NWEN 241 Assignment 4

"Database Management System"

Release Date: **6 May 2019**

Submission Deadline: **20 May 2019, 23:59**

In the previous assignment, you implemented a database table using a dynamically allocated array (of structures) for holding records in memory. In this assignment, you will use vector and linked list to implement the same database table. You will also use File input/output operations to retrieve the table from a comma-separated value (CSV) file.

Sample code showing an example on how you can test your code are provided under the `files` directory in the archive that contains this file.

Full marks is 100.

Instructions and Submission Guidelines:

- You should provide appropriate comments to make your source code readable. If your code does not work and there are no comments, you may lose all the marks. See the marking criteria at the end of this document for details about the marks for commenting.
- You should follow a consistent coding style when writing your source code. See the marking criteria at the end of this document for details about the marks for coding style.
- Submit the required files to the Assessment System (`https://apps.ecs.vuw.ac.nz/submit/NWEN241/Programming_Assignment_4`) on or before the submission deadline.
- Late submissions (up to 48 hours from the submission deadline) will be accepted but will be penalized. No submissions will be accepted 48 hours after the submission deadline.

Program Design

(This is already partly discussed in Assignment #3.)

A fundamental concept in DBMS is the *table*. A table consists of zero or more *records* or entries, and each record can have one or more *fields* or columns. An example of a table for storing information about movies is shown below:

id	title	year	director
13	The Shawshank Redemption	1994	Frank Darabont
25	The Godfather	1972	Francis Ford Coppola
31	The Dark Knight	2008	Christopher Nolan
40	The Godfather: Part II	1974	Francis Ford Coppola
55	The Lord of the Rings: The Return of the King	2003	Peter Jackson
72	Pulp Fiction	1994	Quentin Tarantino

This table contains 6 rows or records. Each record has 4 fields, namely, id, title, year, and director.

In this assignment, you will focus on implementing a single database table with 4 fields (id, title, year, and director). A C structure with tag `movie` will be used for holding a table record. The structure declaration is given below and is defined within `dbms2` namespace in `dbms2.hh`:

```
namespace dbms2 {
    struct movie {
        unsigned long id;
        char title[50];
        unsigned short year;
        char director[50];
    };
}
```

Task 1.

Basics [30 Marks]

In this task, you will declare a C++ abstract class for representing a database table. The class should be named `AbstractDbTable` and should have the following public members:

- A function named `rows()` which returns an integer, and should not modify any member variables. This should be declared as a pure virtual function. *In the implementation*, the function should return the number of rows in the table.

- A function named `show()` which accepts an integer parameter, and should not modify any member variables. This should be declared as a pure virtual function. *In the implementation*, the function should display the information stored in a row. You are free to format the print out, but all fields of the row should be shown. The input parameter indicates the row number of the record to be displayed. If the record exists, the function should return `true`, otherwise, it should return `false`.

- A function named `add()` which accepts a reference to a `movie` structure. This should be declared as a pure virtual function. *In the implementation*, the function should insert a record into the table. The input parameter contains the record details to be stored in the table. The function should return `true` if the record was successfully inserted into the table, otherwise, it should return `false`.

- A function named `remove()` which accepts an unsigned long integer. This should be declared as a pure virtual function. *In the implementation*, the function should remove a record from the table. The input parameter contains the id of the record to be removed. The function should return `true` if the removal was successful, otherwise, it should return `false`.

- A function named `get()` which accepts an integer parameter and should not modify any member variables. This should be declared as a pure virtual function. *In the implementation*, the function should return a pointer to a `movie` structure. The input parameter indicates the row number of the record to be returned.

- `bool loadCSV(const char *infn)`: This should be declared as a normal (non-virtual) function. See Task 5 for more details.

- `bool saveCSV(const char *outfn)`: This should be declared as a normal (non-virtual) function. See Task 6 for more details.

Note that you have implemented the first four functions in Assignment #3 for a database table implemented as an array of structures. In this assignment, you will implement them for a database table that uses a vector (Task 4) and a linked list (Task 7).

The class should be defined within `dbms2` namespace.

Save the class in a header file named `dbms2.hh`.

Task 2.

Basics [10 Marks]

Declare a C++ class named `VectorDbTable` that is a subclass of `AbstractDbTable`. You will use this class to implement a database table using a `vector`. You may declare constructors, destructors, additional member variables and functions. Provide sufficient comments to justify the declaration of these additional members.

The class should be defined within `dbms2` namespace.

Save the class in a header file named `vdb.hh`.

Task 3.

Basics [10 Marks]

Declare a C++ class named `LinkedListDbTable` that is a subclass of `AbstractDbTable`. You will use this class to implement a database table using a linked list data structure. You may declare constructors, destructors, additional member variables and functions. Provide sufficient comments to justify the declaration of these additional members.

The class should be defined within `dbms2` namespace.

Save the class in a header file named `lldb.hh`.

Task 4.

