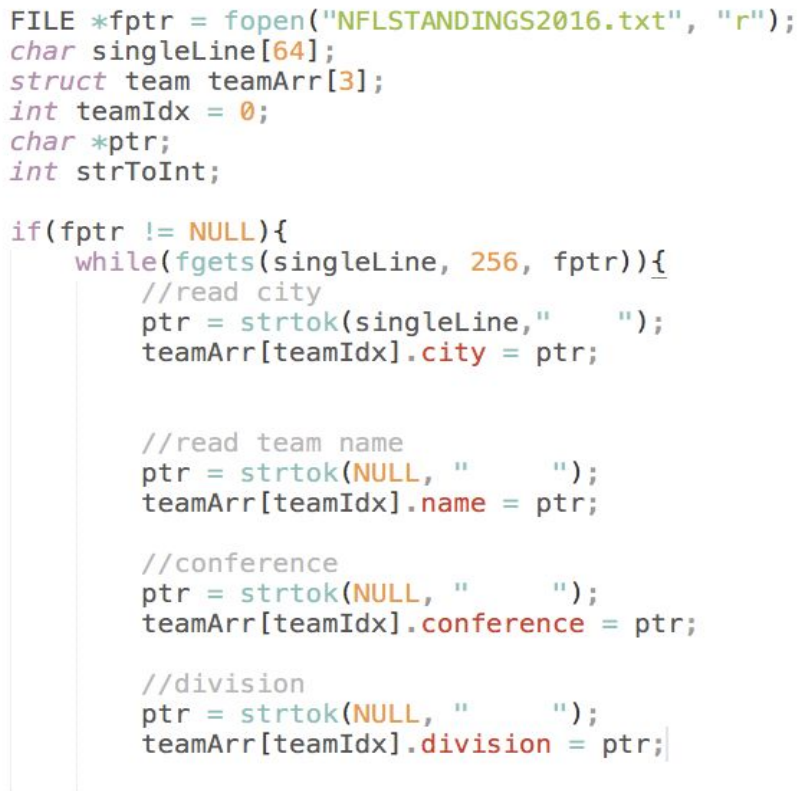


```c
int L = max(strlen(string1),strlen(string2))*2;
char *double_long_string = (char*)malloc(sizeof(char)*L);

printf("sizeof_double_long_string %lu\n", sizeof(double_long_string));
printf("strlen_double_long_string %lu\n\n", strlen(double_long_string));

strcpy(double_long_string,string1);
strcat(double_long_string,string2);

printf("sizeof_double_long_string %lu\n", sizeof(double_long_string));
printf("strlen_double_long_string %lu\n\n", strlen(double_long_string));

free(double_long_string);
return 0;
```

A question about using pointers

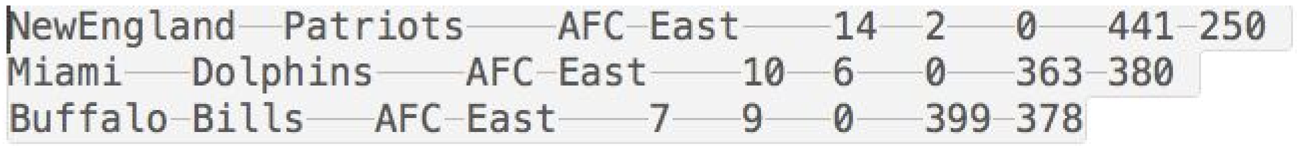


```c
struct team{
    char *city;
    char *name;
    char *conference;
    char *division;
    int num_wins;
    int num_losses;
    int num_ties;
    int ptsScoredFor;
    int ptsScoredAgainst;
};
```

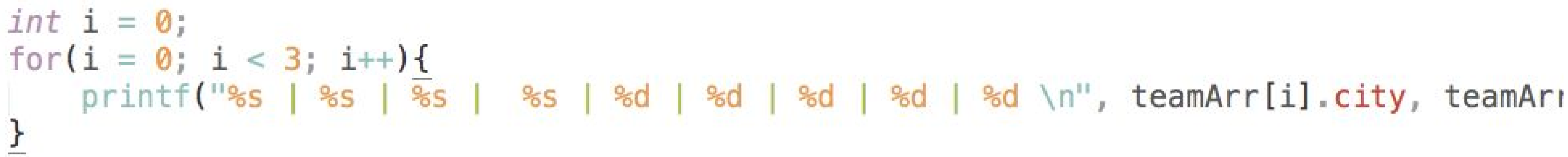


```c
FILE *fptr = fopen("NFLSTANDINGS2016.txt", "r");
char singleLine[64];
struct team teamArr[3];
int teamIdx = 0;
char *ptr;
int strToInt;

if(fptr != NULL){
    while(fgets(singleLine, 256, fptr)){
        //read city
        ptr = strtok(singleLine,"    ");
        teamArr[teamIdx].city = ptr;


        //read team name
        ptr = strtok(NULL, "     ");
        teamArr[teamIdx].name = ptr;

        //conference
        ptr = strtok(NULL, "     ");
        teamArr[teamIdx].conference = ptr;

        //division
        ptr = strtok(NULL, "     ");
        teamArr[teamIdx].division = ptr;
```

