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.

```
/*
 ============================================================================
 Name        : database.c
 Author      : F. Ferrie
 Version     :
 Copyright   : Your copyright notice
 Description : A simple program to manage a small database of student
             : records using B-Trees for storage.
 ============================================================================
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

// To make the "C" implementation completely analogous to Java, one has to create
// an object for each student record and attach it to a corresponding bNode
// object in a B-Tree data structure.  These objects are represented by the
// corresponding structure templates below.

#define MAXLEN 20
#define false 0
#define true !false
#define NR                      // If defined, non-recursive search is used

// Prototypes and templates should go in a file called sortFile.h which
// is subsequently included in sortFile.c.  For a small program like this one,
// a single file will do.

// Structure Templates

typedef struct SR {             // The student record object
    char Last[MAXLEN];
    char First[MAXLEN];
    int ID;
    int marks;
} SRecord;
```

```
typedef struct bN {             // The bNode object
    struct SR *Srec;
    struct bN *left;
    struct bN *right;
} bNode;

// Function Prototypes
//
// Notice that there are two versions of add_node, inorder, and search.
// This is to accommodate sorting the database by last name OR student ID

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);
void str2upper(char *string);
void help();

//
// Since we haven't done pointer-pointers in this course, we'll use a
// global variable to return the matching student record.  This is
// equivalent to what we did in Java using an instance variable.

bNode *match;


// Main entry point is here.  Program uses the standard Command Line Interface

int main(int argc, char *argv[]) {

// Internal declarations

    FILE * NAMESIDS;        // File descriptor (an object)!
    FILE * MARKS;           // Will have two files open
```

```
    bNode *root_N;          // Pointer to names B-Tree
    bNode *root_I;          // Pointer to IDs B-Tree
    SRecord *Record;        // Pointer to current record read in

    int NumRecords;
    char cmd[MAXLEN], sName[MAXLEN];
    int sID;

// Argument check
        if (argc != 3) {
                printf("Usage: sdb [Names+IDs] [marks] \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 ((NAMESIDS=fopen(argv[1],"r"))==NULL) {
                printf("Can't read from file %s\n",argv[1]);
                return -2;
        }

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

// Initialize B-Trees by creating the root pointers;

    root_N=NULL;
    root_I=NULL;

//  Read through the NamesIDs and marks files, record by record.

    NumRecords=0;
```

```
    printf("Building database...\n");

    while(true) {

//  Allocate an object to hold the current data

        Record = (SRecord *)malloc(sizeof(SRecord));
        if (Record == NULL) {
            printf("Failed to allocate object for data in main\n");
            return -1;
        }

//  Read in the data.  If the files are not the same length, the shortest one
//  terminates reading.

        int status = fscanf(NAMESIDS,"%s%s%d",Record->First,Record->Last,&Record->ID);
        if (status == EOF) break;
        status = fscanf(MARKS,"%d",&Record->marks);
        if (status == EOF) break;
        NumRecords++;

//  Add the current record to the B-Tree

        root_N=addNode_Name(root_N,Record);
        root_I=addNode_ID(root_I,Record);
    }

// Close files once we're done

    fclose(NAMESIDS);
    fclose(MARKS);

    printf("Finished, %d records found...\n",NumRecords);


//
//  Simple Command Interpreter:
```

```
//

    while (1) {
        printf("sdb> ");
        scanf("%s",cmd);                        // read command
        str2upper(cmd);

// List by Name

        if (strncmp(cmd,"LN",2)==0) {           // List all records sorted by name
            printf("Student Record Database sorted by Last Name\n\n");
            inorder(root_N);
            printf("\n");
        }

// List by ID

        else if (strncmp(cmd,"LI",2)==0) {      // List all records sorted by ID
            printf("Student Record Database sorted by Student ID\n\n");
            inorder(root_I);
            printf("\n");
        }

// Find record that matches Name

        else if (strncmp(cmd,"FN",2)==0) {      // List record that matches name
            printf("Enter name to search: ");
            scanf("%s",sName);
            match=NULL;
            search_Name(root_N,sName);
            if (match==NULL)
              printf("There is no student with that name.\n");
            else {
              if (strlen(match->Srec->First)+strlen(match->Srec->Last)>15) {
                printf("\nStudent Name:\t%s %s\n",match->Srec->First,match->Srec->Last);
              } else {
                printf("\nStudent Name:\t\t%s %s\n",match->Srec->First,match->Srec->Last);
              }
```

```
            printf("Student ID:\t\t%d\n",match->Srec->ID);
            printf("Total Grade:\t\t%d\n\n",match->Srec->marks);
