OOP Project specifications
Class 2020
catalin.boja@ie.ase.ro
www.acs.ase.ro/cpp

My SQLite engine

Why do it ?

This is equivalent to asking yourself why learning ?

How will it help me ?

This assignment is not an “easy one” and it will require your involvement for the next weeks. It requires commitment and weekly involvement. You can’t do it in the last 2-3 days before the deadline.

If you succeed in doing it you will have a practical experience that will help to see how different classes are combined together to develop a “real” software product (SQLite is real and used a lot)

The final solution will be “horrible” from architecture/design perspective but don’t worry. You will gain something you can’t get otherwise, it’s called experience.

What is the final goal ?

To implement a limited version of a very light DB management system, like SQLite https://www.sqlite.org/index.html

Mandatory requirements for all phases

You CANNOT use an external library to do the parsing or the computations

The solution must be implemented using only standard C++ libraries (like iostream, string, fstream, string.h etc) used during courses and laboratories. STL classes are not accepted, unless explicitly required (only in phase 3 is a requirement). As an exception you can use the C++ standard ‘regex’ library (https://en.cppreference.com/w/cpp/regex)

Accepted solutions: only compiler error free and complete solutions are accepted.

Changes to the project source code are managed using a Git repository. Is mandatory that students do multiple commits after each class implementation/update. Each commit must contain a short message detailing the changes. If the phase has less than 5 commits (in different days), it will not be taken into account.

Phase 1 - The command processor requirements

In the following description | means logic OR and [] means the parameter is not mandatory

Implement different C++ classes that will allow you interpret different SQL commands received from console or from a file:

1. The solution will be used to manage a single database - only one, with multiple tables
2. Accepted commands for managing the database structure are CREATE TABLE, CREATE INDEX, DROP TABLE, DROP INDEX, DISPLAY TABLE

a. CREATE TABLE table_name [IF NOT EXISTS] ((column_1_name,type,size, default_value), (column_2_name,type,size, default_value), …) - the command should receive at least 1 column;
b. DROP TABLE table_name
c. DISPLAY TABLE table_name
d. CREATE INDEX [IF NOT EXISTS] index_name ON table_name (column_name) - creates an index only on a single column
e. DROP INDEX index_name
f. accepted types text, integer, float

3. Accepted CRUD commands for managing data are INSERT, SELECT, UPDATE, DELETE
4. The accepted command for the CRUD commands is (use https://www.sqlite.org/lang.html to get an idea)

a. INSERT INTO table VALUES(...); values are separated by , and they have the exact number and order as the table definition
b. DELETE FROM table_name WHERE column_name = value (deletes allows only one column in the where clause)
c. SELECT (at_least_one_column, ...) | ALL FROM table_name [WHERE column_name = value] - the where clause is optional
d. UPDATE table_name SET column_name = value WHERE column_name = value (the SET column may be different than the WHERE one)

At this point you should develop functions that will receive a string (char* or a C++ string) and identify the commands. If the command is not well formatted you should return an error code.

Recommandations

- Start with a generic function that identifies the command type (SELECT, UPDATE, etc)
- Implement specific functions that will validate a specific command type format and parameters
- Use arrays to store commands data

- Define classes for entities which are described by a noun

Output:

- One or more .h file that contains all classes definitions and implementations
- One or more C++ source file (use .cpp extension) that contains main()
- A single C console application that will allow users to write commands from the previous list. The output for each command is
- Error message if the command is not ok
- Details about the command parameters (the type of the command, the columns names and the table name)

Phase 2 - Storing data in files

Reference to the previous phase:

- All commands implemented in the previous stage will generate data that will be saved in files and / or will be displayed on the console;
- The interface of the application is also based on the console and the user enters the commands one after the other;
- Using the files data, use validations that do not allow the user to create 2 tables with the same name or to execute INSERT, DELETE, UPDATE commands on tables that do not exist or do not have the correct structure;
- INSERT, DELETE, and UPDATE commands will have effects on the data files associated with the tables
- CREATE TABLE and DROP TABLE commands will have effects in the files structure (files will be created or deleted).
- Entities that manage files and file operations are implemented by classes (the source code through which the files are accessed must be included in methods belonging to new or existing classes)

Types of files used in this phase

- Text files of commands sent as arguments for the main () function. They are processed when the application is started;
- Text / binary configuration files loaded when the application is started. The files contain the description of the existing tables. The files have predefined names (chosen by programmers) and their location is known. This information is not requested from users.
- Binary files containing data from tables. Each table has an associated binary file. Creating a table leads to creating the associated file. Deleting the table leads to deleting the file. This information is not requested from users.

- CSV files containing data that can be imported into tables (an alternative to an INSERT script).

Specific requirements

- A module (one or more classes) is implemented through which the application receives one or more input files, of type text, through the arguments of the main function

Example: if the application is called projectPOO.exe then it is launched with the command

projectPOO.exe commands.txt [commands2.txt];

Possible scenario: The application receives a text file with commands that define the tables and their index through CREATE commands. A second file would contain INSERT commands.