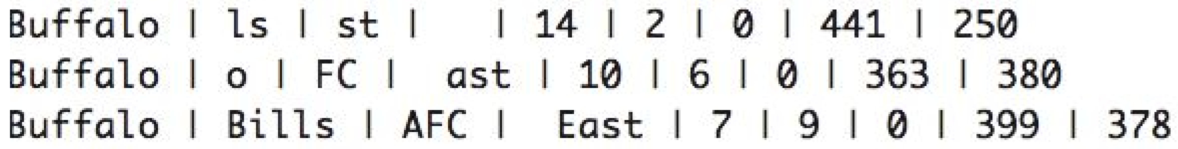
A question about using pointers (Cont.)



```
NewEngland  Patriots    AFC East    14  2   0   441 250
Miami   Dolphins    AFC East    10  6   0   363 380
Buffalo Bills   AFC East    7   9   0   399 378
```



```
int i = 0;
for(i = 0; i < 3; i++){
    printf("%s | %s | %s |  %s | %d | %d | %d | %d | %d \n", teamArr[i].city, teamAr
}
```



```
Buffalo | ls | st |    | 14 | 2 | 0 | 441 | 250
Buffalo | o | FC |  ast | 10 | 6 | 0 | 363 | 380
Buffalo | Bills | AFC |  East | 7 | 9 | 0 | 399 | 378
```

HW4.0 - Implementing "ls" command in C

0. Using opendir and readdir, open the current directory and output all filenames until there are no more

```
char * base = "./";
DIR * thingy = opendir(base);
dirent * newfile = readdir(thingy);
```

HW4.1 - Implementing "ls" command in C

1. Parse the dirent struct to see if an entry is a directory or a file. If it is a file, prepend "./" to the filename, if it is a directory, don't.

```
  .. if newfile != NULL
  //check type field of newfile dirent struct to determine the type of this file endpoint
newfile->d_type
// compare with system defines for different endpoint types (3rd paragraph under 'NOTES' in man 3 readdir).
  ... if == DT_REG //regular file
  elseif == DT_DIR //directory
....
```

HW4.2 - Implementing "ls" command in C

2. Open a file handle to each file and use lseek to determine the file's size in bytes, and print out the file's size next to its name.

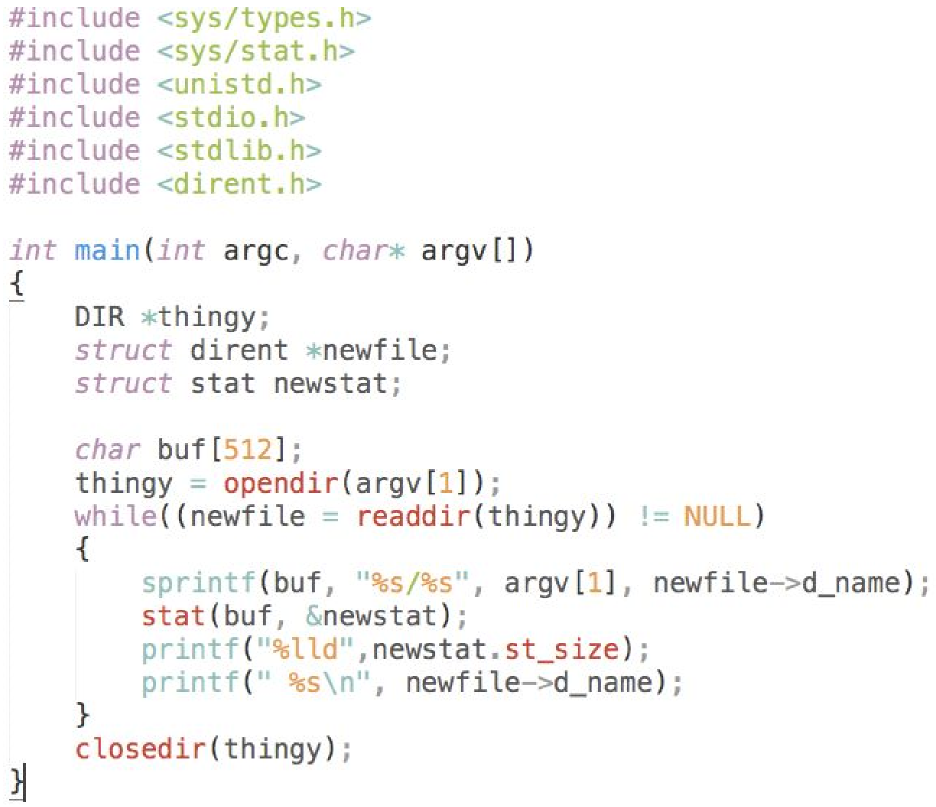
```
//assemble name of file using base directory and current path/name // concatenate all path up until now...
strcat(newpath, base)
// add currend name if it is a file...
newerpath = realloc(newpath, strlen(newpath)+strlen(newfile->d_name));
// d_name is REQUIRED to have a terminating null byte by standard ... yippee!
strcat(newerpath, newfile->d_name);

int checkFD = open(newerpath, RD_ONLY);
 ... if no error...
int len = lseek(checkFD, 0, SEEK_END);
close(checkFD);

printf( filename with full path, either color to indicate file/dir or put a "/" at the end to indicate dir, and number of
bytes of size, if a file )


//be sure to closedir() when done with dir descriptor
```

HW4.3 - Implementing "ls" command in C

3. Add a recursive element. If you find a directory, recursively call your code on that directory and prepend that directory name to each filename and directory name outputted.

```
elseif newfile->d_type == DT_DIR
strcat(newpath, base)
// add currend name if it is a file...
newerpath = realloc(newpath,
strlen(newpath)+strlen(newfile->d_name));
... recursively call my_LS on newerpath
```

HW4 - A Simple Reference



```c
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>

int main(int argc, char* argv[])
{
    DIR *thingy;
    struct dirent *newfile;
    struct stat newstat;

    char buf[512];
    thingy = opendir(argv[1]);
    while((newfile = readdir(thingy)) != NULL)
    {
        sprintf(buf, "%s/%s", argv[1], newfile->d_name);
        stat(buf, &newstat);
        printf("%lld",newstat.st_size);
        printf(" %s\n", newfile->d_name);
    }
    closedir(thingy);
}
```

Good luck with your midterm~