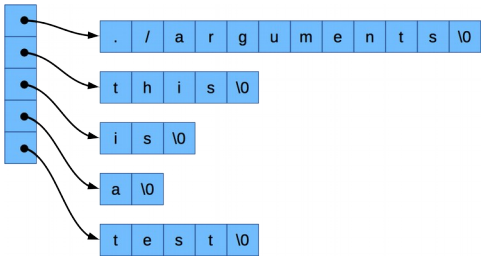
2D Representations and Command-Line Arguments

This exercise will give you a chance to work with strings, command line arguments and to compare two common ways of working with two-dimensional representations in C. On the course homepage, you'll find a partial implementation of a program called arguments.c, along with a couple of expected output files. You can also download these files with the following curl commands:

curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise12/arguments.c
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise12/expected-1.txt
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise12/expected-2.txt

In this exercise, we'll be copying the command-line arguments to a different representation and then printing them out, one argument per line. In practice, there's not really a good reason to copy the command-line arguments like this. We're just doing it to get some experience working with the different options we have for storing two-dimensional data in C.

The figure to the right shows how a C program gets its command-line arguments at startup. You get an integer, argc, telling you how many command-line arguments there are, and an array of char pointers, argv, with each element pointing to the start of one of the command-line arguments.



This array-of-pointers representation works a lot like a two-dimensional array. The expression argv[2] evaluates to a char pointer, pointing at the start of argument 2. Indexing from this pointer, an expression like argv[2][1] will give you an individual character (the character 's' in the figure above).

In C we also have the option of representing a list of strings as a real, two-dimensional array, where the elements of the array are organized in row-major order in memory, and every row has to be exactly the same length. For this exercise, you're going to dynamically allocate a representation like this, a 2D array with a row for every command-line argument, and with exactly enough columns to store the longest string from argv (along with the null terminator marking the end of the string).

.	/	a	r	g	u	m	e	n	t	s	\0
t	h	i	s	\0							
i	s	\0									
a	\0										
t	e	s	t	\0							

The partial implementation helps you with some of the more difficult parts, and shows you where you need to add code. Here's what you need to do:

- In main(), first, add code to figure out the length of the longest command-line argument. This will determine the width of the two-dimensional array you need to allocate.
- I've already created a variable, words, for you. This variable can serve as a pointer to a 2D array that's wide enough to hold a copy of the command-line arguments. Add code to dynamically allocate memory for this array, and make the words pointer point to it. Be sure your array is exactly the right size to hold a copy of the command-line arguments. It should have a row for every argument, and enough columns to hold the longest argument, along with it's null terminator.
- Add a loop to copy all the command-line arguments from argc into the two-dimensional array you just allocated. Each element of argv is a pointer to a string you need to copy, and each row of your words array is a string you need to copy one of the arguments to.
- In the printArguments() function, add code to loop over each of the strings in the args parameter that's passed to it, and print them out, one argument per line. The parameter to this function is a 2D array, but each row of that array should be a string, which you should be able to print out just like any other string (e.g., with printf).

When your program is working, you should be able to run it with any number of command-line arguments, of any (reasonable) size, and it should be able to copy them to a 2D array and print them out. To get the same

output as expected-1.txt, you should be able to run it like:

```
$ ./arguments this is a test
./arguments
this
is
a
test
```

The expected-2.txt output has more command-line arguments, and one that's a little bit longer:

```
$ ./arguments this is a command-line-argument-thats-46-characters-long but the rest
of them are shorter
./arguments
this
is
a
command-line-argument-thats-46-characters-long
but
the
rest
of
them
are
shorter
```

Once your program is working, submit it via the exercise_12 assignment in Moodle.