Completion [10 Marks]

Provide an implementation of the class `VectorDbTable` as declared in `vdb.hh`.

Save the implementation in `vdb.cc`.

(Hint: Implementing a class means implementing all unimplemented member functions, constructors and destructors declared in the class declaration.)

Task 5.

Completion [10 Marks]

Provide an implementation of the member function

```
bool loadCSV(const char *infn)
```

in the `AbstractDbTable` class. The input parameter `infn` denotes the file name of a comma-separated value (CSV) file to be loaded. In a valid CSV file, a line represents a record. An example of a line in a valid CSV file is shown below:

```
13,The Shawshank Redemption,1994,Frank Darabont
```

which has 4 fields (id, title, year, and director) separated by commas. For simplicity, assume that the 2nd (title) and 4th (director) fields would not contain commas.

This function should perform the following:

- Open the file `infn` for reading. You may use either C or C++ File I/O.
- Read in all lines from `infn`. Add every line (which corresponds to a record) from the file into the table. When a line not following the expected format is encountered, the reading of the rest of the lines is terminated.
- Close the file.

The function should return `false` if:

- The file `infn` does not exist or cannot be opened for reading.
- The file `infn` is not a valid CSV file (at least one of the lines does not follow the expected format.)

Otherwise, it should return `true`.

Save the implementation in `dbms2.cc`.

(Hint: You can use the `add()` member function to add a line of record.)

Task 6.

Completion [10 Marks]

Provide an implementation of the member function

```
bool saveCSV(const char *outfn)
```

in the `AbstractDbTable` class. The input parameter `outfn` denotes the file name to write to.

This function should perform the following:

- Open the file `outfn` for writing. The file must be emptied. You may use either C or C++ File I/O.
- Write every record from the table into the file. A record must be written as a comma-separated value (see Task 5 for the specifications of a line of record.)
- Close the file.

The function should return `false` if:

- The file `outfn` cannot be opened for writing.
- Errors were encountered while writing to the file.

Otherwise, it should return `true`.

Save the implementation in `dbms2.cc`.

(Hint: You can use the `get()` member function to get a line of record.)

Task 7.

Challenge [10 Marks]

Provide an implementation of `LinkedListDbTable` as declared in `lldb.hh`. Your implementation should **not** use the C++ standard template library. This means that you should implement the link list data structure in your code (which involves the use of C structure, pointers, and dynamic memory allocation).

Save the implementation in `lldb.cc`.

(Hint: Implementing a class means implementing all unimplemented member functions, constructors and destructors declared in the class declaration.)

Task 8.

Challenge [10 Marks]

Write a `main()` function that accepts command line arguments. Save the implementation in `dbcmd.cc`.

The program should perform the following:

1. Create an instance of a database table. You may use either `VectorDbTable` or `LinkedListDbTable`.
2. Load a CSV database table file named `default.csv` (provided).
3. If the first command line argument is `showall`, it will display all rows in the table.
4. If the first command line argument is `show`, then it must be followed by a second command line argument. The second argument is the row number of the record to be displayed.
5. Destroy instance of database table.

If there are more command line arguments than required, the excess arguments are simply ignored.

To illustrate, suppose that the program is compiled as `dbcmd`. Suppose further that the file `default.csv` contains the following lines:

```
13,The Shawshank Redemption,1994,Frank Darabont
25,The Godfather,1972,Francis Ford Coppola
31,The Dark Knight,2008,Christopher Nolan
```

Then if we execute

```
dbcmd showall
```

The output should be something like (depending on how you implemented the `show()` member function)

```
13  The Shawshank Redemption   1994   Frank Darabont
25  The Godfather              1972   Francis Ford Coppola
31  The Dark Knight            2008   Christopher Nolan
```

If we execute

```
dbcmd showall 100
```

The third argument 100 is simply ignored. Hence, the output should be the same as above.

If we execute

```
dbcmd show 0
```

The output should be something like

```
13  The Shawshank Redemption   1994   Frank Darabont
```

which is the first row of the table. Note that row index starts from 0.

If we execute

```
dbcmd show 100
```

The output should be something like

```
Error: Row 100 does not exist.
```

Marking Criteria for Tasks 1 – 3:

Criteria	Weight	Expectations for Full Marks
Commenting	10%	Source code contains sufficient and appropriate comments
Coding Style	10%	Source code is formatted, readable and uses a coding style consistently
Correctness	40%	Addresses all specifications and correctly uses syntax in the declarations and/or definitions
Completeness	40%	Declaration and/or definition of all required members
	100%	

Marking Criteria for Tasks 4–8:

Criteria	Weight	Expectations for Full Marks
"Compilability"	10%	Source code compiles without warnings
Commenting	10%	Source code contains sufficient and appropriate comments
Coding Style	10%	Source code is formatted, readable and uses a coding style consistently
Correctness	70%	Implements all specifications correctly and handles all possible cases correctly
	100%	