ECE244 Programming Fundamentals

Lab Assignment #5: A Student-Marks Database

1. Objectives

The objectives of this assignment are (1) to provide you with more practice on the use of the various C++ concepts/constructs introduced in the lectures so far, including classes, arrays, dynamic memory allocation, pointers, I/O, and dynamic data structures; and (2) to provide you with more practice on the use of the `Make` utility and the debugger. This will be done in the context of implementing a simple student-marks database.

2. Problem Statement

In this assignment, you will implement a simple database to store and retrieve student marks in a school term. You will essentially implement two classes: `studentRecord` and `studentDB`. The `studentRecord` class will be used to create objects that hold a single student's information. The `studentDB` class will be used to create a database of `studentRecord` objects. You will implement two variants of the `studentDB` class; one is array based, while the other is linked-list based.

2.1 The `studentRecord` Class

The `studentRecord` class has fields to represent the student number, student last and first names, and 5 marks. It also has the following methods associated with it.

- `studentRecord ()`. This is the default constructor. It creates an empty student record.
- `~studentRecord ()`. This is the destructor. It `delete`s all dynamically allocated components of the student record.
- `void setStudentNumber (unsigned int studentNum)`. This method sets the student number of the student record to `studentNum`.
- `void setFirstName (char *firstName)`. This method sets the student first name of the student record to the value of the string `firstName`.
- `void setLastName (char *lastName)`. This method sets the student last name of the student record to the value of the string `lastName`, which must exist.
- `void setMark (int index, unsigned int mark)`. This method sets the mark at index `index` of the student record to `mark`. The marks are indexed 0 to 4. The method does not check that `index` is in this range; the caller must do so. A mark is between 0 and 100, and the caller must also check this.
- `unsigned int getStudentNumber ()`. This method returns the student number of the student record.

- `char *getFirstName ()`. This method returns a pointer to the first name of the student record.

- `char *getLastName ()`. This method returns a pointer to the last name of the student record.

- `unsigned int getMark (int index)`. This method returns the mark at index `index` of the student record. The marks are indexed 0 to 4. The method does not check that `index` is in this range; the caller must do so. A mark is between 0 and 100.

- `void print ()`. This method prints the student record to the standard output, in the following format:

  Student number: `number` ↵
  Student name: `lastname`, `firstname` ↵
  Student marks: `m1`, `m2`, `m3`, `m4`, `m5`

  The student number should be printed as a 9-digit integer. The ↵ character indicates `newline` (i.e., `endl`). Note its absence after the last list of the output.

2.1 The `studentDB` Class

The database has fields to represent the structure of the database and the following methods associated with it:

- `studentDB ()`. This is the default constructor. It creates an empty database.

- `~studentDB ()`. This is the destructor. It deletes all the records in, and the structures of, the database.

- `bool insert (studentRecord * newRecord)`. This method inserts the record pointed to by `newRecord` into the database. If a record with the same student number already exists in the database, `false` is returned. The database must not be full when this method is called.

- `bool retrieve (unsigned int studentNum, studentRecord * searchRecord)`. This method searches the database for a record with a student number `studentNum`. If the record is found, then its contents are copied into the student record `searchRecord` (which must exist before the method is invoked), and `true` is returned. Otherwise, a `false` is returned and the contents of `searchRecord` are unaffected.

- `bool remove (unsigned int studentNum)`. The method deletes the record with student number `studentNum` from the database. If the record is found and deleted, `true` is returned. Otherwise `false` is returned.

- `void clear ()`. This method clears the database by deleting all the student records in the database, effectively returning the database to its initial empty state.

- `bool isEmpty ()`. This method returns `true` if the database is empty, otherwise, it returns `false`.

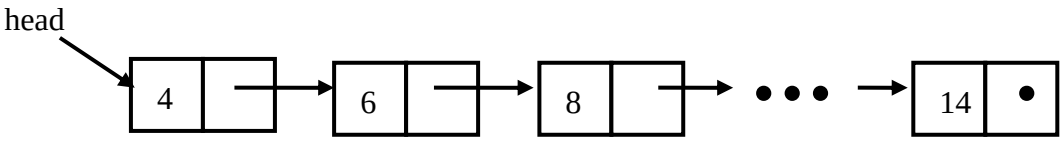
- `bool isFull ()`. This method returns `true` if the database is full, otherwise it returns `false`.

