NWEN 241 Assignment 4

Weeks 9-11

Release Date: **11 May 2023**

Submission Deadline: **29 May 2023, 23:59**

In this assignment, you will be asked to implement a simple database table using C++ programming constructs.

You must submit the required files to the Assessment System (`https://apps.ecs.vuw.ac.nz/submit/NWEN241/Assignment_4`). Any assignment submitted up to 24 hours after the deadline will be penalised by 20%, and any assignment submitted between 24 and 48 hours after the deadline will be penalised by 40%. Any assignment submitted 48 hours or more after the deadline will not be marked and will get 0 marks. Important: The Assessment System is configured **not to accept submissions that do not compile.** So please test that your code compiles using a C++ compiler before submitting it.

Full marks is 100. The following table shows the overall marks distribution:

Criteria	**Marks**	**Expectations for Full Marks**
Compilation	5	Compiles without warnings
Comments	10	Sufficient and appropriate comments
Code Quality	15	Efficient code and use of consistent coding style
Correctness	70	Handles all test cases correctly (see marks distribution below)
Total	**100**	

For the **Correctness** criteria, the following table shows the marks distribution over the different task types:

Task Type	**Marks**
Core	45
Completion	15
Challenge	10
Total	**70**

Introduction

This assignment will test your application of the conceptual knowledge of C++ fundamentals to solve practical programming tasks. You may only use the Standard C++ Library to perform the tasks in this part.

Sample codes showing examples on how you can test your implementation are provided under the `files` directory in the archive that contains this handout file.

Commenting

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.

Coding Style Code quality refers to (*i*) the use of efficient coding techniques and (*ii*) use of consistent coding style or standard.

When writing your source code, strive for *coding efficiency* which covers the following aspects: (*i*) elimination of unessential operations and variables; (*ii*) creation of functions to contain blocks of code that are used repeatedly; (*iii*) minimization of loop iteration; and (*iv*) avoiding the use of global variables (i.e. variables that are declared outside any function.);(*v*) use built-in functions where appropriate.

In addition, you should follow a consistent coding style when writing your source code. Coding style (aka coding standard) refers to the use of appropriate indentation, proper placement of braces, proper formatting of control constructs, and many others. Following a particular coding style consistently will make your source code more readable. There are many coding standards available (search "C coding style"), but we suggest you consult the *lightweight* Linux kernel coding style (see `https://www.kernel.org/doc/html/v4.10/process/coding-style.html`). The relevant sections are Sections 1, 2, 3, 4, 6, and 8. Note that you do not have to follow every recommendation you can find in a coding style document, you just have to apply that style consistently.

Program Specifications

(This is already partly discussed in Assignment #2.)

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 that stores information about movies is shown below:

id	title	year	director
10	The Goonies	1985	Richard Donner
23	The Godfather	1972	Francis Ford Coppola
37	Avatar	2009	James Cameron
43	Citizen Kane	1941	Orson Welles
14	The Fellowship of the Ring	2001	Peter Jackson

This table contains 5 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 structure with tag movie will be used for holding a table record. The structure declaration is given below and is defined within `nwen` namespace in `abstractdb.hpp`:

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

Task 1.

Core [15 Marks]

Declare a C++ abstract class for representing a database table. The class should be named `AbstractDbTable` and should have the following public members:

- A pure virtual function named `rows()` which returns an integer and does not modify any member variables.
- A pure virtual function named `get()` which accepts an integer parameter, returns a pointer to a `movie`, and does not modify any member variables.
- A pure virtual function named `add()` which accepts a `movie` structure.
- A pure virtual function named `update()` which accepts an unsigned long integer and a (non-pointer) `movie` structure parameters and returns a boolean.
- A pure virtual function named `remove()` which accepts an unsigned long integer parameter and returns a boolean.
- A function named `loadCSV` which accepts a C++ string parameter and returns a boolean.
- A function named `saveCSV` which accepts a C++ string parameter and returns a boolean.

Note that the last two member functions are not pure virtual.

The abstract class should be defined within `nwen` namespace. Save the class declaration in a header file named `abstractdb.hpp`.

Task 2.

Core [10 Marks]

Declare C++ class named `VectorDbTable` that extends `AbstractDbTable`. The class should be concrete, which means that it should provide declarations of all pure virtual members of `AbstractDbTable`. Do not include declarations for the member functions `loadCSV()` and `saveCSV()`.

You must use a `vector` as table for storing the movie records. You may declare a default constructor, additional member variables and functions. Provide sufficient comments to justify the declaration of these additional members.

The class `VectorDbTable` should be declared within `nwen` namespace. Save the class declaration in a header file named `vectordb.hpp`.

Task 3.

Core [20 Marks]

Provide implementations for the following member functions of `VectorDbTable` (which are inhereted from `AbstractDbTable`):

- `rows()`: returns the number of rows in the table.
- `get()`: returns a pointer to a `movie` structure. The input parameter indicates the row number of the record to be returned.
- `add()`: inserts a record into the table. The input parameter contains the record details to be stored in the table. If there an existing record in the table with the same id, then the insertion should not proceed. The function should return `true` if the record was successfully inserted into the table, otherwise, it should return `false`.
- `update()`: updates a record in the table. The unsigned long integer input parameter indicates the id of the record to be updated and the structure is the data the row should be updated with. The function should return `true` if the update was successful, otherwise, it should return `false`.
- `remove()`: removes 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`.

Save the implementation of the above member functions in a C++ source file named `vectordb.cpp`.

Task 4.

Completion [15 Marks]

Provide an implementation for the member function `saveCSV()`. Note that `saveCSV()` is declared in `AbstractDbTable`, hence, you will save your implementation in a C++ source file named `abstractdb.cpp`.

As mentioned in Task 1, `loadCSV()` accepts a C++ string parameter which denotes the filename to which to save the contents of the table.

The function `saveCSV()` should perform the following:

1. Open the file for writing using C++ File I/O. You are not allowed to use any of the C File I/O functions. The file must be emptied.
2. Write every record from the table into the file. A record must be written as a comma-separated value (one record per line). For instance, for a movie record with id `37`, title `Avatar`, year `2009`, and director `James Cameron`, the output should be

```
37,"Avatar",2009,"James Cameron"
```

3. Close the file.

The function should return `false` if:

1. The file cannot be opened for writing.
2. Errors were encountered while writing to the file.

Otherwise, it should return `true`.

Save the implementation of this function in `abstractdb.cpp`.

Task 5.

Challenge [10 Marks]

Provide an implementation for the member function `loadCSV()`. Note that `loadCSV()` is declared in `AbstractDbTable`, hence, you will save your implementation in a C++ source file named `abstractdb.cpp`.

As mentioned in Task 1, `loadCSV()` accepts a C++ string parameter which denotes the 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:

```
37,"Avatar",2009,"James Cameron"
```

which has 4 fields (id, title, year, and director) separated by commas.

The function `loadCSV()` should perform the following:

1. Open the file for reading using C++ File I/O. You are not allowed to use any of the C File I/O functions.
2. Read in all lines from the file. Add every line (which corresponds to a record) from the file into the table using the `add()` function declared in `AbstractDbTable`. When a line not following the expected format is encountered, the reading of the rest of the lines is terminated.
3. Close the file.

The function should return `false` if:

1. The file does not exist or cannot be opened for reading.
2. The file 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 of this function in `abstractdb.cpp`.