Department of Electrical and Computer Engineering
ECSE 202 – Introduction to Software Development
Assignment 2
Pointers, Sorting and Searching

Problem Description

In this assignment we explore the problem of organizing data (sorting) in order to facilitate looking up information (searching). This week's tutorials will introduce the concept of recursion and the quicksort algorithm, which we will use to organize a set of data records. They will also discuss search methods, notably binary search which will be applied to the sorted data to retrieve specific information.

To make this assignment tractable, you will be provided with template code for reading in a set of data files into an array of records used to hold the data. (In fact, one of the first things you should try is to use the template code to generate a short program for reading the two data files provided and printing out their contents on the standard output.) Your program should operate as follows:

```
%lookup NamesIDs.txt marks.txt Thompson

Student Record for ID 2667:
Name:  Thompson, Fred
ID:  2667
Grade: 83

%
```

The first argument, argv[1], corresponds to a text file containing names and student IDs, the second, argv[2], a text file containing student grades, and finally the third, argv[3], to the search key, which is a string corresponding to the Last Name of the student. In the above example, the record displayed corresponds to the student with Last Name = Thompson. To make search efficient, the data are first sorted according to student Last Name and then a binary search is performed to retrieve the specified record.

Designing the Program

Like our previous examples, this program follows the same sequence of steps:

1. Get data from user
2. Build the necessary data structures (in this case reading in a small database).
3. Transform the input (turn an index, the student Last Name, into a data record).
4. Display the results on the output stream

Step 1 follows from Assignment 1, here there are 4 tokens – the program name, a character string corresponding to a list of names, a second character string corresponding to the correspondent student grades, and finally a third character string corresponding to the Last Name which is used

as the index in the search. We intentionally skipped over the File I/O section in the notes, but have provided a template file, A2Template.c, that contains the code you will need to open the 2 data files and read the results into the following data structure.

```
struct StudentRecord {
     char FirstNames[MAXNAMELENGTH];
     char LastNames[MAXNAMELENGTH];
     int IDNums;
     int Marks;
};

struct StudentRecord SRecords[MAXRECORDS];
```

In order to perform a binary search, the array of student records must be sorted in order according to the search key, in this case the Last Name. Thus your first task is to code an implementation of the QuickSort algorithm described in the notes and discussed in the tutorials. This involves writing two functions, swap, which exchanges records given two indices, and qsort, that implements the QuickSort algorithm, e.g.,

```
void swap(int indx1,int indx2,struct StudentRecord SRecords[]);

void qsort(int left,int right,struct StudentRecord SRecords[]);
```

The example shown in the notes assumes that the input is an array of integers, which is not the case here. You'll need to modify the code to compare two strings (hint, the `strcmp()` function) and make a couple of other minor changes to make the algorithm work. Similarly a minor change will be needed for `swap` as well. Develop and test your QuickSort algorithm first before attempting to write the rest of the program.

Moving Records vs. Moving Pointers

This would be the most direct implementation, but not necessarily the most efficient one (but perhaps one to try first to make sure that your algorithm works correctly). The reason is as follows. Consider

```
struct StudentRecord temp;
temp=SRecords[1];
SRecords[2]=SRecords[1];
SRecords[1]=temp;
```

Each assignment results in an entire record being copied, which is inefficient, especially for large record structures. A better approach is to create an array of pointers as follows:

```
struct StudentRecord *pSRecords[MAXRECORDS];
struct StudentRecord  SRecords[MAXRECORDS];

for (i=0;i<numrecords;i++) {
```

```
  pSRecords[i]=&SRecords[i];
}
```

Instead of passing SRecords, we pass the array of pointers instead:

```
void swap(int indx1,int indx2,struct StudentRecord *SRecords[]);

void qsort(int left,int right,struct StudentRecord *SRecords[]);
```

To reference a particular record from the original SRecords array given the pointer array, we replace the "." operator with the "->" operator:

```
printf("The holder of record 10 is %s\n",
          SRecords[10]->LastNames);
```

At the end, the structure of SRecords is unchanged, but the pointers in pSRecords are rearranged such that pSRecords[0] points the first record in the sorted list and pSRecords[numrecords-1] to the last.

Finishing Up

Before moving on, you should write a program to read in the two data files, populate the SRecords array, print out the unsorted list of records, sort the array, and print out the list in sort order (by Last Name).

With the data structures in place and the data in sort order, the last step is to implement a binary search algorithm. This can be done with little difficulty following the example in the notes. As was done in Assignment 1, your program should be able to reproduce the results shown below:

```
ferrie@FastCat{Assignment 2}: lookup NamesIDs.txt marks.txt Tait
The following record was found:
Name: Suzi Tait
Student ID: 2519
Student Grade: 82
ferrie@FastCat{Assignment 2}: lookup NamesIDs.txt marks.txt Mushrush
The following record was found:
Name: Felicita Mushrush
Student ID: 2825
Student Grade: 76
ferrie@FastCat{Assignment 2}: lookup NamesIDs.txt marks.txt Daggett
The following record was found:
Name: Yuonne Daggett
Student ID: 2029
Student Grade: 74
ferrie@FastCat{Assignment 2}: lookup NamesIDs.txt marks.txt Spece
The following record was found:
Name: Matilde Spece
Student ID: 2917
Student Grade: 68
```

```
ferrie@FastCat{Assignment 2}: lookup NamesIDs.txt marks.txt Aylor
The following record was found:
Name: Francene Aylor
Student ID: 2728
Student Grade: 72
ferrie@FastCat{Assignment 2}: lookup NamesIDs.txt marks.txt foo
No record found for student with last name foo.
```

Instructions

Write a "C" command line program, lookup, as shown at the beginning of these assignment notes. 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, be written so as to minimize data movement (i.e. use pointers instead of copying structures), and be reasonably well commented.

As with Assignment 1, your "C" source file should be named lookup.c and include your name and student ID.

Run the examples shown and save your output to a file called lookup.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/Sept. 25, 2017

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