The DS-OOP Project

A Chance to Put Everything Together …

In this project, you and a partner will write a *database application* that keeps track of something useful. You do not have to interface with or store your "*something usefuls*" in a true database, but you will be storing your items in a file. This will require you to be able to read your items from the database, write your items to the database and also store (contain) your items in memory while your program is executing…

Mandatory Functionality

Your final project must support the functionality listed below.

- add item
    - if you have more than a few data fields, keep in mind the user experience (i.e. don't make entering information difficult or arduous or your mark will suffer)
- delete item
- delete entire database
- find and display item in a formatted and readable fashion
    - this should be case-insensitive, unless it wouldn't make sense for the data
- display entire database in a formatted and readable fashion, sorted by whatever makes sense for your data
- load database from file automatically (if it exists) upon program startup
    - the data file must be stored in the "current directory"
    - the data file must have a filename starting with one of your first names, followed by the first three letters of that person's last name (to not overlap with other projects)
- add an arbitrary number of random items to the database
    - what this means is that you should have a menu item that, when chosen, prompts the user for a number and creates that many fake data items, inserted into the database
        - your random data must not cause your program to misbehave (e.g. a month of 93)
    - random strings can be made up of random alphabetic characters if you wish
    - this is obviously used for testing
- exit, automatically saving database changes

Project Proposal

You must provide a brief outline of your proposal for the project (due on Tuesday March 24, 2015). This proposal is not an SRS and does not have to follow that format. It should simply be called a *Project Proposal* outline and be submitted into the ***DS/OOP Project Proposals*** DS eConestoga dropbox. It typically would not have to be longer than a paragraph or two. If your outline is not submitted on time, you will lose 3 marks per day it is late, to a maximum of 30 marks. If you do not submit an outline by 10 days after it is due, you cannot submit the project.

- You must also tell me who you are partnered with (full names, please).
- You do not necessarily have to follow the proposal in the final product, although you should let us know if there will be a significant deviation. The main objective for this is to get you thinking about the project NOW.

The DS-OOP Project

A Chance to Put Everything Together …

General Programming Requirements

- Follow the normal style requirements as expected in the OOP class.
- You will be using *DOxygen-Style* commenting for your:
  - Class Header comments
  - Method and Function Header comments
- Please include your project description from your Project Proposal as the *Main Page* contents of your DOxygen Comments
- Don't forget to also comment the body of your methods according to the guidelines discussed in OOP.
- Do not use global variables.
- Your program must not crash or hang.
- Communicate clearly with the user when you want them to do something.
  - Do not expect the user to "be a mind reader" with respect to input format or content.
  - Communicate clearly with the user regarding any special formatting of input that is required (e.g. dates).
- Handle invalid input appropriately - including letting the user know that the input is invalid.
  - You can assume that the user will never enter more than 255 characters at any one time.
- Your program must not allow invalid data to be stored within the database.
  - If you are working with dates and you do not have need for a wider range of dates, assume that all dates will be in the range Jan. 1, 1980 to Dec. 31, 2099.
    - If your data deals with other dates outside that range (e.g. climate change temperature data), feel free to have your code handle that with another appropriate range.
- Make sure that any output that you display is formatted properly and that all output is readable (e.g. doesn't scroll off the screen).
- The project must be done using C++.
- Call the source file that your main() is in project.cpp (you can call other files whatever you want). Call the executable project.exe
- Follow best design principles in separating your program's design into View, Data-Access and Domain layers

Course Specific Requirements

If you are taking the OOP course, you must:

- design the program using classes that you have created
  - you must have at least two non-trivial classes
  - the classes you create must have appropriate data members and methods
  - you must use appropriate encapsulation
- if you use dynamic memory allocation, you must use new, not the malloc()-family
  - if you are taking the DS course, there ***must not be*** a fixed limit on the number of items stored in your database
  - if you are **not** taking the DS course, you can use an array (minimum of 100 elements upon startup) of pointers to data elements (where the pointers are obtained by new when required) if you wish
- when using library functions, etc., use namespaces wherever possible
- use exceptions for error handling where appropriate

The DS-OOP Project

A Chance to Put Everything Together …

If you are taking the DS course, you must meet the following requirements:

- storage of the database must be done by using one (or more) of the data structures or STL containers that you have learned about in the course
  - your choice of data type should be appropriate for the data that you are keeping track of

General Quality Requirements

- Your program must compile without warnings. It is acceptable to use the #pragma method described in the C course notes to turn off deprecation warnings.
- The output of your program must not have typos, grammatical errors, or misspellings.
- The output of your program must not display output that is more for the developer than for the end user (e.g. debugging output). The decision of the marker (as an end user) is not up for negotiation.
- You must do **unit testing** on at least three distinct *parts* of your program.
  - A testable *part* might include:
    - an individual function or method
    - a set of functions or methods used in conjunction with each other to carry out some feature or function of your program
  - In order to demonstrate your unit testing, you must hand in three *Unit Test Driver* files (similar to that done for C++ Assignment #1).
  - Each unit test driver file must:
    - be called unitTestDriver#.cpp, where # represents the numbers 1, 2, or 3
    - have a file header comment that accurately describes of what part of your program is being tested - (i.e. describe what the ***unit*** is that is being tested)
    - much like you did in C++ Assignment #1, it is expected that in each unitTestDriver##.cpp source module, you write :
      - **at least** three *normal tests*, three *exception tests* and three *boundary tests* (where applicable)
      - have comments in the code (before each test case) that indicate what specifically is being tested, the type of test and what the appropriate results are

