COSC2430: Programming and Data Structures
Indexing of records using trees

Introduction

In this homework, you will create a C++ program that stores information about a university roster in an array list and uses indexes to access records quickly.

Each student record will be stored in the array list in order of student ID. This means that, when we need to retrieve a student record based on student ID, we can use binary search. But what happens when we need to look up a student by last name, or date of birth? We would have to sequential search through the array until we find a match. Unless we have indexes!

An index contains two data: the searched value (or key), and its position in the array. This way we can keep the index sorted by key and retrieve the position of the record in the array, without having to create duplicates of the records.

Input and Output

This program will need two input files to work: a file with the student records that will fill the array list, and a file with the key(s) that will need to be retrieved.

Each student record will include (separated by comma):

- Student ID (string)
- First name (string)
- Last name (string)
- Date of birth (date)
- Major (string)
- GPA (float)

Labels will be used in the key file to identify what field to search, followed by ":"

- ID
- FIRSTNAME
- LASTNAME
- DOB
- MAJOR
- GPA

Example of input file for student records

```
03985, Carita, Palencia, 1988-06-14, CS, 3.24
11122, Nannie, Usrey, 1995-10-12, Math, 2.65
… (more records)
```

Example of input file for keywords

LASTNAME: Palencia

The output file should include the full retrieved record (or records, in case of multiple matches/keys), followed by the number of accesses necessary to retrieve the record searching the array list and the one achieved using indexes.

If no record is found, print "No record found for <LABEL> <key>", followed by the accesses necessary to determine that the record was not in the database.

Example of output file

```
03985, Carita, Palencia, 1988-06-14, CS, 3.24
Without index: 22 accesses
With index: 4 accesses
```

Example of output file for missing record

```
No record found for LASTNAME Alvarez
Without index: 100 accesses
With index: 7 accesses
```

The main C++ program will become the executable to be tested by the TAs. The result should be written on another text file (output file), provided on the command line. The input and output files are specified in the command line, not inside the C++ code.

The general call to the executable (search_roster in this example) is as follows:

```
search_roster "A=<file>;B=<file>;C=<file>"
```

Call example with two input files and another output file.

```
search_roster "A=database.txt;B=keyword.txt;C=c.out"
```

Filling the array list with student records

Your first step in this homework will be reading the first input file and store its content in an **array list**. You can assume a maximum of 100 entries.

The array should store **objects** of an appositely designed **"student" class**. The class should include the following member attributes:

- Student ID (string)
- First name (string)
- Last name (string)

- Date of birth (date)
- Major (string)
- GPA (float)

The date of birth will be stored using another custom **class "date"** with 3 member attributes:

- Day (int)
- Month (int)
- Year (int)

Both classes should be complete with all the necessary method to create and manipulate the objects.

The array list must be sorted by student ID.

Creating indexes

After filling the array completely, you must create index trees for **each attribute** in the student record. Each node of the tree will include the key and the location (array index) where to retrieve the record.

Each node should be a **structure template** with the following attributes:

- Key (Type)
- Index (LinkedList)
- rlink (pointer to right subtree)
- llink (pointer to left subtree)

Index should be a **linked list** so that every key in the tree remains unique, but, in case of records with the same key, it will be possible to store multiple locations by adding them to the linked list in the node.

Nodes must be organized in a binary AVL tree.

Retrieving records

When asked to retrieve a record, you will look for it in the array list using linear search, and then using the indexes in the binary tree. Checking the value stored in an item of the array or in a node of the tree counts as 1 access. You will need to count all the accesses necessary to find the record and add them to your output file. For example, if the item we are looking for is the 40th element of the array, we will need 40 accesses to find it using linear search.

Note that you need to account for one extra access to the array when using indexes! For example, if I visit 3 nodes in the binary tree before finding the key, then I need one more access to retrieve the record from the array list using the index, for a total of 4 accesses. You do not need to add the accesses to the nodes of the linked list in case a key is associated with multiple indexes.

Multiple records, or records retrieved using different keys, in case the file includes more than one, should be separated by 1 line when printed.

You may be asked to retrieve a record using the student ID. In that case, you should use **binary search** directly on the array list and the output should include only the number of accesses without indexing.

Requirements

- **It is NOT allowed to use vector classes or other classes provided in the STL.**
- **You must delete all dynamically allocated memory.**
- Your C++ code must be clear, indented and commented.
- Your program will be tested with GNU C++. Therefore, you are encouraged to work on Unix, or at least make sure that your code compiles and runs successfully on the server.
- You can use other C++ compilers, but the TAs cannot provide support or test your programs with other compilers.
- The output file must contain the result in the format specified.

Testing for grading:

- All the cpp files will be automatically compiled. Make sure you have only 1 file containing main(). If there is an error in compilation it will not be executed. **Programs that do not compile on the server will receive a score of 0.** You are responsible of the content of your submission folder.
- **The name of your submission folder must be hw3**. The folder must not be nested in another folder. Make sure that the TAs have read access to your folder.
- **Submitted files must be limited to headers and source files (.h and .cpp).**
- Your program should not crash, halt unexpectedly, take an unusual amount of time to run (more than 10 seconds) or produce unhandled exceptions.
- Your program will be tested with 10 test cases, going from easy to difficult.

DEADLINE: April 13th, 11:59PM – NO LATE SUBMISSIONS

Grading

- A program not submitted by the deadline or that does not compile is worth zero points.
- A program that compiles but does not work is worth 10 points.
- A code with no comments will receive a 5 points penalty.
- The classes used in the program must be implemented exactly as described. Failing to follow the specifications will result in point deductions.
- **Bonus (10 pt): Retrieve range.** As a bonus, you will be asked to retrieve not an exact match, but all records within a range (inclusive). Each individual record should be followed by the number of accesses necessary to retrieve it.

  Example: GPA: 3.00-4.00
  Result:
  03985, Carita, Palencia, 1988-06-14, CS, 3.24
  Without index: 22 accesses
  With index: 4 accesses

  00987, Hyman Tschanz, 1990-06-20, Math, 3.92
  Without index: 12 accesses
  With index: 2 accesses

- **A program with memory leaks (non-deleted dynamic memory) will receive a 20 points penalty.**

Plagiarism and cheating: C++ code is individual: it cannot be shared (other than small fragments used in class or in lab examples). Programs will be compared for plagiarism. If the TAs detect that a portion of your C++ code is highly similar to C++ on the Internet or other student the grade will be decreased at least 50%.