          }
        }

// Find record that matches ID


        else if (strncmp(cmd,"FI",2)==0) {    // List record that matches ID
            printf("Enter ID to search: ");
            scanf("%d",&sID);
            match=NULL;
            search_ID(root_I,sID);
            if (match==NULL)
              printf("There is no student with that ID.\n");
            else {
              if (strlen(match->Srec->First)+strlen(match->Srec->Last)>15) {
                printf("\nStudent Name:\t%s %s\n",match->Srec->First,match->Srec->Last);
              } else {
                printf("\nStudent Name:\t\t%s %s\n",match->Srec->First,match->Srec->Last);
              }
            printf("Student ID:\t\t%d\n",match->Srec->ID);
            printf("Total Grade:\t\t%d\n\n",match->Srec->marks);
          }
        }

// Help

        else if (strncmp(cmd,"H",1)==0) {  // Help
            help();
        }

        else if (strncmp(cmd,"?",2)==0) {     // Help
            help();
        }

// Quit
```

```
        else if (strncmp(cmd,"Q",1)==0) {  // Help
            printf("Program terminated...\n");
            return 0;
        }

// Command not understood

        else {
            printf("Command not understood.\n");
        }
    }
}



//
// B-Tree functions used by the program
//

// addNode_Name - a new record to the B-Tree ordered by the name field
//
// This is the non-recursive version from the Java notes,
// ported to "C".
//


    bNode *addNode_Name(bNode *root, SRecord *data) {

        bNode *current;

    // Empty tree

        if (root == NULL) {
            root = makeNode(data);
        }

    // If not empty, descend to the leaf node according to
    // the input data.
```

```
    else {
        current = root;
        while (true) {

// Here we do a case insensitive comparison between the Last name
// field of the student record.  Notice the similarity to the
// Java version.

            if (strcasecmp(data->Last,current->Srec->Last)<0) {

// New data < data at node, branch left

                if (current->left == NULL) {            // leaf node
                    current->left = makeNode(data);     // attach new node here
                    break;
                }
                else {                                  // otherwise
                    current = current->left;            // keep traversing
                }
            }

            else {

// New data >= data at node, branch right

                if (current->right == NULL) {           // leaf node
                    current->right = makeNode(data);    // attach
                    break;
                }
                else {                                  // otherwise
                    current = current->right;           // keep traversing
                }
            }
        }
    }
    return root;
}
```

```
/**
* makeNode
*
* Creates a single instance of a bNode
*
* @param    SRecord *data - pointer to new data to be added
* @return   SRecord *node - new node returned
*/

    bNode *makeNode(SRecord *data) {
        bNode *node=(bNode *)malloc(sizeof(bNode));         // create new object
        node->Srec=data;                                    // link to data object
        node->left = NULL;                                  // set both successors
        node->right = NULL;                                 // to null
        return node;                                        // return handle to new object
    }


//
//  This is a clone of the addNode_Name function with the ordering
//  determined by the ID field.  This could have been done in a single
//  function by passing a 3rd argument specifying how to order.
//

    bNode *addNode_ID(bNode *root, SRecord *data) {

            bNode *current;

    // Empty tree

            if (root == NULL) {
                root = makeNode(data);
            }

    // If not empty, descend to the leaf node according to
    // the input data.
```

```
        else {
            current = root;
            while (true) {

// In this version of addNode, ordering is by student ID, so
// we don't need an external function to do the comparison.

                if (data->ID < current->Srec->ID) {

// New data < data at node, branch left

                    if (current->left == NULL) {            // leaf node
                        current->left = makeNode(data);     // attach new node here
                        break;
                    }
                    else {                                  // otherwise
                        current = current->left;            // keep traversing
                    }
                }

                else {

// New data >= data at node, branch right

                    if (current->right == NULL) {           // leaf node
                        current->right = makeNode(data);    // attach
                        break;
                    }
                    else {                                  // otherwise
                        current = current->right;           // keep traversing
                    }
                }
            }
        }
        return root;
    }
```

```
// InOrder - traverses B-Tree in sort order, copying pointers to each record.