Group Requirements

- This project must be done in groups of two, unless there are an odd number of students.
  - If you do not have a partner, you can post in the OOP/DS Project *Looking for a Partner* forum in the DS eConestoga space.
  - If you do not have a teammate by Tuesday March 24, 2015 and there are others without teammates, you will be assigned one. You don't want this. Really.
    - If you have a partner but haven't agreed on a topic by Tuesday March 24, submit something indicating that, just so you won't be assigned a partner. You will still need to resubmit your topic, though.
    - If you know that you are not intending to do the project and do not want to be in a group, submit something saying this to the *DS/OOP Project Proposals* DS eConestoga dropbox by March 24. It is not fair to others if you don't intend to complete the project but you end up being assigned to someone.
    - If you were assigned a partner, you will be sent an e-mail (to your Conestoga e-mail account) that you must acknowledge (to Carlo, Sean, and your group partner) within

three days of it being sent. If you do not acknowledge it on time, you might lose marks or lose the right to submit the project at all.

  - You must mention who is in your group in the document that you submit on March 24. If you don't, you could lose 5 marks.
- It is your responsibility to exchange complete contact information with your teammate.
- If you do not intend to fulfill your responsibilities in the project, let your teammate, Carlo, and Sean know.
- If your teammate doesn't pull their weight in the project, you are still responsible for completing the project. When you submit the project, you can describe how there was an imbalance in the work done (in the Submission text box) and there will be a subsequent discussion that could change the proportion of the marks.
- The project is only submitted by one member of the team.
  - If both members of the team submit the project, both will be marked (hard!) and the lower mark of the two will count.

Progress Requirements

- In an arbitrary (not to be announced) OOP or DS class on March 31 or later, you will be expected to show what work your group has put towards the project. It is expected that you will be at least 25% done at that point (although you should be further than that). If you are not close to that or if you do not show the work for whatever reason (including simply not having it with you), you could lose marks.
  - It is entirely possible that this could be repeated later in the month with a higher expectation of completeness.

Submittal Requirements

- This project has many components that are required for submission – you need to submit your source code, your Visual Studio solution files, the executable that you've built and tested from your code and your DOxygen comments.
  - So let's pretend that you (John Smith) and your partner (Alice Kramden) have decided to design and develop your DS/OOP project to keep track of Pokemon™. In this case, you might have created and named your project as **MyPokedex**.
  - The resultant Visual Studio directory structure would look like:

```
C:\MyPokedex                (this directory is the name of your VS Solution)
 ↳this directory contains MyPokedex.sln, MyPokedex.suo – the VS Solution files
 ↳MyPokedex                 (this directory is the name of your VS Project)
        ↳this directory contains your source files (.h, .cpp)
 ↳Debug
        ↳this directory will contain your executable if you compile in DEBUG mode
                ↳MyPokedex.exe
 ↳Release
        ↳this directory will contain your executable if you compile in RELEASE mode
                ↳MyPokedex.exe
 ↳DOXygen
        ↳I suggest you make a directory in your main solution directory structure to be the
         place where your DOxygen comments get built into.  This directory will contain a
         subdirectory called "html"
```

    - In the case of this example, I would simply ZIP up my C:\MyPokedex directory (making sure to include and maintain subdirectories) and produce a file called MyPokedex.zip
    - Please rename the ZIP file to indicate your name and your partner's name – so I would rename the ZIP file to be smithJ-kramdenA.zip
- Submit your ZIP file to the *DS/OOP Project Solutions* eConestoga DS dropbox.
- Pinky Bucks are not usable.

Optimizing your Mark

- In order to optimize your mark, don't forget that you have at least **two** very distinct users of your program with very different requirements:
    - User #1: Whoever would be using your program in real life. The main focus of their requirements is to make sure that the program does its job well, intuitively, and easily.
    - User #2: Whoever is marking your program. There are two main requirements for this user:
        - #1: Ease of evaluation of how well you applied the knowledge you learned during the year.
        - #2: Ease of marking, so that marking your project isn't an enormous pain and doesn't take significantly longer than other projects.
- As has been stated to you, the best way to *add value* to any program or system you are designing and implementing is to **think like the user**.

Alternative:

If you wish to do another type of project for the courses, please see Carlo or Sean. It will be expected that you produce a description of the project's programming requirements (like this one) by March 24, 2015.

Information:

- Any further information about the project will be contained in the DS eConestoga course material.