NWEN 241 Assignment 4

(Weeks 8–9 Topics)

Release Date: **02 June 2020**

Submission Deadline: **17 June 2020, 23:59**

This assignment is divided into 2 parts.

- In Part I (Tasks 1–3), you will be asked to answer questions about Weeks 8–9 topics.
- In Part II (Tasks 4–8), you will be asked to define structures, classes and implement functions, the specifications for which are presented in each of the tasks.

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

Task Type	Part I	Part II	**Total**
Core	20	45	**65**
Completion	8	12	**20**
Challenge	5	10	**15**
Total	**33**	**67**	**100**

Part I: Concepts

This part will test your conceptual knowledge of Weeks 8–9 topics. Your answers should be submitted in a plain text file named `part1.txt`.

Task 1.

Core [20 Marks]

1) **[2 Marks]** What is a constructor in C++?

2) **[2 Marks]** What is an abstract class in C++?

3) **[10 Marks]** Given the following declaration of the class `Foo`, are the following valid or invalid ways to construct an object `f` for that class `Foo` (note the difference between parenthesis and curly braces):

```
class Foo
{
public:
    int i = 0;
    Foo();
    Foo(int i);
};
```

(a) `Foo f;`

(b) `Foo f();`

(c) `Foo f(1);`

(d) `Foo f(Foo(1));`

(e) `Foo f = Foo;`

(f) `Foo f = Foo(i = 1);`

(g) `Foo f{};`

(h) `Foo f{1};`

(i) `Foo f = {1};`

(j) `Foo f = {.i = 1};`

4) [2 Marks] Rewrite the following C code, using streams in C++, to produce the same output.

```
int i = 2;
char str[] = "foo";
printf("str[%d] = %c\n", i, str[i]);
```

5) [4 Marks] Define a class `rational` that can represents a rational number with the following properties:

- two private integer data members: `numerator` and `denominator`.
- a public constructor that takes two integers and assigns to the data members using an initializer list.
- a public member function `getFloat` with no arguments that returns the floating point number represented by class.

Task 2.

Completion [8 Marks]

1) **[2 Marks]** What is the difference between an `inline` function and a function-like `macro`.

2) **[4 Marks]** Are the following statements true or false?

   (a) All members of a structure or class are public by default.

   (b) Objects of a class do not share non-static members - every object has its own copy.

   (c) A structure can not have constructors or member functions, unlike a class.

   (d) `cin`, `cout`, `clog`, and `cerr` are all classes used for IO in C++.

3) **[2 Marks]** Consider the following C++ code snippet:

```
namespace foo
{
    int a = 50;
    void increment()
    {
        a++;
    }
}
```

   Write statement that allows the function `increment()` to be called from outside the namespace `foo`, such that:

   (a) all identifiers in namespace `foo` are accessible without the scope resolution operator.

   (b) only the function `increment()` in namespace `foo` is accessible without the scope resolution operator.

Task 3.

Challenge [5 Marks]

1) **[5 Marks]** Consider the following C++ class declaration:

```
namespace foo
{
    class Rectangle
    {
    public:
        virtual int height() const = 0;
        virtual int width() const = 0;
        virtual int area() const = 0;
    protected:
        int height;
        int width;
    };
}
```

Declare a class `Square` in the namespace `bar` such that

- `Square` is a derived class of `Rectangle` and preserves the access specifiers of `Rectangle`.
- `Square` defines an inline constructor that takes a single integer argument initializes the member variables `height` and `width` to that argument.
- `Square` provides an appropriate inline definition that overrides the member functions `height`, `width`, and `area`, while preserving the `virtual` and `const` specifiers.

Part II: Practical Programming

This part will test whether you can apply the conceptual knowledge you have learned in Weeks 8–9 to solve practical programming tasks.

In the programming tasks, you will implement a simple database table using C++ programming constructs.

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

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.

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/C++ coding style"), but we suggest you consult the *lightweight* Linux kernel coding style (`https://www.kernel.org/doc/html/v4.10/process/coding-style.html`). The relevant sections are 1, 2, 3, 4, 6 and 8. Note that you do not have to follow every recommendation you find in a coding style document. If you change, for instance the tab size from 8 to 4, that is fine. 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 music albums 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 4.

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 `show()` which accepts an integer parameter, returns a boolean, 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 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 (`const char *`) parameter and returns a boolean.

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

Task 5.

Core [30 Marks]

Declare and define a C++ class named `VectorDbTable` that extends `AbstractDbTable` within `nwen` namespace. The class `VectorDbTable` should be declared in a header file named `vectordb.hpp` and defined (implemented) in a source file named `vectordb.cpp`. You will use this class to implement a database table using a `vector`. You may declare a default constructor, additional member variables and functions. Provide sufficient comments to justify the declaration of these additional members.

Provide implementations for the following member functions:

- `rows()`: returns the number of rows in the table.
- `show()`: displays the information stored in a row. The input parameter indicates the row number of the record to be displayed. If the record exists, the function should return `true` and show the record, otherwise, it should return `false` and not show anything.
- `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. The function should return `true` if the record was successfully inserted into the table, otherwise, it should return `false`.

Task 6.

Completion [12 Marks]

Provide implementations for the following member functions:

- `update()`: updates a record in the table. The integer input parameter indicates the row number 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`. Note that the

Task 7.

Challenge [10 Marks]

Provide an implementation for the `loadCSV` member function in the `AbstractDbTable` class.

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 artist) separated by commas.

This function should perform the following:

- Open the file for reading using C++ File I/O.
- 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 for `AbstractDbTable`. 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 `abstractdb.cpp`.

Marking Criteria for Tasks 4–6:

Criteria	Weight	Expectations for Full Marks
Compilation	10%	Compiles without warnings
Commenting	10%	Sufficient and appropriate comments
Coding Style	10%	Consistent coding style
Correctness	70%	Implements all specifications and uses the syntax in the declarations and/or definitions
Total	100%	

Marking Criteria for Task 7:

Criteria	Weight	Expectations for Full Marks
Compilation	10%	Compiles without warnings
Commenting	10%	Sufficient and appropriate comments
Coding Style	10%	Consistent coding style
Correctness	40%	Implements all specifications and uses the syntax in the declarations and/or definitions
I/O	30%	Clean file handling with streams closed appropriately
Total	100%	