- `void printProbes ()`. This method writes to the standard output the number of ***probes*** performed by the last invocation of the `retrieve` method. A probe is defined as a comparison operation between the search key (i.e., student number) and a key in the database. The number of probes is proportional to the time it takes to search for the search key.

- `void dump ()`. This method dumps out the database to the standard output, one student record at a time (using the `studentRecord`'s `print` function), separated by empty lines, and sorted in ascending order of student numbers.

3. Background

An ordered linked-list is a linked-list for which: (1) each node has a key that is unique; and (2) the key of each node is less than the keys of all its successor nodes in the list. An example of an ordered linked-list is shown below, in which the keys of the nodes are integers.



4. Preparation

This is a more elaborate assignment than the first four assignments. Thus, it pays to start very early. You must also work on the assignment ***before*** you come to the lab. You should prepare an initial implementation of both classes: `studentRecord` and the array-based `studentDB` (Parts I and II below) before you arrive to the lab session of the first week of the assignment. You should prepare an initial implementation of the list-based `studentDB` and `Driver.cpp` (Parts III and IV below) before you arrive to the lab session of the second week of the assignment.

5. Procedure

Create a sub-directory called `lab5` in your `ece244` directory, using the `mkdir` command. Make it your working directory. ***You may modify any of the `.h` files that you will download for this assignment only by adding private function members. You may NOT add data members nor public function members***!

5.1 Part I – The Student Record

Use a browser to download the file `studentRec.h`. It contains the definition of the class `studentRecord`, which is intended to hold one student's information. The purpose of the fields and purpose of function members are described in this file. ***You may modify this file only as described above; i.e., only by adding private function members. You may NOT add data members nor public function members***.

Write the implementation of this class in a file called `studentRec.cpp`.

Write a short program that serves as a test-harness for this class, and call it `test-studentRec.cpp`. For example, you may want to write a short program that prompts the user for student numbers, student names, and student marks, creates a `studentRecord` object, and then prints it using the `print` method. This program will be similar to the command parser program you developed as part of a previous lab assignment.

Write a `Makefile` that can be used to generate the executable of the test-harness. The `Makefile` should include lines that look like:

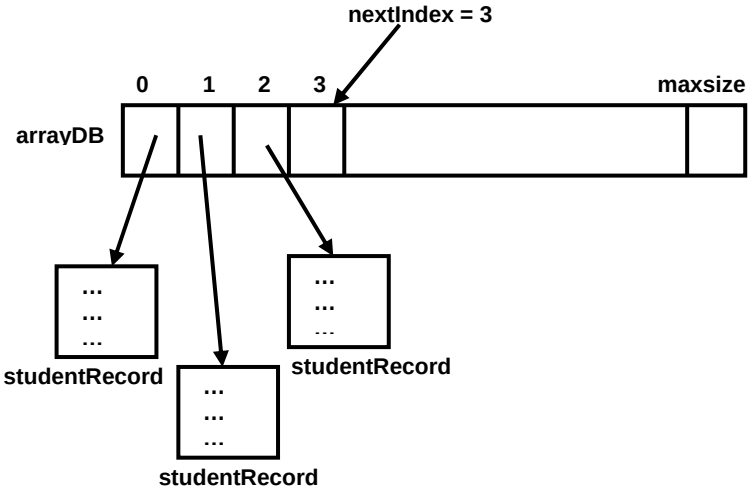
```
test-studentRec:    test-studentRec.o studentRec.o
          g++ test-studentRec.o studentRec.o –o test-studentRec

test-studentRec.o:  test-studentRec.cpp studentRec.h
          g++ -c test-studentRec.cpp
```

You can make the test-harness by typing:

```
make test-studentRec
```

5.2 Part II – The Array-based Database

The array-based database implementation assumes that the database consists of an array of pointers to `studentRecord`s, as shown below.



