COSC 311, Project #5
Implementing a Simple Database Using Hashing
Due Date: December 6, 2022

In this project you will design and implement a small database system capable of the following operations:

1- Retrieving a record
2- Modifying a record
3- Adding a new record
4- Deleting a record

The database is composed of a random access file called "STUDENTS" containing student's academic records. The records in the STUDENTS file are of the form:

First name	Last name	Student ID	GPA
40	40	4	8

where the integers represent the field width in terms of the number of bytes. The student's first and last names are at most 20 characters (Unicode characters) long, the ID is an integer, and the GPA is of type double. There are actually two files in this project, a text file and a random access file. You will create a text file containing information for, at least, 10 students (one line per student), and this program will take that information and put it in a random access file. The information can be manipulated much faster when it is stored in the random access file. All of the operations defined below use the random access file.

Your database system will maintain an index (a HASH TABLE) for the ID field of the records in the database so it can perform the above operations more efficiently (i.e., in constant time). Use a hash table of size 37 and resolve collisions using separate chaining. Each chain must be implemented as a binary search tree (or an AVL-tree). Your hash table is an array of binary search trees (or AVL-trees), where the element stored at each tree node has two components: a KEY (student ID); and an ADDRESS (the record's position in the random access file). Write a menu driven program to implement this system. The menu should allow a user to select one of the following:

1- Make a random-access file
2- Display a random-access file
3- Build the index
4- Display the index
5- Retrieve a record
6- Modify a record
7- Add a new record
8- Delete a record
9- Quit

1- Make a random-access file: You have already implemented this in project #1.

2- Display the random-access file: You have already implemented this in project #1.

3- Build the index: It asks the user to enter the database name (a random-access file). It reads the database records sequentially one at a time, hashes the KEY (student ID), and inserts the pair (KEY, ADDRESS) in the hash table. The ADDRESS is the position of the record

containing the ID in the database (first record in the database is at position 0, second record is at position 1, and so on). Use the following hash function in your program:

hash (key) = ((key*key) >>> 10) % 37

(Note: Due to the use of lazy deletions, the key and the position of a deleted record should not be added to the hash table.)

4- Display the index: It asks the user for the starting and ending index values (default value for the starting index is 0 and for the ending index is 36), and displays the non-empty hash table entries that are within that range. If the user doesn't enter a value (just presses the return key) for either or both of the staring and ending values, the default values will be used instead. For each non-empty hash table entry, it prints the index value followed by the level-by-level traversal of the associated binary search tree (or AVL-tree) with each level printed on a separate line. The information that is printed for each node is the pair (KEY, ADDRESS).

5- Retrieve a record: It asks the user to enter a student ID (a key value). It uses the hash function to transform the student ID into the hash table index. It searches the binary search tree (or AVL-tree) referenced through that index for the key, and if the search is successful it uses the ADDRESS associated with that key to retrieve and then display the corresponding record. If the search is unsuccessful, it prints a message indicating the failure of the search.

6- Modify a record: It asks the user to enter a student ID (a key value). It uses the hash function to transform the student ID into the hash table index. It searches the binary search tree (or AVL-tree) referenced through that index for the key, and if the search is successful it uses the ADDRESS associated with that key to retrieve and then display the corresponding record. It allows the user to modify any fields of this record, except the student ID field, and then it writes the modified record over the original record in the database. Note that modifying a record doesn't require any changes to the index (the hash table) because the user is not allowed to change the ID field. If the search is unsuccessful, it prints a message indicating the failure of the search.

7- Add a new record: It asks the user to enter data for a new record. This new record will be appended to the end of the database (the random-access file). Next, the student ID (the key value) and the position (ADDRESS) of the record just written to the end of the database must be added to the index. This means, the student ID must be transformed into a hash table index and a new node containing (KEY, ADDRESS) must be inserted into the binary search tree (or AVL-tree) referenced through that index.

8- Delete a record: It asks the user to input the student ID (the key value) of the record that needs to be deleted. It uses the hash function to transform the student ID into the hash table index. It searches the binary search tree (or AVL-tree) referenced through that index for the key, and if the search is successful it uses the ADDRESS associated with that key to delete the corresponding record from the database (using lazy deletion). It also deletes the node containing the key value (i.e., (KEY, ADDRESS)) from the binary search tree (or AVL-tree) corresponding to that index.

9- Quit: It displays the entire hash table and then terminates the program. The hash table entries are printed as follows:

0- (KEY, ADDRESS) (KEY, ADDRESS) ……
1- (KEY, ADDRESS) ….
3- (KEY, ADDRESS) …..

.....

Note that the empty entries are not printed and also all the nodes for a given binary search tree is printed on the same line.

IMPLEMENTATION REQUIREMENTS:

- This project must have at least **SIX** classes: a student class, a pair class that represents a (key, address) pair, a queue class, a binary search tree (or an AVL-tree) class that includes an inner node class, a hash table class, and a class that has your driver program.
- The queue class and the binary search tree class must be **generic classes.**
- The program must be robust and not crash, no matter in what order the operations are performed. That means you must test for the existence of the random access file when you implement options 2-8, and also the existence of the index when implementing options 4-8.

OPERATIONAL SPECIFICATIONS: The program should always return to the main menu after processing the selected item. The program execution should only be halted if EXIT is selected from the main menu.

SUBMISSION: Please submit a zip file containing the following to the class website:

- source codes (well documented).
- an input file (create an input file containing at least 10 students)
- a text file containing a sample run that shows you have tested all the menu options
- compress your Eclipse project folder, including all the material listed above, and submit it to the class website (see below)

Important note: Test your program carefully prior to submitting it because you **can only submit once.** Compress your Eclipse project folder (the entire folder) containing all the source codes (all the classes), the input file, the sample run, and the random access file and then submit that zip file on the class website. Once your submitted file is unzipped, it should compile and run in Eclipse as is. **This project must be submitted on or prior to the due date in order to receive a grade for it. Otherwise, a grade of zero will be assigned to the project.**