Program:	**CPA2/3**
Course:	**INFO1156 "Object Oriented Programming in C++", W21**
Professors:	**Michael Feeney**
Project # 2:	**Cities connected by roads.**
Weight:	**10% of your final mark**
Due Date:	~~**Friday, February 12th, @ 11:59 PM**~~
	UPDATE: Sunday, February 14th, @ 11:59 PM

Description and Purpose

With all the scrutiny about facebook, you've decided to make your own.

A "mug" is US/Canadian slang for someone's face (comes from a police "mug shot", I'm guessing).

So your "facebook" replacement will be called "MugBook". Get it? #hilarious

But you need to start with 'baby steps' and just get the "friend" feature working first.

You will make a C/C++ console based "Win32" application that allows you to add users and their friends.

You will submit:

Your (compressed) amazing **Visual Studio solution + accompanying files** to the FOL drop box.

~~There's seven (7) parts (questions?) to this project, with the marks indicated.~~

The "helper program" is in the **`INFO1156NameGenerator.7z`** solution + file.

Be sure to read the "**Submission Rules and Restrictions**" to *avoid getting a mark of zero*.

Details

Mike Acton lists "3 big lies" of software development (https://cellperformance.beyond3d.com/articles/2008/03/three-big-lies.html):

- Software is the platform.
- Code should be designed around a model of the world.
- Code is more important than data.

Read them again, remembering that they are "Big **Lies**".

The key one here is #3: the **only** reason you write a program is to manipulate data.

This is **ALWAYS** the case. You should always be suspicious of people who suggest otherwise.

#1 is also key here because we're using a "system language" (C/C++). The truth is "the *platform"* is the platform. i.e. your program runs directly on the hardware of the computer, not being parsed, running in a virtual machine, or anything else.

You must write your code to comply with the API listed below.

If you change this, you "break" the API, then the code will not compile or run. And you will get a mark of zero.

(if you think this is "unfair", imagine how "changing/re-writing" the API would be handled in the workplace. You may not "agree" with the API, but you can't change it. As a programmer, you will always have to work with existing code and APIs. Always.)

NOTE: You can ADD stuff to the classes as you see fit, but you have to assume that your code is working within a larger code base. For example, if you add "getters/setters", that's "fine" *but anyone else's code isn't calling those, right?* Anything "private" is implementation details, so is ignored (and is inaccessible, too).

You must implement the following functions, using the data structures listed below. (these are also in the INFO1156_P1_API.7z file)

cPerson **MUST** be in a file called cPerson.h (class definition) and cPerson.cpp (implementation).

cMugBook **MUST** be in file called cMugBook.h (class definition). You can use any .cpp (implementation) files you want *except* the file that contains your main() function.

UPDATED:

cPersonGenerator **MUST** be in a file called cPersonGenerator.h (class definition). You can use any .cpp (implementation) files you want *except* the file that contains your main() function.

The cPersonGenerator is now completely OPTIONAL.

There is now one provided for you that you can use (in the github repository).

You don't even need to submit it, or add it to your project.

However, if you did it already, and want to submit it, then it's a bonus! Hazzah!

You will get a bonus of 10% if you successfully implement and demonstrate the cPersonGenerator.

(Note: you can't just use mine. It has to be "significantly different")

I will be using some basic marking code that will "exercise" your code.

```
#include <string>

class cPerson
{
public:
	// Constructors and Destructors are ALWAYS present (created)
	// If you list them here, then you are telling the compiler that
	//     you are WRITING YOUR OWN.
	cPerson();			// constructor (c'tor)
	~cPerson();			// destructor (d'tor)

	std::string first;
	std::string middle;
	std::string last;

	// enum inside the class "tightly coupled"
	enum eGenderType
	{
		MALE = 0,
		FEMALE,
		NON_BINARY,
		RATHER_NOT_SAY_UNKNOWN
	};

	eGenderType gender;
	std::string getGenderAsString(void);

	int age;			// age is in years. default = -1 (which is invalid);

	//  - streetName = "MISSION BAY"
	//  - streetType = "BLVD"
	//  - streetDirection = "NORTH"		Note: Can be blank
	//
	int streetNumber;			// default = -1			(which is invalid)
	std::string streetName;			// default for these is blank (i.e. "")
	std::string streetType;
	std::string streetDirection;
	std::string city;
	std::string province;
	// Canadian postal codes are 6 characters
	// They are: leter number letter space number letter number
	// All capitols
	char postalCode[7];

	// Social insurance numbers are 9 digits
	unsigned int SIN;	// = 0
	//unsigned int SIN = 0;			// C++ 11

private:
};
```

