SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

ENSC 251 D100 – Software Design and Analysis for Engineers (4 sem. hrs.)
Summer 2018

Lab 3

Assigned	the week of May 28, 2018
Due	Sat June 2, 2018 @ 9:00am.

This is an individual assignment.

- You may consult with professor and TA about any aspect of the assignment.
- You may consult with other students only in a general way, e.g., about debugging or C++ issues, or questions about wording on the assignment.
- You cannot actively work with another student in this assignment.

General Information

- Use the header format. Replace text in GREEN with the appropriate information.

```
/**
 * @file XX.cpp
 * @author FIRSTNAME LASTNAME
 * @Date   DATE
 * @version 1.0
 * @section DESCRIPTION
 */
// I declare that this assignment is my own work and that I have correctly acknowledged the
// work of others.  I acknowledged that I have read and followed the Academic Honesty and
// Integrity related policies as outlined in the syllabus.
//
// _____ (PRINT YOUR NAME HERE) ____      _____(DATE)_______________
//
// ____ (STUDENT ID) ______
//
```

1. Specifications

PART A – Patient Charges

Write a class named `Patient` that has member variables for the following data:

- First name, middle name, last name
- Address, city, state, and ZIP code
- Phone number
- Name and phone number of emergency contact

The `Patient` class should have a constructor that accepts an argument for each member variable. The `Patient` class should also have accessor and mutator functions for each member variable.

Next, write a class named `Procedure` that represents a medical procedure that has been performed on a patient. The `Procedure` class should have member variables for the following data:

- Name of the procedure
- Date of the procedure
- Name of the practitioner who performed the procedure
- Charges for the procedure

The `Procedure` class should have a constructor that accepts an argument for each member variable. The `Procedure` class should also have accessor and mutator functions for each member variable.

Next, write a program that creates an instance of the `Patient` class, initialized with sample data. Then, create three instances of the `Procedure` class, initialized with the following data:

Procedure #1	Procedure #2	Procedure #3
Procedure name: Physical Exam Date: Today's date Practitioner: Dr. Irvine Charge: $250.00	Procedure name: X-ray Date: Today's date Practitioner: Dr. Jamison Charge: $500.00	Procedure name: Blood test Date: Today's date Practitioner: Dr. Smith Charge: $200.00

The program should display the patient's information, information about all three of the procedures, and the total charges of the three procedures.

PART B
Car Class

Write a class named `Car` that has the following member variables:

- `yearModel`—an `int` that holds the car's year model
- `make`—a string that holds the make of the car
- `speed`—an `int` that holds the car's current speed

In addition, the class should have the following constructor and other member functions:

- Constructor—The constructor should accept the car's year model and make as arguments. These values should be assigned to the object's `yearModel` and `make` member variables. The constructor should also assign 0 to the `speed` member variables.
- Accessor—appropriate accessor functions to get the values stored in an object's `yearModel`, `make`, and `speed` member variables
- `accelerate`—The `accelerate` function should add 5 to the `speed` member variable each time it is called.

- `brake`—The `brake` function should subtract 5 from the `speed` member variable each time it is called.

Demonstrate the class in a program that creates a `Car` object, then calls the `accelerate` function five times. After each call to the `accelerate` function, get the current speed of the car and display it. Then, call the `brake` function five times. After each call to the `brake` function, get the current speed of the car and display it.

2. Submission Instructions

You can use the example zip file from lab 1 as a starting point. Create *.cpp file as needed. Modify the makefile such that it will compile your code into a binaries executable.

a) Create a directory with your name, e.g. "\LastnameFirstname", where Lastname is student's last name and Firstname is the first name.
b) Save the files (*.cpp, other files, and makefile) in this directory. Uses these files as a starting point to write the following program.

For example, for student Mary Smith, this will be the directory and files

```
\SmithMary
      \partA
           \*.cpp   [e.g. Patient.cpp]
           \makefile
      \partB
           \*.cpp   [e.g. Car.cpp]
           \makefile
```

Then Zip up the directory "\LastnameFirstname" and the files within this director into a zip file "2018-2-ENSC251-LastnameFirstname.zip." Submit the zip file to Canvas before the deadline.

3. Resources

- C++ Formatter https://codebeautify.org/cpp-formatter-beautifier
- Vim Basics - https://www.howtoforge.com/vim-basics
- Common Linux Commands http://www.dummies.com/computers/operating-systems/linux/common-linux-commands/

4. Rubric for marking

PART A (50%) & PART B (50%)

Criteria	Ratings				Pts
Program Specifications / Correctness	**Excellent** - No errors, program always works correctly and meets the specification(s). 50.0 pts	**Adequate** - Minor details of the program specification are violated, program functions incorrectly for some inputs.	**Poor** - Significant details of the specification are violated, program often exhibits incorrect behavior.	**Not met** - Program only functions correctly in very limited cases or not at all. 0.0 pts	50

		40.0 pts	30.0 pts		
Readability	**Excellent -** No errors, code is clean, understandable, and well-organized. 20.0 pts	**Adequate -** Minor issues with consistent indentation, use of whitespace, variable naming, or general organization. 16.0 pts	**Poor -** At least one major issue with indentation, whitespace, variable names, or organization. 12.0 pts	**Not met -** Major problems with at three or four of the readability subcategories. 0.0 pts	20
Documentation	**Excellent -** No errors, code is well-commented. 20.0 pts	**Adequate -** One or two places that could benefit from comments are missing them or the code is overly commented. 16.0 pts	**Poor -** File header missing, complicated lines or sections of code uncommented or lacking meaningful comments. 12.0 pts	**Not met -** No file header or comments present. 0.0 pts	20
Code Efficiency	**Excellent -** No errors, code uses the best approach in every case. 5.0 pts	**Poor -** Code uses poorly-chosen approaches in at least one place. 3.0 pts	**Not met -** Many things in the code could have been accomplished in an easier, faster, or otherwise better fashion 0.0 pts		5
Assignment Specifications	No errors 5.0 pts		Minor details of the assignment specification are violated, such as files named incorrectly or extra instructions slightly misunderstood 3.0 pts	Significant details of the specification are violated, such as extra instructions ignored or entirely misunderstood 0.0 pts	5
				Total	100

-END-