CSC 373 Winter 2020 Prof. Lytinen
Homework Assignment 2

Due: as specified on D2L

This assignment will be graded on a scale of 0 to 10 points. There are 5 functions for you to complete, each worth 2 points. The functions focus primarily on the use of C strings. You may receive 2 points if your function works correctly, 1.5 points of partial credit for a problem if your function compiles and works in most cases, or 1 point for a problem if the function compiles and works in some (but not all) cases. No credit will be given for functions that do not compile or that dot work properly in any test cases.

While you may talk with other students about general topics, you must write the C code in this homework by yourself. Please note my homework policies as described in the course syllabus. In particular, regarding academic integrity:

- All work must be your own, without any collaboration from anyone; unless specified otherwise for a specific assignment.
- Copying or paraphrasing a solution which you find from any source (**including on the Web**) is considered plagiarism.

Violating these policies can jeopardize your standing in this course and eventually at DePaul.

You should submit your assignment solution by uploading it into the D2L "HW2" submissions folder. You should turn a file called **hw2.c**, which contains your solutions for the 5 problems I've asked you to write. In the **hw2.c** file that you turn in, do **not** include any definition of the **main** function. I will test your code using my own (possibly different) main function.

Note that I have provided a template file for you to start with, called **hw2.c.** You should complete your 5 functions in this file. I am also providing a **hw2main.c** file in which I will write various test function calls to the functions you write. Finally, you should find a txt file which contains sample output from once I completed my own code.

I reserve the right to change **hw2main** for the purposes of grading, so you should make sure that you test your code for a wider variety of test cases. When you're done, all you need to turn in is **hw2.c**.

Your assignment is to write your own version of some of the functions in the built-in `<string.h>` C library. As you write these functions, keep in mind that a string in C is represented as a `char` array, with the '\0' character at the end of the string. Therefore, when a string is passed as a parameter, the length of the string does **not** necessarily need to be passed as a separate parameter. The string itself may or may not completely fill the array.

Note that the main function that I have provided **does** use <string.h> as it constructs test strings to pass to your functions. **However, your solutions** for the 5 functions below **may not use any of the built-in C string functions from the `<string.h>` library.**

1. Write a function called `streql373.` It is passed two parameters, both of which are C strings. It returns 0 if the strings consist of the same sequence of characters (with a '\0' at the end), or a non-zero integer otherwise. **You should use pointer syntax when writing this function;** that is you may use & and/or *, but not [ ].

   Here are some examples of how your code should work:

   `streql373("bin", "bag"); //` returns 0
   `streql373("computer", "game");` // returns 0
   `streql373("computer", "computer");` // returns 1
   (or other non-zero int
   `streql373("are", "area");` // returns 0.

2. Write a function called `strcmp373.` This function is passed two parameters, both of which are C strings. **You should use array syntax when writing this function**; that is, you may use [ ], but not * or &. The function should return an int as follows:

   a. A negative number if the first string is alphabetically before the second string. In C, the return value of `strcmp` is a number which reflects the difference in the ASCII codes for the first 2 letters which are not the same. For example, a call to `strcmp("programming", "project")` returns -3, since 'g' - 'j' in ASCII is -3.
   b. Zero if the strings are the same
   c. A positive number if the second string is alphabetically before the first string. Again, the number is the difference in ASCII codes between the first 2 letters which are not the same.

   Here are some examples of how your code should work:

   `strcmp373("bin", "bag"); //` returns 8
   `strcmp373("computer", "game");` // returns -4

```
strcmp373("computer", "computer"); // returns 0
strcmp373("are", "area"); // returns -97, ( '\0'- ‘a’)
```

3. Write a function called `strcat373`. The function is passed two parameters, both of which are C strings. **You should use pointer syntax when writing this function** (in other words, you can use * and &, but not [ ]). The function should modify the first parameter (a char array, often called the *destination* array) so that it contains the concatenation of the two strings, and returns a pointer to the destination array. For example:

```
int main() {
  char str1[9] = "comp";
  char str2[ ] = "uter";
        strcat373(str1, str2);
  // prints computer
  printf("str1 contains %s\n, str1);
```

Note that `strcat373` is guaranteed to work properly only if str1's length is big enough to contain the concatenation of the two strings. In this case, "computer" takes up 9 bytes, and since `str1` is 9 bytes, the function should run properly since the array is just long enough to contain “computer” (plus ‘\0’). On the other hand,

```
:
char str1[] = "comp";  // only 5 bytes
char str2[] = "uter";
strcat373(str1, str2); // takes up 9 bytes and
                       // therefore overflows str1
```

Upon execution, a runtime error may occur, or (even worse) no runtime error will occur, but some other variable(s) in your program may be overwritten.

4. Write a function called `strchr373`. It is passed 2 parameters: a string and a `char` Here is the prototype for the function:

```
char *strchr373(char str[], char ch);
```

The function should return a pointer to the first instance of `ch` in `str`. For example:

```
int main {
   char s[ ] = "abcbc";
   printf("%s", strchr373(s, 'b'));   // prints bcbc
   printf("%s", strchr373(s, 'c');    // prints cbc
   printf("%d", strchr373(s, 'd');    // prints 0
```

5. Write a function called `strncpy373`. It is passed 3 parameters: 2 strings and the length of the destination array (the first parameter). The intent of including a third parameter is to prevent overflow of the destination array in case the source array (the second parameter) contains a string that is too long to be

stored in the destination. `strncpy373` returns a pointer to the destination array.

For example:

```
char s1[] = "comp";
char s2[] = "systems";
strncpy373(s1, s2, 4);
printf("%s\n", s1);
```

The output of the above code should be `syst`.