This is now an OPTIONAL BONUS
(see page 3 for more details)

```
#include "cPerson.h"

class cPersonGenerator
{
public:
	// Returns false if there's an issue loading this file
	bool loadPeople( std::string dataFileName );

	// Picks a "random" person from the list of people loaded.
	// UPDATE: This can return the same person if called enough times.
	//		Example: loadPeople() loads 25 people, but you call
	//		pickAPerson() 100 times will *definitely* return “the same”
	//		person. Note this isn’t the point: you’d load way more people
	//		than you’d pick. Just don’t bother double checking if you’ve
	//		*already* returned that particular person.
	cPerson pickAPerson(void);

private:
	// Whatever containers, etc. you want should go here...

};

class cMugBook
{
public:
	// You MUST implement a constructor (c'tor) and destructor (d'tor)
	// EVEN IF IT DOESN'T ACTUALLY DO ANYTHING
	cMugBook();
	~cMugBook();
	bool addUser( cPerson thePerson );
	bool deleteUser( unsigned int SIN );

	// This will only update the data that is DIFFERENT **AND** VALID
	bool updateUser(cPerson thePersonWithNewInfo);
	bool updateUserLastName( unsigned int SIN, std::string newLastName );
	bool updateUserFirstName( unsigned int SIN, std::string newFirstName );
	bool updateUserAddress( unsigned int SIN, int streetNumber, std::string streetName,
	                        std::string streetType, std::string streetDirection,
	                        std::string city, std::string province,
	                        std::string postalCode /*NOTE: This is different*/);
	bool updateUserGender( unsigned int SIN, cPerson::eGenderType newGenderExpression );

	// Returns true if the person is found, false if not (then "theUser" is invlaid)
	// Note tha the user is "returned" by reference.
	bool getUser( unsigned int SIN, cPerson &theUser );

	// Returns true if it's OK.
	// Returns false if:
	//     * user doesn't exist
	//     * friend doesn't exist (or is invalid)
	//     * user and friend are the same person
	bool addFriend( unsigned int UserSIN, cPerson theFriend );
	bool unFriend( unsigned int UserSIN, cPerson theFormerFriend );

	// Returns false if user doesn't exist
	// vecFreindList can be zero (which is sad, but valid)
	bool getFriendList( unsigned int UsersSIN, std::vector<cPerson> &vecFriendList );

};
```

What it's supposed to actually do:

cPersonGenerator:

- Will load a text file with a list of users that came from the INFO1156NameGenerator program.
- INFO1156NameGenerator:
    - You have to compile this.
    - It needs the datfile.dat file to be in the "working directory" to operate.
    - As command line parameters, you add your student number and how many people you want to add, for example "INFO1156NameGenerator.exe 2736379 100" will generate 100 people.
    - Your student number is used as a random number seed.
    - I will use a file that has 1,000 users for testing. I won't be using your student number, tho.
    - The format of this file is below, but it's a "comma delimited" file where each line is one person.
    - Expressed gender is M (male), F (female), and N (non-binary).
- You will have to "process" the file a little bit:
    - Make sure certain things are valid: address, SIN, postal code, etc.
    - If they're ***anything*** invalid, you are to ignore them (**NOT** load that person).
    - All text has the 1st letter as a capitol letter, then lower case:
        - So "AMY SMITH" would have to be changed to "Amy Smith", "STREET" has to be changed to "Street", "LONDON" to "London", etc.
- pickAPerson() will return a "random" person from the list of people you loaded.
    - The point is that you can call pickAPerson() then pass *that* person into the cMugBook addUser() or addFriend() methods.

