Department of Electrical and Computer Engineering
ECSE 202 – Introduction to Software Development
Assignment 3
BTrees, Dynamic Memory Allocation

Problem Description

In this assignment you are to design a program that behaves identically to the one you wrote for Assignment 2. However, the implementation must incorporate the following changes:

1. Instead of storing student records as an array of structs that has to be allocated at compile time, records will now be stored in a Binary Tree (B-Tree). The `addnode` example in the notes provides a method for constructing the tree as you read in data from the two files.
2. Once the data are stored as a B-Tree, sorting can be done very efficiently by traversing the tree IN ORDER (left, root, right). Have a look at the ptree example to see how this can be accomplished. Instead of printing the in order nodes, you need to save the corresponding pointer in an array (i.e. `pSRecords`). Note that you will need to allocate this array dynamically as well.
3. The user interface should be case *insensitive*, in other words *cranford, Cranford, and CRANFORD*, should be lexicographically equivalent.

The rest of the program can remain largely unchanged, but you will have to pay attention to how the data are read in. In the original A2Template.c, the *NamesIDs* and *marks* files were opened and read in sequence, requiring 2 passes through the `SRecords` array. This is a bit more difficult to accomplish with a B-Tree, so it is suggested that you open both files and read from *NamesIDs* and *marks* at the same time. In this way you can assemble a complete record from the input, create a new node and copy over the current data.

Designing the Program

1. Replacing the SRecords array with a B-Tree

The class notes provide sufficiently detailed examples on how to design a node struct that will support a B-Tree, and an example function for creating and linking nodes. You should notice some similarity to the `QuickSort` program from Assignment 2, which should tell you how to structure the `addnode` function. Also notice that the input to `addnode` now corresponds to a struct.

As mentioned above, you will need to re-arrange A2Template.c to open both *NamesIDs* and *marks* at the same time so that all of the data corresponding to a particular student can be assembled at the same time, i.e., read LastName, FirstName, ID and marks into a `StudentRecord` struct, and then pass that struct to *addnode*. This should not be too difficult to figure out.

2. Sorting

Once you have completed building the database, you might want to verify by printing out the student records in sort order. The `ptree` example in the notes should be easy to modify for this purpose. Observe that the only difference between a sort routine and an In Order traversal of the B-Tree (which is exactly what `ptree` does), is that a sort routine would need to "remember" the order in which nodes were printed, e.g. by saving the results to an array of pointers.

There are a number of things to consider here. Since we are not allowed to allocate the pointer array ahead of time, we need to

```
malloc(numrecords*sizeof(struct StudentRecord **));
```

and cast to an array of pointers to `StudentRecord`. Once you have created a pointer array, this can be passed to the In Order traversal routine where the print would be replaced by assigning the current record pointer to the pointer array. Important detail – as you are traversing the B-Tree, you will need to increment a counter to keep track of the number of records visited. Since this routine is *recursive*, you cannot simply declare a local variable, and do something like `pointer_array[i++]=root`. This is because each subsequent call will re-initialize `i` back to 0! Think about this carefully!

If all goes well, your sort routine will return a pointer array that looks exactly like `pSRecords` in Assignment 2. The rest of your code should be unchanged.

3. User Interface

The user interface (command line input) is identical to Assignment 2. You might take this opportunity to make sure that your program can cope with incorrect user input. Note that the `strcmp` function is case sensitive. The obvious fix would be to replace the call to `strcmp` with a call to an equivalent function that is case insensitive. Fortunately, one is available to you in the standard "C" library.

Finishing Up

Repeat the same sequence of tests used in Assignment 2, but altered so as to demonstrate the case insensitivity of the new program.

```
ferrie@FastCat{Assignment 2}: Tlookup NamesIDs.txt marks.txt tait
The following record was found:
Name: Suzi Tait
Student ID: 2519
Student Grade: 82
ferrie@FastCat{Assignment 2}: Tlookup NamesIDs.txt marks.txt MUSHRUSH
The following record was found:
Name: Felicita Mushrush
Student ID: 2825
Student Grade: 76
ferrie@FastCat{Assignment 2}: Tlookup NamesIDs.txt marks.txt DAGGett
The following record was found:
```

```
Name: Yuonne Daggett
Student ID: 2029
Student Grade: 74
ferrie@FastCat{Assignment 2}: Tlookup NamesIDs.txt marks.txt spece
The following record was found:
Name: Matilde Spece
Student ID: 2917
Student Grade: 68
ferrie@FastCat{Assignment 2}: Tlookup NamesIDs.txt marks.txt aYlor
The following record was found:
Name: Francene Aylor
Student ID: 2728
Student Grade: 72
ferrie@FastCat{Assignment 2}: Tlookup NamesIDs.txt marks.txt foo
No record found for student with last name foo.
```

Instructions

Write a "C" command line program, Tlookup, as shown in the above examples. It should be able to validate the correct number of arguments and fail gracefully if the specified files are not found. A correctly working program should be able to replicate the examples shown above *exactly*.

To obtain full marks, your program must work correctly, implement all data structures specified, and be reasonably well documented.

As with Assignment 2, your "C" source file should be named Tlookup.c and include your name and student ID.

Run the examples shown and save your output to a file called Tlookup.txt. Again, make sure that this file contains your name and student ID.

Upload your files to myCourses as indicated.

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/Oct. 7, 2017

Original version of this assignment written by Prasun Lala, Fall 2016.