CS 214: Systems Programming, Spring 2017
Assignment 0: String Sorting

0. Introduction
In this assignment, you will practice programming with C pointers and using dynamic memory. Manipulation of dynamic memory and the pointer construct are two defining characteristics of the C language. They are powerful tools but can be fiddly to work with. For your first assignment, you will get a little practice working with both.

You will write a rather simple program to sort words given in a single input string. Your code will take a single, long input string that contains all words to be sorted, break them out into individual strings, sort them, and output them one per line in descending alphabetical order.

You will not know the length of the input string or its constituent strings beforehand, or how many there are, so you will not know how much memory to allocate or how many pointers you'll need to handle the input string. These are two fundamental issues you must consider.

1. Implementation
The input string will contain a series of component strings, e.g.: “thing stuff otherstuff blarp”. You must extract and sort these strings. The component strings will be separated by non-alphabetic characters. You can consider any non-alphabetic character a separator, e.g.: “thing1stuff3otherstuff,blarp” is equivalent to the input string above. Be careful not to run off of the end of the input string as you are scanning it. Your first task should be to determine its length.

You have three major segments to think about:
0. How to deal with component strings of unknown lengths
1. How to deal with an unknown number of component strings
2. How to sort an unknown number strings of an unknown length

<u>1.0 Unknown Lengths</u>
You can only ask for a given amount of memory to store some data in C. Since you do not know the lengths of the component strings when you start, you'll need to discover them so that you can request memory to store them in with malloc().

Any time you do not know exactly how long a process will take, but know when are you done, is often an excellent time for a loop. You should use pseudocode and rough things out first.

```
while( haven't fallen off the input string )
{
  .. read from input string
  while( haven't found a separator and haven't fallen off the input string )
  {
     .. read from input string
     .. advance to next character
  }
  // either fell off of input string, or found a separator
```

```
if( found a separator )
{
  .. copy the component string out of the input string
}
```

You will need to move along the input string testing characters until you find a separator. Remember, any non-alphabetic character is a separator (can't just split()).

Once you find a separator, you can determine the length of the string it marked the end of. You can then request some memory to store that string with malloc() and copy the string over with memcpy(). You now need to hold the pointers to your component string copies in a data structure.

1.1 Unknown Number

Since you do not know how many total component strings will be in a single input string, you need to be ready with an extensible data structure. There are several ways to approach this problem. You could build a link list, or manage an array to increase its size as needed, or implement a red/black tree. However you solve it, the data structure you use to hold your component string pointers must stretch to hold any number of them.

1.2 Sorting

Since you must discover the end of a component string before you copy it out of the input string and into your list of strings, you must insert them into your list one by one. You might find it easier to sort them and insert them into the list in sorted order. If not, you can easily build the list, but will need to sort all of it at once.

When writing your sorting function, remember that it will not know the lengths of the strings it is comparing. Be careful not to fall off the ends of your component strings when comparing them.

2. Organization

You may use any functions or libraries available in the iLab machines. You will likely want to investigate the functions defined in the string.h library. Keep in mind that coding style will affect your grade. Your code should be well-organized, well-commented, and designed in a modular fashion. In particular, you should design reusable functions and structures and minimize code duplication. You should always check for errors. For example, you should always check that your program was invoked with the correct number of arguments. Your code should compile correctly (no warnings and errors) with the -Wall and either the -g or -O flags, e.g.:$ gcc -Wall -g -o pointersorter pointersorter.c should compile your code to a debug-able executable named tokenizer without producing any warnings or error messages (note that -O and -o are different flags).

Be sure to also document your code, describe how it operates and test it. A significant portion of your grade is testing your code. You should also expect, and test for, unfriendly input. For instance, running your code with too few arguments, with too many arguments, with blank strings, with two separator characters together, etc. Your code should handle these situations gracefully. If you decide you must exit, exit nicely with an informative error message. Be sure to free any and all dynamic memory before returning or exiting.

3. Submission Requirements

Your submission MUST (un)tar (see below), compile and execute on the iLab machines or a zero grade will be given. Be sure to test your code on an iLab machine before handing it in. If it works on your laptop, but does not compile on the iLab machines, it will still net a zero.

Turn in a tarred, gzipped file named "Asst0.tgz" that contains a directory called Asst0 with the following files in it:

- A pointersorter.c file containing all of your code.
- A file called testcases.txt that contains a thorough set of test cases for your code, including inputs and expected outputs.
- A readme.pdf file that contains a brief description of the program and any great features you want us to notice.

Creating the tar file:
First create a directory named "Asst0" and copy your three files in to it.
Run "tar cfz Asst0.tgz Asst0"
This should create a new file named "Asst0.tgz" that contains your directory and your code.

Your grade will be based on:
Correctness (how well your code works).
Quality of your design (did you use reasonable algorithms).
Quality of your code (how well written your code is, including modularity and comments).
Testing thoroughness (quality of your test cases).

3. Examples

./pointersorter "thing stuff otherstuff blarp"
blarp
otherstuff
stuff
thing

./pointersorter "thing1stuff3otherstuff,blarp"
blarp
otherstuff
stuff
thing

4. Additional

Be careful to output your strings in descending alphabetical order
Treat capital letters as 'greater than' lower case – they should come before lowercase:
for words: aand, aAnd, Aand, Aand
output as:
AAnd
Aand
aAnd
aand

Be robust to error!
- expect the wrong number of inputs (return and print out a usage string)
- expect only non-letter inputs (print out nothing)
- expect empty inputs (print out nothing)
- expect terrible, terrible things