//
// Note: the getline function does not strip off the \n from each string read
//       in from the file.  When printing, do not use the "%s\n" format as
//       this will skip a line in between each record.

void inorder(bNode *root) {
    if (root->left != NULL) inorder(root->left);

// Calculate the length of First + Last and use the appropriate number
// of tabs (\t) so that columns line up.
//
    if (strlen(root->Srec->First)+strlen(root->Srec->Last) < 15)
      printf("%s %s\t\t%d\t\t %d\n",root->Srec->First,root->Srec->Last,root->Srec->ID,root->Srec->marks);
    else
      printf("%s %s\t%d\t\t %d\n",root->Srec->First,root->Srec->Last,root->Srec->ID,root->Srec->marks);

    if (root->right != NULL) inorder(root->right);
}

//
//  Search Implementations.
//
//  if NR is defined, use the non-recursive versions of search_Name and search_ID,
//  otherwise use the recursive versions
//

#ifdef NR

//
//  Search for record by Lname using binary search.
//
//  This is a non-recursive version which follows the same
//  logic as addNode_Name.
//
```

```
void search_Name(bNode *root, char *data) {

    bNode *current;

// Empty tree

    if (root == NULL) {
        return;
    }

// If not empty, descend to the leaf node according to
// the input data.

    else {
        current = root;
        while (true) {

// Here we do a case-fold comparison using the strcasecmp()
// function.

            if (strcasecmp(data,current->Srec->Last)<0) {
                current = current->left;
                if (current == NULL) return;
            }

            else if (strcasecmp(data,current->Srec->Last)>0) {
                current = current->right;
                if (current == NULL) return;
            }

            else {
                match=current;
                return;
            }
        }
    }
}
```

```
//
//  Search for record by ID using binary search.
//
//  This is a non-recursive version which follows the same
//  logic as addNode_ID.
//

void search_ID(bNode *root, int data) {

    bNode *current;

// Empty tree

    if (root == NULL) {
        return;
    }

// If not empty, descend to the leaf node according to
// the input data.

    else {
        current = root;
        while (true) {

//  Here we do a simple comparison on the ID field data

            if (data < current->Srec->ID) {
                current = current->left;
                if (current == NULL) return;
            }

            else if (data > current->Srec->ID) {
                current = current->right;
                if (current == NULL) return;
            }

            else {
```

```
                match=current;
                return;
            }
        }
    }
}

#else

//
//  Search for record by Lname using binary search.
//
//  Since this routine is recursive, we need to return the matching
//  pointer to a variable external to this function.  Since we
//  haven't covered pointer-pointer in this course, we use the same
//  method as we used in Assignment 4 - a variable that is global
//  to main and this function.
//
//  match is declared outside of main making it a static variable.
//  It is analogous to using a Class Variable in Java.
//

void search_Name(bNode *root, char *data) {
    if (root != NULL) {

// Match
        if (strcasecmp(root->Srec->Last,data)==0) {
            match=root;
            return;
        }

// node < key -> search right
        else if (strcasecmp(root->Srec->Last,data)<0) {
            search_Name(root->right,data);
        }

// node > key -> search left
```

```
        else {
            search_Name(root->left,data);
        }
    }
}

//
//  Clone the search routine, except change the key comparison
//

void search_ID(bNode *root, int ID) {
    if (root != NULL) {

// Match
        if (root->Srec->ID == ID) {
            match=root;
            return;
        }

// node < key -> search right
        else if (root->Srec->ID < ID) {
            search_ID(root->right,ID);
        }

// node > key -> search left

        else {
            search_ID(root->left,ID);
        }
    }
}

#endif

//
//  Convert a string to upper case
//
```

```
void str2upper (char *string) {
    int i;
    for(i=0;i<strlen(string);i++)
       string[i]=toupper(string[i]);
     return;
}

// Help
// prints command list

void help() {
    printf("LN List all the records in the database ordered by last name.\n");
    printf("LI List all the records in the database ordered by student ID.\n");
    printf("FN Prompts for a name and lists the record of the student with the corresponding name.\n");
    printf("FI Prompts for a name and lists the record of the student with the Corresponding ID.\n");
    printf("HELP Prints this list.\n");
    printf("? Prints this list.\n");
    printf("Q Exits the program.\n\n");

    return;
}
```

```
Last login: Thu Dec  5 17:03:49 from 192.168.78.103
ferrie@Lizard{ferrie}: cd ~/eclipse-cdt-workspace/A6-Fall-2019-Demo
ferrie@Lizard{A6-Fall-2019-Demo}: ls
Debug   src
ferrie@Lizard{A6-Fall-2019-Demo}: cd Debug
ferrie@Lizard{Debug}: ls
A6-Fall-2019-Demo       A6-Fall-2019-Dev        NamesIDs.txt            makefile
marks.txt               objects.mk              sources.mk              src
ferrie@Lizard{Debug}: A6-Fall-2019-Demo NamesIDs.txt marks.txt
Building database...