- Only text files can be used as arguments for the main () function. Up to 5.
- The application can be started without receiving arguments through the main function (it runs with projectPOO.exe).
- The description of the tables (column names, their type, etc.) existing in the system will be stored in configuration text files associated with these structures (the file name can correspond to the table name). The configuration file or files are known to the programmer and are automatically accessed by the application at startup.

Recommendation: using binary or text files the application loads the description of the tables in the system. At startup, the application knows what tables exist based on these configuration files.

- All data inserted by users in tables via INSERT commands will be stored in binary files.
- Each table in the system will have a binary file associated with the table name and an extension of your choice.
- A module (one or more classes) is implemented through which the application data are saved in binary files. The application data is considered to be that data obtained from the text files received as arguments for main or data uploaded by users during the work session through INSERT or IMPORT commands.

Recommendation: To manage different data types, it is recommended to store the data in the form of byte arrays. When reading them, the description of the table will be used to convert them to the associated primitive types (string, numeric, etc.).

- A module is implemented through which the execution of SELECT or DISPLAY TABLE commands automatically generates reports for the displayed data (eg data list from a specific table, etc.) at the console but also in text files (text file names are generated automatically).

For example: for a SELECT command the text file SELECT_1.txt, containing the result of that command, will be generated. For the next SELECT command. It is generated SELECT_2.txt The results displayed by commands on the screen will also be saved in these files.

- The application allows the upload of data from CSV files (comma separated values); the symbol chosen to separate the values (,; # | or other) is chosen by the programmer and is used for all CSV files of the application. Importing CSV files will be done by adding the next command

IMPORT tablename file_name.csv

- If the CSV file does not contain the expected separator or the structure associated with the table then the command will display an error message indicating the required separator and table structure
- For the IMPORT command the two arguments are mandatory
- Phase 2 is considered completed if at least 75% of the requirements are implemented

Technical requirements:

- Minimum 3 new classes containing has-a relations with other classes
- Adding in the existing classes or in the new classes at least 2 arrays of objects. At least one of the vectors is dynamic.
- Entities that manage files and file operations are implemented by classes (source code through which files are accessed must be included in methods belonging to new or old classes)

Bonus (optional requirement):

- Implementation of the index for the indicated table (see CREATE INDEX command) The
- The index must allow quick search of data in response to SELECT commands with a WHERE clause in which the condition is the index.
- The index will define a binary file associated with it in which you store the value of the index column and the corresponding record position (offset) in the binary file associated with that table.
- For SELECT commands with a WHERE clause based on the index column, the search will be done in the index and then the related records are extracted directly (without index, the search for records will be done sequentially in the binary file)

Next in Phase 3

- refactoring by using type relations is between classes the
- use of collections of STL type and the addition of new functions the

- implementation of a module to manage a simple form of menu through which the user can navigate the functions of the application (after starting the application, the user has the ability to use numeric keys or enter text to decide which functions to activate - e.g. to see the invoices in the system, to generate different types of reports, etc.)
- Making a 5-minute video in which the team will present a live demo of the application

Phase 3 - Wrapping up

- refactoring by using is-a relations between classes
- Optional Making a 5-minute video in which the team will present a live demo of the application

Presentation of project

- Depending on the requests of the seminar coordinator - Making a 5-minute video (maximum 5 minutes) in which the team will present a live demo of the application that will focus only on the functions of the application (source code is NOT presented). The registration shows only the use of the application by exemplifying the introduction of different commands and the obtained results (on the screen or in the data files)

Specific requirements

- All requirements requested in the previous phases must be functional
- The application must run “out of the box” without generating execution errors
- Optional - implementation of a module to manage a simple form of a console menu (using numeric keys) through which the user can select a series of predefined reports (eg list of tables in the system or records in certain predefined tables). The menu can be activated automatically when the application is started (after uploading the received files from the command line) or is activated by a specific command (ex MENU). The menu must contain an option that allows the user to return to the command entry mode.

Technical requirements

- Implementation / Modification of a class to exemplify the concept of class composition through has-a relations (if it has not already been implemented in the previous phases)
- The extension of classes defined in the previous phases by adding new attributes (necessary in this phase) is done only by derivation and NOT by modifying the existing classes - minimum 2 extensions
- Implement at least one abstract class (with or without attributes) that must contain at least 2 pure virtual methods
- Add to an existing class or a new class at least 2 virtual methods (other than pure virtual methods) that are overridden in derived classes

- By using an abstract class or a basic class (newly defined or existing one) define at least one hierarchy to describe a family of classes (eg for types of expenses, etc.); an example of a family of classes defined to describe geometric shapes
- Implement at least on vector of pointers to a base class type (the parent of the hierarchy) and use it to manage real objects from the hierarchy (a hierarchy/framework of classes is given by a set of classes that are in a is-a relation between them and have a common parent); an example of a family of classes defined to describe geometric shapes (for this example the pointers in the array will have the Shape type)
- Optional Implement at least one vector, set, list, and map collection from the STL library to manage application data (collections are used either in existing classes or in new ones added at this stage). You can refactor an existing class by replacing common arrays with STL collections.