The integer variable `next_index` indicates the index of the next array location at which a pointer to a new record can be inserted. Pointers to records are stored in the array in the order in which they are inserted into the database, one at a time, starting with index 0. That is, the pointer to first record inserted into the database is stored at `_arrayDB[0]`, the second at index `_arrayDB[1]`, and so on.

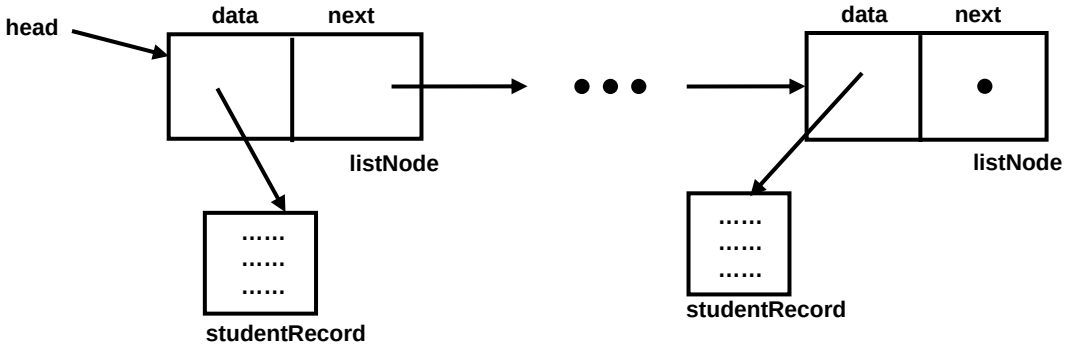
Use a browser to download the file `arrayDB.h`. It contains the definition of the class `studentDB`. ***You may modify this file only as decribed above***. Write the implementation of this class in a file called `arrayDB.cpp`.

Write a short program that serves as a test-harness for this class, and call it `test-arrayDB.cpp`.

Extend your `Makefile` from Part I, to generate the executable of this test-harness.

5.3 Part III – The List-based Database

The list-based implementation assumes that the database is made of an ordered linked-list, with data being pointers to `studentRecord`s, as shown below.



The `next` pointer field in each `listNode` is a pointer to the next `listNode` in the list. The list is "sorted" by student numbers. That is, the `studentNumber` field of each `studentRecord` is they key of the list.

Use a browser to download the files `listNode.h` and `listDB.h`. They contain the definitions of the class `listNode` and the list-based class `studentDB`, respectively. ***You may modify these file only as described above***. Write the implementation of these classes in two files, called `listNode.cpp` and `listDB.cpp`, respectively.

Write a short program that serves as a test-harness or this class, and call it `test-listDB.cpp`. Extend your `Makefile` from Part II, to generate the executable of this test-harness.

5.4 Part IV – The Driver Program

In this part of the assignment, you will write the user interface to the database in the `main` function. The function should first create an empty database. It should then input from `cin` a sequence of commands. Each command consists of an operation, followed by its parameter. The command and the parameters are separated by white spaces. Your function should process commands until the end-of-file (`eof`) is encountered. The commands and their parameters are:

- **`new`** *`num`*. This command creates a new student record with *`num`* as the student number. The last name and first name fields are initialized to "`LNU`" and "`FNU`", respectively. All marks are initialized to 0's. The new record is then inserted into the database. If there already exists a record with the same number, the error message "`Error: a record with student number` *`num`* `already exists.`" is printed to `cout`. If the database if full, thn the error message "`Error: database is full.`" is printed to `cout`. Otherwise no output is produced.

- **`locate`** *`num`*. This command locates the record with the student number *`num`* in the database, and prints it contents to `cout`. If no such record exists, the error message "`Error: no record with student number` *`num`* `exists.`" is printed to `cout`.

- **`updatename`** *`num lname fname`*. This command locates the record with the student number *`num`* and updates its last name and first name fields with *`lname`* and *`fname`*, respectively. If no such record exists, the error message "`Error: no record with student number` *`num`* `exists.`" is printed to `cout`. If the operation is successful, the contents of the updated record are printed to `cout`.