Finished, 50 records found...
sdb> ln
Student Record Database sorted by Last Name

Renita Allbright        1716            81
Kortney Arner           1371            84
Francene Aylor          2728            72
Dorethea Benes          2583            97
Dara Boyette            1974            81
Teisha Britto           2871            68
Cristobal Butcher       2969            77
Shantelle Cassady       2079            94
Tasia Cranford          2274            73
Yuonne Daggett          2029            74
Caleb Denning           1257            71
Suzanna Duff            1583            91
Billy Ennals            2191            82
Edward Follett          1345            71
Loree Foor              1370            74
Clarisa Freeze          2135            70
Dante Galentine         1194            89
Marvella Greenlee       2582            82
Sarita Gutierez         1512            95
Eleanora Hamel          1266            78
Kamala Harbison         1645            92
Lavinia Jenkinson       2959            86
Kerstin Kohls           2130            92
Lucia Lally             1359            76
Vanda Lamarr            1456            82
Kristyn Lanoue          2101            72
Filomena Leasure        1946            79
Sammie Mackay           2330            90
Barton Mango            1198            68
Dewey Manzi             2563            77
Kori Marsch             2524            65
Alona Mclees            1097            99
Felicita Mushrush       2825            76
Mariann Novack          1549            79
Dianne Plaisted         2637            57
Matilda Poll            2729            75
Anika Pop               1452            76
Luba Quaranta           2790            75
Tamie Roza              2586            93
Joleen Saulnier         2356            81
Holley Sia              1497            74
Matilde Spece           2917            68
Nia Stutes              2872            97
Suzi Tait               2519            82
Adria Tamplin           2675            94
Hyon Thibodaux          2985            87
Yvette Wojtowicz        2667            59
Stasia Wolford          2193            68
Jay Woll                2986            76
Ciera Woolery           1531            81

sdb> li
Student Record Database sorted by Student ID
```

```
Alona Mclees            1097            99
Dante Galentine         1194            89
Barton Mango            1198            68
Caleb Denning           1257            71
Eleanora Hamel          1266            78
Edward Follett          1345            71
Lucia Lally             1359            76
Loree Foor              1370            74
Kortney Arner           1371            84
Anika Pop               1452            76
Vanda Lamarr            1456            82
Holley Sia              1497            74
Sarita Gutierez         1512            95
Ciera Woolery           1531            81
Mariann Novack          1549            79
Suzanna Duff            1583            91
Kamala Harbison         1645            92
Renita Allbright        1716            81
Filomena Leasure        1946            79
Dara Boyette            1974            81
Yuonne Daggett          2029            74
Shantelle Cassady       2079            94
Kristyn Lanoue          2101            72
Kerstin Kohls           2130            92
Clarisa Freeze          2135            70
Billy Ennals            2191            82
Stasia Wolford          2193            68
Tasia Cranford          2274            73
Sammie Mackay           2330            90
Joleen Saulnier         2356            81
Suzi Tait               2519            82
Kori Marsch             2524            65
Dewey Manzi             2563            77
Marvella Greenlee       2582            82
Dorethea Benes          2583            97
Tamie Roza              2586            93
Dianne Plaisted         2637            57
Yvette Wojtowicz        2667            59
Adria Tamplin           2675            94
Francene Aylor          2728            72
Matilda Poll            2729            75
Luba Quaranta           2790            75
Felicita Mushrush       2825            76
Teisha Britto           2871            68
Nia Stutes              2872            97
Matilde Spece           2917            68
Lavinia Jenkinson       2959            86
Cristobal Butcher       2969            77
Hyon Thibodaux          2985            87
Jay Woll                2986            76

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: 0
There is no student with that name.
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> q
Program terminated...
ferrie@Lizard{Debug}:
```