## String Sorter

---

Hetal Patel - hp373
Rushabh Jhaveri - rrj28
Systems Programming, Spring 2018

---

**Synopsis:**

Stringsorter is an assignment that consists of two main files:

1. stringsorter.c
2. stringsorter.h

stringsorter takes in 1 argument: a string (enclosed within double quotes).

Stringsorter then takes the string, breaks down the string into the distinct words in the string based on the given delimiters (all non-alphabetic characters), and sorts each word in alphabetical order where uppercase letters have higher precedence.

**Design and Working**

---

This program is designed to take in a single string argument. The program first checks whether the number of arguments passed to the program is correct. If the number of arguments passed to the program is not equal to 2, the program prints an error message and explains how many arguments are supposed to be passed to the program, and then exits the program.

The program then checks whether the string passed as argument is an empty string. If the string passed is an empty string, the program prints out nothing, and then exits the program.

The program then passes the string to the function:

```
char **processAndBuildStrArr(char* str, int *retArrLen)
```

This function processes and builds the words contained within the single long string. This function dynamically allocates space for the data structure used to build and store each word (data structure used: array). When more memory is required, the array size is doubled using

**realloc().** Checks have been included to take care of all memory allocation and reallocation failures - an error message is printed and the program exits. This resizing of the array occurs when the number of strings is 2 less than the size/length of the array. After storing each word in the array, it returns the array containing each distinct word in the string. This array is then passed to the function:

```
void sort(char **m, int dim)
```

This is where the array gets sorted using selection sort, and the sorted array is returned. This function calls another function:

```
void swap(char **s1, char **s2)
```

This function performs the in-array swap that is characteristic in the selection sort algorithm.

The sorted array is then printed as the final output.

**Challenges Faced in the Making of this Program**

---

1. Deciding what data structure would be best.
2. Getting accustomed to when to use malloc and realloc, and how to use them correctly.
3. Dealing with segmentation faults.

**Running Time Analysis**

---

This program runs in has a worst running time of $O(n^2)$ because the worst case running time of selection sort is of second order.