COP3014: Project 2 – Contact App

Overview

Software development projects using the object-oriented approach involve breaking the problem down into multiple classes that can be tied together into a single solution. In this project, you are given the task of creating a program to manage contacts. Specifically, you are asked to identify and implement at least two classes that would work together to make the Contacts application.

Learning Objectives

The focus of this assignment is on the following learning objectives:

- Be able to identify the contents of class declaration header and class definition files and to segregate class functionality based on class description
- Be able to write object creation and initialization code in the form of constructors
- Be able to implement an encapsulated approach while dealing with data members for the classes involved through accessors and mutators (getters and setters)
- Be able to make the program code spread across multiple files work together for the solution to the computing problem
- Be able to design and implement classes that are appropriate for a particular application.

Prerequisites

- To complete this project, you need to make sure that you have the following:
- C++ and the g++ compiler
- A C++ IDE or text editor (multiple editors exist for developers)
- An understanding of the material presented in class
- An understanding of the material covered in the zyBook.

Problem Description

In this project, you are asked to create a program to maintain contacts. We would like to store name, email, and phone number in a Contact. The maximum number of contacts that your program can store should be passed as a parameter to the appropriate class. See below.

The following are the operations that your program should provide to its user.

1. Add a new contact
2. Remove a contact
3. Edit a contact
4. Search for a contact
5. Display all contacts

Example Run

Here is an example run of the program.

```
---Main menu---
   1. Add a new contact
   2. Remove a contact
```

```
   3. Edit a contact
   4. Search for a contact
   5. Display all contacts
   0. Exit
$ Enter choice: 1
$ Enter name: Jane
$ Enter email: jane@company.com
$ Enter phone #: 1234567890

---Main menu---
   1. Add a new contact
   2. Remove a contact
   3. Edit a contact
   4. Search for a contact
   5. Display all contacts
   0. Exit
$ Enter choice: 1
$ Enter name: Tom
$ Enter email: tom@company.com
$ Enter phone #: 9876543210

---Main menu---
   1. Add a new contact
   2. Remove a contact
   3. Edit a contact
   4. Search for a contact
   5. Display all contacts
   0. Exit

$ Enter choice: 3
$ We currently have the following names:
[1. Jane, 2. Tom]
$ Which contact do you want to edit? 1
$ What do you want to edit?
 [1. Name, 2. email, 3. phone #]
$ Enter choice: 1
$ Enter new name: Maria
Contact edited successfully

---Main menu---
   1. Add a new contact
   2. Remove a contact
   3. Edit a contact
   4. Search for a contact
   5. Display all contacts
   0. Exit

$ Enter choice: 5
[Maria, jane@company.com, 1234567890]
[Tom, tom@company.com, 9876543210]
```

```
---Main menu---
   1. Add a new contact
   2. Remove a contact
   3. Edit a contact
   4. Search for a contact
   5. Display all contacts
   0. Exit

$ Enter choice: 4
$ Enter name to search for? Maria
[Maria, jane@company.com, 1234567890]

---Main menu---
   1. Add a new contact
   2. Remove a contact
   3. Edit a contact
   4. Search for a contact
   5. Display all contacts
   0. Exit

$ Enter choice: 2
$ We currently have the following names:
[1. Maria, 2. Tom]
$ Which contact do you want to remove? 2
Contact removed successfully!
---Main menu---
   1. Add a new contact
   2. Remove a contact
   3. Edit a contact
   4. Search for a contact
   5. Display all contacts
   1. Exit
$ Enter choice: 0
$
```

Implementation Notes:

- For this project, you are required to create a **Contact** class to model a single Contact, **ContactManager** class to store up to a maximum number of contacts. The maximum number of contacts can be passed as a parameter to the constructor of the **ContactManger** class.

- To allow **ContactManager** class to store up to a given number of Contacts, you are required to use an array of contacts (with the given size) as a private member of **ContactManager**. **The array needs be dynamically allocated.** Also, you may want to use a member variable that indicates how many valid contacts the **ContactManager** currently has. This variable can be initialized with zero and incremented every time a new contact is added. Similarly, the variable is decremented every time a contact is removed.

- If a contact is removed from the middle of the array, you can shift the array elements to the left by one position to maintain contagious positioning of valid contacts.

- Prepare a makefile that compiles and runs your project. The program should be tested on the UWF SSH server before submission and should be submitted with a ***makefile***. The instructor will test the program files by only running 'make' and 'make clean'. Please make sure that the makefile also contains the name of the target executable file and the command to run the target. Entering the 'make' command on the command prompt should run the target executable file.

Development Diary

For this project, you must complete the following tasks in the order that they are described:

1. Write down the list of all classes that you need to implement. For each class, list the associated member variables and the set of member functions.
2. Create a plan to implement your solution to the problem with a timeline. Each step in your plan refers to an increment of the program that you will be creating. It is recommended to complete the implementation of a single logical action per step.

   Your responses to these tasks should be submitted as a PDF document called **lastname_firstname_project03.pdf.**

Grading Breakdown

- **[25 pts]** fully defined and implemented the **Contact** class
- **[50 pts]** fully defined and implemented the **ContactsManager** class as well as **ensuring proper allocation and freeing of dynamic memory.**
- **[10 pts]** proper handling of user input and displaying results.
- **[10 pts]** Clear, easy-to-read code (e.g. class declarations in .h files, class definitions in .cpp files, working **makefile**, etc.)
- **[5 pts]** Development "diary" (report) that details design and implementation of each class, as well as the project as a whole. Should include a reflection of the process as a whole upon completion of the project (identify the challenges, lessons learned, ways to improve in the future, etc.)

Submission

Points will be deducted for not following these instructions.

Before submitting this project in eLearning make sure that you follow the following steps:

1. Make sure that your name appears at the top of each file. This should be done as a comment for any source code files.
2. Put your development diary document (.pdf), source code (.hpp & .cpp), and working makefile into a .zip file with the file name "lastname_firstname_project03.zip".
3. Turn your **zipped up** project into eLearning/Canvas.