CSCI 4210 — Operating Systems
CSCI 6140 — Computer Operating Systems
Homework 1 (document version 1.0)
Strings and Memory Allocation in C

Overview

- This homework is due by 11:59:59 PM on Thursday, September 15, 2016. Homeworks are to be submitted electronically.
- This homework will count as 9% of your final course grade (unless the syllabus changes in terms of the number of homeworks assigned for the semester).
- This homework is to be completed **individually**. Do not share your code with anyone else.
- You **must** use C for this homework assignment.
- Your program **must** successfully compile and run on Ubuntu v14.04.5 LTS or higher.
- Your program **must** successfully compile via `gcc` with absolutely no warning messages when the `-Wall` (i.e., warn all) compiler option is used. We will also use `-Werror`, which will treat all warnings as critical errors.

Homework Specifications

In this first homework, you will use C to implement a relatively simple text file manipulation program. The goal is to become more comfortable programming in C on Linux, in particular handling strings and dynamically allocating (and re-allocating) memory. Overall, your program will open and read a text file, parse all words from the file, store each parsed word into a dynamically allocated array, then display words that contain a specific substring. To open a file, consider using `fopen()`.

Words are defined as containing one or more alphanumeric characters. There are many ways to read input from a file using C. Consider using `fscanf()`, `fgetc()`, `isalnum()`, and other such string and character functions. Check out the details of each function by reviewing the corresponding `man` pages.

The file to read is specified on the command-line as the first argument, while the match-substring is the second command-line argument. As an example, you could execute your program to read `filename.txt` and find all words containing substring `hi` as follows:

```
bash$ ./a.out filename.txt hi
```

If improper command-line arguments are given, report an error message to `stderr` and abort further program execution.

Key requirements are:

- You must store all words in memory, even those words that do not match the given substring. Further, do not perform any deduplication of words.
- You must dynamically allocate memory to store these words. To do so, dynamically create an array of character pointers (`char*`) using `calloc()`. Start with an array size of `8`. If the size of the array needs to be increased, use `realloc()` to do so, doubling the size each time.
- You must also dynamically allocate the memory to store each word. To do so, use `malloc()` or `calloc()`; be sure to calculate the number of bytes to allocate as the length of the word plus one, since strings in C are implemented as `char` arrays that end with a `'\0'` character.

Required Output

When you execute your program, you must display a line of output each time you allocate or re-allocate memory for the array.

As an example, if you are given the following `filename.txt` file:

```
Once when a Lion was asleep a little Mouse began running up and down upon him;
this soon wakened the Lion, who placed his huge paw upon him, and opened his
big jaws to swallow him. "Pardon, O King," cried the little Mouse: "forgive me
this time, I shall never forget it: who knows but what I may be able to do you
a turn some of these days?" The Lion was so tickled at the idea of the Mouse
being able to help him, that he lifted up his paw and let him go. Some time
after the Lion was caught in a trap, and the hunters, who desired to carry him
alive to the King, tied him to a tree while they went in search of a wagon
to carry him on. Just then the little Mouse happened to pass by, and seeing
the sad plight in which the Lion was, sent up to him and soon gnawed away the
ropes that bound the King of the Beasts. "Was I not right?" said the little Mouse.
         "LITTLE FRIENDS MAY PROVE GREAT FRIENDS."
```

Executing the code as follows should then display the output shown below:

```
bash$ ./a.out filename.txt hi
Allocated initial array of 8 character pointers.
Re-allocated array of 16 character pointers.
Re-allocated array of 32 character pointers.
Re-allocated array of 64 character pointers.
Re-allocated array of 128 character pointers.
Re-allocated array of 256 character pointers.
All done (successfully read 184 words).
Words containing substring "hi" are:
him
this
his
```

```
him
his
him
this
him
his
him
him
him
while
him
which
him
```

Match the above output format **exactly as shown above**.

Think about how large of an input file you would need (i.e., how many words) to cause your program to crash due to insufficient memory. Try to figure this out through trial and error (i.e., repeated program executions with increasingly larger datasets).

Answers will vary here, but add a comment at the top of your code specifying the number of words that caused your program to run out of memory and crash. Format the comment as follows:

```
/* Number of words that caused out-of-memory case: ##### */
```

Finally, be sure to use `free()` to ensure all allocated memory is properly deallocated.

Submission Instructions

To submit your assignment (and also perform final testing of your code), please use Submitty, the homework submission server. The URL is on the course website.