NWEN 241 Assignment 3

"Database Management System"

Release Date: **8 April 2019**

Submission Deadline: **6 May 2019, 23:59**

In this assignment, you will implement a simple database management system (DBMS). DBMS is a systems program that performs storage, retrieval, and updating of data in a computer system. DBMS addresses two major problems in conventional file-based systems: data redundancy and data dependence.

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_3`) 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.

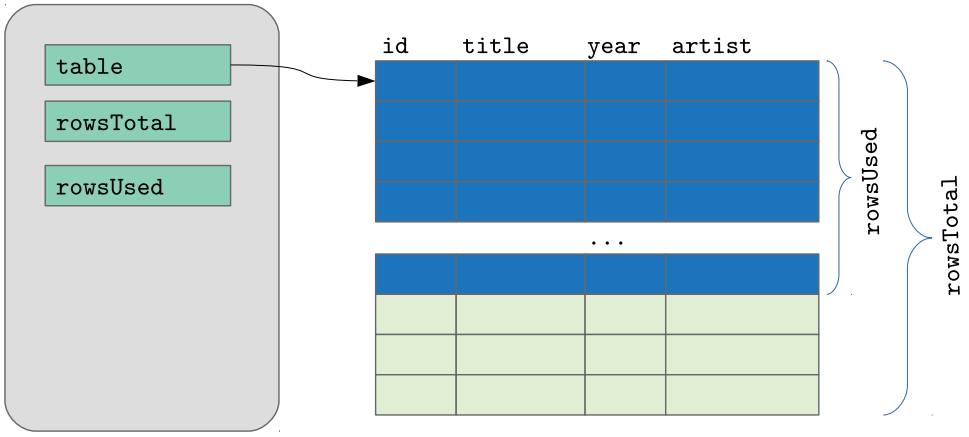
Program Design

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 music albums is shown below:

id	title	year	artist
10	The Dark Side of the Moon	1973	Pink Floyd
14	Back in Black	1980	AC/DC
23	Their Greatest Hits	1976	Eagles
37	Falling into You	1996	Celine Dion
43	Come Away With Me	2002	Norah Jones
55	21	2011	Adele

This table contains 6 records. Each record has 4 fields, namely, `id`, `title`, `year`, and `artist`.

In this assignment, you will focus on implementing a single database table with 4 fields (id, title, year, and artist). To guide you in the implementation, a high-level design of the table is shown below:



C++ Class Array of Structures

The design consists of a C++ class and an array of structures. The C++ class should have the following member variables:

- `table`: a pointer to an array of structures that is dynamically allocated

- `rowsTotal`: the total number of rows in the array of structures
- `rowsUsed`: the number of rows in the array of structures that contains valid records.

The array of structures will hold the records. This array should be dynamically adjusted using the following scheme:

- Upon instantiation of the class, memory should be dynamically allocated for holding 5 records. `rowsTotal` should be set to 5 and `rowsUsed` should be set to 0.
- When adding a record and there is unused space in the table (`rowsUsed` is less than `rowsTotal`), the record is stored in the next unused row, and `rowsUsed` is updated accordingly.
- When adding a record and there is no space in the table (`rowsUsed` is equal to `rowsTotal`), additional memory for holding 5 records should be allocated. Then, the record is stored in the next unused row, and `rowsUsed` and `rowsTotal` are updated accordingly.
- When removing a record, records after the removed record should be moved up by one row so that there will be no gaps in between used rows. When the number of unused rows reaches 5, the unused rows should be released, and `rowsUsed` and `rowsTotal` are updated accordingly. When removing the last remaining record, do not release the remaining unused rows.
- Upon destruction of the class, allocated memory should be freed.

Task 1.

Basics [15 Marks]

In this task, you will declare a C structure for holding one (1) table record. The structure should have a tag `album` with the following members:

- `id`: an unsigned long integer
- `title`: an array of characters with length 100
- `year`: an unsigned short integer

- `artist`: an array of characters with length 100

The structure should be defined within `dbms` namespace.

Save the structure in a header file named `dbms.hh`.

Task 2.

Basics [15 Marks]

Declare a C++ class for holding information about the table, as well as supporting table operations specified in Tasks 5–7. The class should be named `DbTable` and should be defined within `dbms` namespace.

Minimally, the class should have member variables `table`, `rowsTotal`, and `rowsUsed` as described in the *Program Design* section. These variables should be unsigned integers and should be private.

In addition to the member functions specified in Tasks 5–7, the class should have the following public member functions that are defined inline:

- A function named `rows()` which should return `rowsUsed`.
- A function named `allocated()` which should return `rowsTotal`.

You may declare additional member variables and functions needed by your implementations.

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

Task 3.

Basics [15 Marks]

Provide an implementation of the default constructor for the `DbTable` class.

As mentioned in the *Program Design* section, during instantiation, memory should be dynamically allocated for holding 5 records. Use either `malloc()` or `calloc()` for this purpose.

Save the implementation in `dbms.cc`.

Task 4.

Completion [15 Marks]

Provide an implementation of a destructor for the `DbTable` class.

As mentioned in the *Program Design* section, during destruction, allocated memory should be freed.

Save the implementation in `dbms.cc`.

Task 5.

Completion [15 Marks]

Provide an implementation of a member function for displaying the information stored in a row. You are free to format the print out, but all fields of the row should be displayed. The function should be called `show()` and should take an unsigned integer as parameter. This 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`.

Task 6.

Completion [15 Marks]

Provide an implementation of a member function for a adding a record into the database table. The function should be called `add()` and should take a *reference* to `album` structure as input parameter. The input parameter contains the record details to be stored in the table.

As mentioned in the *Program Design* section:

- When adding a record and there is unused space in the table (`rowsUsed` is less than `rowsTotal`), the record is stored in the next unused row, and `rowsUsed` is updated accordingly.
- When adding a record and there is no space in the table (`rowsUsed` is equal to `rowsTotal`), additional memory for holding 5 records should be allocated. Then, the record is stored in the next unused row, and `rowsUsed` and `rowsTotal` are updated accordingly.

The function should always return `true` unless the allocation of additional

memory (when needed) fails.

Task 7.

Challenge [10 Marks]

Provide an implementation of a member function for a removing a record from the database table. The function should be called `remove()` and should take an unsigned long integer as input parameter. The input parameter specifies the id of the record to be removed.

As mentioned in the *Program Design* section:

- When removing a record, records after the removed record should be moved up by one row so that there will be no gaps in between used rows. When the number of unused rows reaches 5, the unused rows should be released, and `rowsUsed` and `rowsTotal` are updated accordingly. When removing the last remaining record, do not release the remaining unused rows.

The function should return `true` if the removal was successful, otherwise, it should return `false`.

Marking Criteria for Tasks 3–7:

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
Dynamic Memory Allocation	30%	Uses the specified dynamic memory allocation functions correctly
Correctness	40%	Handles all possible cases correctly
	100%	

Marking Criteria for Tasks 1 and 2:

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%	