Department of Electrical and Computer Engineering
ECSE 202 – Introduction to Software Development
Assignment 6
A Simple Database Program

Due December 2nd at 5:00 pm

Problem Description

The attached file, dbReader.c, contains most of the code for a simple program that builds a database of student records from data stored in two input files: NamesIDs.txt and marks.txt. Once the database is built, a simple command interpreter is started that responds to the following list of commands:

```
LN List all the records in the database ordered by last name.
LI List all the records in the database ordered by student ID.
FN Prompts for a name and lists the record of the student with the
corresponding name.
FI Prompts for a name and lists the record of the student with the
Corresponding ID.
HELP Prints this list.
? Prints this list.
Q Exits the program.
```

The input is case insensitive, e.g., `ln, lN, and Ln` will all be interpreted as LN.

Examples

This program needs to be run from the command line (CMR, Terminal, Bourne shell, etc.). Assume that the program resides in your eclipse workspace:

cd c:\Users\ferrie\eclipse\A6\Debug
C:\Users\ferrie\A6\Debug>

Before running the program you need to copy NamesIDs.txt and marks.txt into this directory! To check, do a direectory listing:

C:\Users\ferrie\A6\Debug>dir

Directory of C:\Users\ferrie\A6\Debug>

2019-11-15 03:23 PM 1,064 NamesIDs.txt
2019-11-16 02:14 PM 392 sources.mk
2019-11-16 02:14 PM 1,009 makefile
2019-11-16 02:14 PM 14,444 A6-Fall-2019-Dev
2019-11-15 12:55 PM 231 objects.mk
2019-11-15 03:23 PM 150 marks.txt
2019-11-16 02:14 PM <DIR> src

6 File(s) 17,290 bytes
1 Dir(s) 261,675,446,272 bytes free

To start the program:

C:\Users\ferrie\A6\Debug>A6-Fall-2019-Dev NamesIDs.txt marks.txt
Building database…
Finished…

sdb:

sdb: LN

Student Record Database sorted by Last Name

Dorethea Benes	2583	97
Teisha Britto	2871	68
Cristobal Butcher	2969	77
Billy Ennals	2191	82
Clarisa Freeze	2135	70
Dante Galentine	1194	89
Nia Stutes	2872	97
Suzi Tait	2519	82
Ciera Woolery	1531	81

sdb: LI

Student Record Database sorted by Student ID

Dante Galentine	1194	89
Ciera Woolery	1531	81
Clarisa Freeze	2135	70
Billy Ennals	2191	82
Suzi Tait	2519	82
Dorethea Benes	2583	97
Teisha Britto	2871	68
Nia Stutes	2872	97
Cristobal Butcher	2969	77

sdb: FN
Enter name to search: Freeze

Student Name: Clarisa Freeze
Student ID: 2135
Total Grade: 70

```
sdb: FI
Enter ID to search: 2519

Student Name:     Suzi Tait
Student ID:       2519
Total Grade:      82

sdb: foo
Command not understood.

sdb: FN
Enter name to search: Fubar
There is no student with that name.

sdb: FI
Enter ID to search: 0
There is no student with that ID.

sdb: QUIT
Program terminated…
```

Approach:

To get the program to function, you need to implement the following functions:

```
bNode *addNode_Name(bNode *root, SRecord *Record);
bNode *addNode_ID(bNode *root, SRecord *Record);
bNode *makeNode(SRecord *data);
void inorder(bNode *root);
void search_Name(bNode *root, char *data);
void search_ID(bNode *root, int ID);
```

addNode_Name and addNode_ID are almost identical, the difference being in which quantity is used to order the B-Tree. The same holds for search_Name and search_ID which implement a binary search on the B-Tree. The remaining functions, inorder and makeNode, are identical in function to the Java code. From the class notes and your prior experience with Java, you should be able to create the necessary B-Tree functions. It is strongly suggested that you code and test these separately. Once you have this worked out, you can add them to the main code and test each of the supported functions.

The key difficulty in this assignment is understanding the function arguments and returns. The key data structure is SRecord, which is the structure that holds the student record data. Each time a new entry is read from the files, a new SRecord object is created and populated with data. Looking more closely at the code, the SRecord object, Record, is a pointer to the object allocated – exactly the same as is done in Java. Comparing the addNode functions to their Java counterparts, the root of the B-Tree is passed as an argument in the “C” version whereas it was

an instance variable (global) in the Java code. In "C" it's usually advisable to avoid global variables is it makes the code less portable and error prone. You will notice that addNode returns the root node. When the B-Tree is empty, the first node allocated becomes the root node and is returned to the main program. In all other instances, it simply returns the same value that it is called with. In the Java makeNode, the single argument corresponds to the object being added to the tree (aBall); in the "C" version this corresponds to the second argument which is a pointer to SRecord, the structure holding the current record being added. Aside from the function arguments and "C" pointer notation (the use of -> in place or "."), the "C" code is largely unchanged from the Java version. The makeNode function is also similar (-> in place of ".") and the use of malloc instead of new; inOrder is similarly straightforward - but must format the data as shown in the examples (hint: look at the printf statements in the FI and FN commands).

For this assignment you are to implement the *non-recursive* version of makeNode.

What is new here are the two search functions which *must* be implemented as binary search. This ends up looking very similar to addNode with the exception that the matching node is returned instead of a new one added. A few minutes with Google should provide any missing details. There is one remaining detail, how to return a value from inside a recursion. In Java we used an instance variable, essentially a global variable accessible from any instance of the recursion. To avoid complications with pointer-pointers (which we will mostly avoid in this course), we define a static variable bNode *match (which is globally accessible like a Java instance variable) which is used to return a pointer to the matching record.

It is worth doing this assignment carefully, especially if this is your first time developing software. This covers most of the key topics in Part II and is good preparation for your final exam.

Instructions:

Starting off with dbReader.c, modify this program to implement the full simple database program. The dbReader.c file will compile (with warnings) and run, simply listing the database and starting the command interpreter – with help and quit functional. Remember to make searches case insensitive.

To obtain full marks, your program must work correctly, avoid the use of arrays except for representing character strings, and be reasonably well commented.

Run the examples shown above and save your output to a file called database.txt. Place all of your source code in a single file, database.c

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/November 16, 2019.