- **`updatemark`** *`num idx mark`*. This command locates the record with the student number *`num`* and updates its mark at index *`idx`* with *`mark`*. If no such record exists, the error message "`Error: no record with student number` *`num`* `exists.`" is printed to `cout`. If the operation is successful, the contents of the updated record are printed to `cout`. Further, *`idx`* must be in the range 0-4, otherwise the error message "`Error:` *`idx`* `is out of the range 0-4.`" is printed to `cout`. Also, *`mark`* must be in the range 0-100, otherwise the error message "`Error:` *`mark`* `is out of the range 0-100.`" is printed to `cout`.

- **`delete`** *`num`*. This command deletes the record with the student number *`num`* in the database. If no such record exists, the error message "`Error: no record with student number` *`num`* `exists.`" is printed to `cout`.

- **`printall`**. This command prints all the records in the database, sorted in ascending order of student number, separated by spaces.

- **`printprobes`** *`num`*. This command locates the record with the student number *`num`* in the database. If no such record exists, the error message "`Error: no record with student number` *`num`* `exists.`" is printed to `cout`. Otherwise, the number of probes (as defined above) is printed to `cout`.

- **`deleteall`**. This command deletes all the records in the database, retuning it to the empty state.

The following is an example of commands and their outputs. The example assumes an empty database at the beginning. The output is shown in red.

```
% ./Driver
new 123456789
locate 987654321
Error: no record with student number 987654321 exists.
locate 123456789
Student number: 123456789
Student name: LNU, FNU
Student marks: 0, 0, 0, 0, 0
updatename 123456789 Abdelrahman Tarek
Student number: 123456789
Student name: Abdelrahman, Tarek
Student marks: 0, 0, 0, 0, 0
updatemark 123456789 2 500
Error: 500 is out of the range 0-100.
updatemark 123456789 2 99
Student number: 123456789
Student name: Abdelrahman, Tarek
Student marks: 0, 0, 99, 0, 0
updatemark 123456789 5 98
Error: 5 is out of the range 0-4.
updatemark 123456789 4 98
Student number: 123456789
Student name: Abdelrahman, Tarek
Student marks: 0, 0, 99, 0, 98
delete 987654321
Error: no record with student number 987654321 exists.
new 123456789
Error: a record with student number 123456789 already exists.
new 123454321
updatename 123454321 Voss Michael
Student number: 123454321
Student name: Voss, Michael
Student marks: 0, 0, 0, 0, 0
updatemark 123454321 1 57
Student number: 123454321
Student name: Voss, Michael
Student marks: 0, 57, 0, 0, 0
printall
Student number: 123454321
Student name: Voss, Michael
Student marks: 0, 57, 0, 0, 0

Student number: 123456789
Student name: Abdelrahman, Tarek
Student marks: 0, 0, 99, 0, 98

delete 123454321
printall
Student number: 123456789
```

```
Student name: Abdelrahman, Tarek
Student marks: 0, 0, 99, 0, 98

delete 123456789
(eof)
```

Extend your `Makefile` from the previous, to generate the executable of this driver. You must test your driver with both your array and list based implementations of the `studentDB` class. You must call the array based driver `arrayDriver.cpp`, with an executable called `arrayDriver`. You must call the list based driver `listDriver.cpp`, with an executable called `listDriver`.

For this part of the assignment, you will want to reuse code that you have written for Assignment 3.

6. Deliverables

Submit the following components of your implementations: `listNode.h`, `listNode.cpp`, `studentRec.h`, `arrayDB.h`, `listDB.h`, `studentRec.cpp`, `arrayDB.cpp`, `listDB.cpp`, `arrayDriver.cpp`, `listDriver.cpp` and `Makefile` using the `submitece244f` command as follows:

```
submitece244f 5 studentRec.cpp arrayDB.cpp listDB.cpp arrayDriver.cpp
                listDriver.cpp listNode.cpp listNode.h
                studentRec.h arrayDB.h listDB.h Makefile
```