- **CLARIFICATION:** Your cPersonGenerator would read the ***output*** of the INFO1156NameGenerator program. It *doesn't* read the datfile.dat file – that is the ***input*** for the INFO1156NameGenerator.

cMugBook:

- The functions are pretty self explanatory.
- If you've never used "facebook" (or other social media), users have a list of "friends" (followers/whatever).
- Only other user can be friends. i.e. there should be no "friend" that is not a "user".
- The "update" methods need to check if the information is valid, so things like:
    - Postal codes OK
    - Street numbers OK
    - Any strings aren't blank ("")
- The functions ***shouldn't cataclysmically crash*** with simple "invalid" operations.
- It gives you more freedom and creativity with how you write your code (maybe?).
- Updating a user *should be reflected in the friend list.* So if a user changes their last name, returning the friend list should have that same change.
- Note that expressed genders are male, female, **and** non-binary.

Output of the program INFO1156NameGenerator program (i.e. file format):

- **CLARIFICATION:** This ***output*** file is the ***input*** of your cPersonGenerator class. i.e. the file that the "bool loadPeople(std::string dataFileName);" method loads and processes.

  It'll be your student number and "peopleList.txt" by default, so "17363_peopleList.txt" or whatever. You can rename this file whatever you'd like, though.

- 1st line is "Number of people in this file: XXX"
  - Where "XXX" is an unsigned int of how many people are listed
- People are listed one per line
  - First name (all capital letters, like "HARRY")
  - Middle name (all capital letters)
  - Last name (all capital letters, like "POTTER")
  - Gender ("M", "F", or "N")
  - Street number (an unsigned int, starting at 1)
  - Street name
  - Street type ("AVE", "STREET", "ST", etc.)
  - Street direction (optional, but "NORTH", etc. If not present, is blank)
  - Postal code (7 digits, "XXX XXX" Canadian Postal code rules)
  - City
  - Province
- All the test is UPPER CASE.
- While it's possible an individual person has invalid information, assume ***number of people*** is correct.
- **UPDATE:** This *doesn't* output an **age**, so please pick a "random reasonable adult human age" for each person, so >= 18 and <= 100. (Don't stress about that *precise* range, but if you return -7, 3, or 182,302 for the age, you'll lose marks.)

How you get your marks:

- It's basically a list of the methods, where things that are more difficult are worth more.
- cMugBook has 11 methods + 2 (c'tor and d'tor), so it'll be broken into 11 categories.
- Things like "AddUser()" is more trivial, so would be worth, say 10 marks.
- Things like getFriendList() or updateUserAddress() are much more involved, and so would be worth, say 50 marks. Why? Because they would take you 5x (approximately) the effort.
- If your method crashes, then you're likely going to get a mark of zero on that.
- Even if the method is more "trivial", they can still be crucial: like if I can't call "AddUser()", then I can't really call getFriendList(), right?
- But Feeney, how on Earth am *I* supposed to *know* what's harder than other things?
    - Well, how long did it take you to get it to work properly?
    - Relatively to that, that's what it's going to be worth.
    - Note I'm mentioning "relative to how long" *you* took, not anyone else.
        - So if it took *you* "10 hours" to make "addUser()", but it would take a professional programmer 10 minutes, then that doesn't mean you get "10 hours worth of marks" so to speak. (It really means something completely different if you think about it...)

          i.e. "how long it took *you*" isn't really a valid argument.

- It MUST build and run, or you get a mark of zero. Period.

  (How do you know this? Seriously? How do you think?)

  (Also, if it "runs on your machine" and doesn't run on one of my many, many machines, then it's your computer. But seriously, that's *never* the issue.)

