CSCI 4210 — Operating Systems
CSCI 6140 — Computer Operating Systems
Homework 1 (document version 1.2)
C Programming

Overview

- This homework is due by 11:59:59 PM on Thursday, September 10, 2015. Homeworks are to be submitted electronically.
- This homework will count as 4% of your final course grade.
- 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 v14.04.3 LTS or v15.04.
- Your program **must** successfully compile via `gcc` with no warning messages using the `-Wall` compiler option.

Homework Specifications

In this first homework, you will use C to implement a simple text file manipulation program. The goal is to become more comfortable programming in C on Linux, in particular handling strings and dynamically allocating memory.

Overall, your program will open and read a text file, parse the words from the file, store each parsed word into an array, then display all words that have a specific prefix. To open a file, you could use `fopen()`.

Words are defined to contain one or more non-whitespace characters. Therefore, you can use `fscanf()` to read words into a string using the default whitespace delimiters (i.e., `' '`, `'\t'`, `'\n'`, and a few others shown in the `man` page for `isspace()`).

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

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

Key requirements are:

- You must store all words in memory, even those words that do not match the given prefix. 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 `20`. 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()` (and 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 debug line of output each time you allocate or reallocate memory for the array.

Given the following example `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 20 character pointers.
Reallocated array of 40 character pointers.
Reallocated array of 80 character pointers.
Reallocated array of 160 character pointers.
Reallocated array of 320 character pointers.
him;
his
him,
his
him.
him,
his
him
him
him
him
him
```

Think about how large of an input file you 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).

Submission Instructions

To submit your homework, please create a single compressed and zipped file (using `tar` and `gzip`). Please include only source and documentation files (i.e., do not include executables or binary files!).

To package up your submission, use `tar` and `gzip` to create a compressed `tar` file using your RCS userid, as in `goldsd.tar.gz`, that contains your source files (e.g., `main.c` and `file2.c`); include a `readme.txt` file only if necessary.

Here's an example showing how to create this file:

```
bash$ tar cvf goldsd.tar main.c file2.c readme.txt
main.c
file2.c
readme.txt
bash$ gzip -9 goldsd.tar
```

Submit the resulting `goldsd.tar.gz` file via the corresponding homework submission link available in LMS (`http://lms9.rpi.edu`). The link is in the Assignments section.