Department of Electrical and Computer Engineering
ECSE 202 – Introduction to Software Development
Assignment 6
Stacks, B-Trees, Sorting Data in "C"

Due March 16, 2018 at 5:00 pm

Problem Description

This assignment is similar to Assignment 2, with the exception that you only have to print the list of names in alphabetical order. Although "C" is not an object-oriented programming language by design, it is possible to apply many of the same concepts you have learned in Java towards creating and manipulating "objects" in "C". The key concepts are user-defined data types (structs) and dynamic memory allocation (malloc) that we will have discussed in class.

Normally in taking the object-oriented approach, one would create a pair of files for each "class" – a .h file with structure definitions corresponding to the objects created, and a .c file with the corresponding methods that operate on these objects. For this assignment we'll keep things simple and produce a single file for the entire assignment.

Your program should operate as follows:

1. Opens a file for reading.
2. Iterates through the file, line by line as shown in the example, calling the addNode function to store the current entry.
3. On completion, calls the inorder function to print the tree in sort order.

Your functions should mirror the methods you used in Assignment 2. There is no analog to Java's receiver syntax in "C". For instance, if myTree is a pointer to your bTree data structure, you do not invoke the addNode function using myTree.addNode(line). In "C" all methods are static, so the pointer to the tree must be passed as a function argument.

For example, the declaration for the addnode function might look like this:

```
struct bNode *addnode(struct bNode *root, char *name);
```

Your program should operate EXACTLY like the one you wrote for Assignment 2, except that this assignment will use the CLI instead of interactive console.

```
% sortFile
Usage: sortFile [text file name]
% sortFile foo
Error: Can't read from file foo
%sortFile Names-short.txt
```

(next page)

```
Assignment 6 - File Sorting Program
Enter name of file to sort: Names-short.txt


File in sort order:

Benes Dorethea
Britto Teisha
Freeze Clarisa
Galentine Dante
Woolery Ciera


File in reverse sort order:

Woolery Ciera
Galentine Dante
Freeze Clarisa
Britto Teisha
Benes Dorethea


Program terminated
```

As before, we have included a listFile program that takes care of dealing with "C" files – you can use this as the base for your main program.

```
/*

======================================================================
 Name        : listFile.c
 Author      : F. Ferrie
 Version     :
 Copyright   : Your copyright notice
 Description : A "C" version of the fileReader program originally
             : written Java for Assignment 2.  This one uses the
command
             : line interface to obtain the file name.  Like its
predecessor,
             : this program reads a text file line by line, displaying
the
             : output on the standard output.  Use this program as a
basis
             : for Assignment 6.

======================================================================
 */
```

```c
#include <stdio.h>
#include <stdlib.h>
#define MAXBUF 100

int main(int argc, char *argv[]) {
// Internal declarations
    FILE * FileD;           // File descriptor (an object)!
    char *line;             // Pointer to buffer used by getline
    int bufsize = MAXBUF;   // Size of buffer to allocate
    int linelen;            // Length of string returned (getline)

// Argument check
    if (argc != 2) {
     printf("Usage: fileReader [text file name]\n");
     return -1;
    }

// Attempt to open the user-specified file.  If no file with
// the supplied name is found, exit the program with an error
// message.

    if ((FileD=fopen(argv[1],"r"))==NULL) {
     printf("Can't read from file %s\n",argv[1]);
     return -2;
    }

// Allocate a buffer for the getline function to return data to.

    line=(char *)malloc(bufsize+1);
    if (line==NULL) {
     printf("Unable to allocate a buffer for reading.\n");
     return -3;
    }

// If the file exists and a buffer was successfully allocated,
// use the getline function to read the file libe by line.  This
// is directly analogous to the readLine method in Java.
//
// Notice that the first argument to getline is a pointer to
// a pointer.  This type of argument passing is used when
// the function modifies that actual value of the pointer itself
// as well as the data that the pointer references.  This detail
// is beyond the scope of this course.

// Another detail about the behavior of the getline function -
// it returns the \n (newline) delimiter at the end of the
// current line, so you need to remember to strip this off
// depending on how you use the string.  Since this function
// simply prints the file, the newline is left in and does not
// have to be added in the printf call.

    printf("Listing of file %s:\n",argv[1]);

    while ((linelen=getline(&line,(size_t *)&bufsize,FileD))>0)
     printf("%s",line);
    printf("End of file\n");

    return EXIT_SUCCESS;
}
```

Strategy

Since you already have a working java program that implements this program, you can use this as a template for the present assignment. Essentially, you need to design an approprate bNode struct to hold your data, and the corresponding addNode and inorder functions that operate on the data structure. These functions look a lot like their corresponding methods except for how nodes are allocated.

Once you have the bTree "class" implemented, it should be easy to modify the listFile program and create sortFile as required.

Structure

Your program can be submitted as a single file named sortFile.c

Instructions

Test your program using the Names.txt file. When inputting the file name, you must specify the complete name including path, e.g., \Users\ferrie\assignments\A2\Names.txt, unless the file resides in the *default directory*. If your project lives at \Users\ferrie\workspace\A2, then if you copy the Names.txt file to that location, then it suffices to type *Names.txt* when prompted for the file name.

If your program works correctly, it should produce a display similar to that shown on Page 2, with the exception of printing all the entries in Names.txt. Save this output by copying and pasting into a text file, e.g. into Notepad if you are using Windows, for example. Save this file as sortFile.txt.

About Coding Assignments

We encourage students to work together and exchange ideas. However, when it comes to finally sitting down to write your code, this must be done *independently*. Detecting software plagiarism is pretty much automated these days with systems such as MOSS.

https://www.quora.com/How-does-MOSS-Measure-Of-Software-Similarity-Stanford-detect-plagiarism

Please make sure your work is your own. If you are having trouble, the Faculty provides a free tutoring service to help you along. You can also contact the course instructor or the tutor during office hours. There are also numerous online resources – Google is your friend. The point isn't simply to get the assignment out of the way, but to actually learn something in doing.

fpf/April 2, 2018