Bonuses: 💔 **UPDATE** 💔

- (2.5%) Simple valid SIN validation: ***doesn't*** start with 0 or 8 (https://en.wikipedia.org/wiki/Social_Insurance_Number)
- (5%) Check that the provinces are valid (It's got to work with the output of the INFO1156NameGenerator, which could output any of the following: Alberta, British Columbia, Manitoba, New Brunswick, Newfoundland and Labrador, Northwest Territories, Nova Scotia, Nunavut, Ontario, Prince Edward Island. Quebec, Saskatchewan, or Yukon.)
- (5%) 1st letter of postal code matches the province (https://en.wikipedia.org/wiki/Postal_codes_in_Canada)
- (5%) Complete validation for SIN (i.e. is it a valid SIN?) (https://en.wikipedia.org/wiki/Social_Insurance_Number)

A "bonus" is added directly on top of your mark, so you can get > 100% on the project, or even the course. While they bonuses are "separate" (like you don't have to get 100% to be "eligible" for a bonus), they are a little bit at my discretion: Like if you only did these, but ignored validation for trivial stuff like "missing first name", then I might not give you the bonus. The ideas is that in addition to all the other validation, you also did these extra things.

Submission Rules and Restrictions:

Things that will **get you zero** and are **non-negotiable**:

- Changing the API so it won't build. This includes:
    - The classes, methods, etc.
    - The "signature" of the methods/functions/whatever.
    - What the format of the files you have to input/output.
      (In this case, the output of the INFO1156NameGenerator program is what you have to "deal with", so do *not* change it "because reasons").
- Using the "`auto`" keyword (nor can you use a #define/typedef to circumvent this).
- Using or including the boost library.
- Using "lamba" functions. You *may* use "predicate functions" or "functors".
- Using another external library that you didn't write. You may only use the STL (standard template library) libraries – i.e. the ones "built in" to the language.

Project Corrections

If any corrections or changes are necessary they will be posted to the course web site and you will be notified of any changes in class. It is your responsibility to check the site periodically for changes to the project. Additional resources relating to the project may also be posted.

75/10-year old “squinty eye” plagiarism test:

I have very little tolerance for plagiarism, but many students are unclear about what it is.

Basically, it’s submitting **somebody else’s work as your own**.

There is sometimes some confusion over this because you could argue nothing is actually “unique” (see: http://everythingisaremix.info/ for a fascinating overview of this).

The whole point of assignments/tests/projects in this course (or any course, really) is to try to see if you are actually able to ***do*** the coding that’s asked of you. In other words: How competent are you? Handing me someone else’s code and/or making a trivial change isn’t good enough.

Also, it’s illegal:

- http://www.plagiarism.org/ask-the-experts/faq/
- http://definitions.uslegal.com/p/plagiarism/
- http://en.wikipedia.org/wiki/Plagiarism
- https://www.legalzoom.com/articles/plagiarism-what-is-it-exactly

In other words, I’m not going to be drawn into a giant debate over how “different” your code is from mine or anyone else’s, if any sensible person (including me) would conclude that the code/application is pretty much the same thing, then it is. It is up to my discretion to decide this.

- While you may freely “borrow” mine (or anyone other) code ***but*** your code should be “sufficiently” different from mine (you might want to replace the word “sufficiently” with “significantly”).
- In other words, you *cannot* simply use an existing game engine (or part of a game engine) to complete this assignment; it should be either completely new of **significantly** modified.
- How will I determine this?

    - If I showed your application and/or your source code to either my pragmatic 83-year-old mother, or a typical 10-year-old, or even some random person walking down the hallway (i.e. a non-expert), and they looked at it, tilted their heads, squinted their eyes, and said “you know, they look the same,” then they ***are*** the same.

    - Another test would: How much time it would take for a "competent programmer" (for example, *me*) to make the changes you are submitting? The point here is that I don’t “care” if you tell me “But it took me *weeks* to make the changes!” Fine, but if I can make those same changes in 10 minutes, then not a lot of work has been done (certainly **not** sufficient work – these projects should show take **days** of work having been done).