Purpose of the Assignment

The general purpose of this assignment is to develop a simple web server in C++ using the Wt framework, given a number of requirements, making use of the principles and techniques discussed throughout the course. This assignment is designed to give you experience in:

- programming using C++, using basic language constructs, classes, and data types
- using the Wt framework for building a web server in C++
- creating graphical, user-facing web content as well as a more programmatical, RESTful web service

The assignment is intended to give you some freedom in design and programming the explore the subject matter, while still providing a solid foundation and preparation for the type of work you will later be doing in the group project.

Assignment Task

Your assignment task is to familiarize yourself with Wt and develop a simple web server in C++ that randomly generates address book information on a number of people and then serves this data programmatically in a way that is accessible to other web services. As detailed below, your server will have two main aspects: a user-facing interface to set options and generate data, and a RESTful API to serve this data in JSON format. In crafting your assignment, your program must adhere to a number of coding and style guidelines, as discussed below.

The Address Book

As noted above, your web server will ultimately be serving up address book information on a collection of people. This information is rather straightforward and it is somewhat up to you to determine how to represent and store things subject to a few constraints. Essentially, your program will need to maintain a list of records on people, with each record containing the persons name, their address, and a contact phone number. In particular:

- When I said "collection" and "list" above, I am not implying what data structure you should be using here. I will say that you will not know in advance how many people you are dealing with, as that is set in the user interface below, so you are not allow to use a statically sized structure. While you are free to use an array that you dynamically allocate, you will likely find it easier (and more beneficial) to use one of the other structures available in C++ for such purposes.
- You will need to define classes to represent the concepts of person, name, address, and phone number for this assignment and use objects from these classes in creating your address book data. In each case, you should have appropriate constructors, destructors, getters, and setters. In no case should an object directly access data members of objects from another class. (It's good form to have getters and setters for each data member even if you don't anticipate using them ...)
- A name will consist of two strings, one for the first name and one for the last name.
- An address consists of four strings, one for the street address, one for the city, one for the province, and one for the country. (To keep things simple, we'll assume the country is always "Canada" but it should be represented in each address anyways.)
- A phone number will consist of a valid-looking 10-digit phone number packed into a string in the format ###-###-####.
- You are free to include other data members on top of the above as necessary to support your program.

As noted above, the above data is to be randomly generated. Details on doing this are discussed below under the User Facing Interface, as that is where the generation process is triggered.

User Facing Interface

When a user connects to your server at its base URL (with no other pathing/files specified) using a web browser, the user should be presented with a simple interface for changing settings and managing the generation of address book data for the program. (This is similar to how other Wt examples work, so they provide a good guideline for how to do this.) Ultimately, the structure, layout, and visual appeal of this interface is up to you. Regardless of how you do things, your interface must include the following:

- The user must be able to specify the number of person-records to generate in the address book and then initiate the generation of the data. (For example, a simple text box and button should suffice for this.)
- The site must list the currently generated set of people, which will be helpful for debugging purposes. Initially, as there will be no people generated when your server first starts, there will be no information to be displayed to the user. (For example, this can be done using a table or even simple text.)

While your interface does not need to be completely bullet-proof, it should prevent most reasonable ways that the user might inadvertently break things. For example, a user should not be able to specify non-numeric text for the number of records and crash your program. For that matter, they should not be able to enter negative numbers or non-integers either. A value of 0 would be acceptable, however, to simply destroy the current list of people without generating a new one. You do not have to, however, put an upper bound on the number of people to avoid crashing the program, as this will depend on your configuration and a good number of other factors.

When the user specifies the number of people to generate and initiates the generation process, the following must occur:

- Any existing person records must be destroyed. Any memory attached to them must be appropriately freed up. Note that the container structure storing the records does not itself need to be destroyed, as you can reuse that below. Only the records need to be cleaned up.
- A new person object is created for each new record to generate, with the quantity specified by the user as noted above. Again, if the user has specified 0 people, no new person objects need to be created at this time. Each person is then added to whatever container structure you are using to manage the collection/list of people.
- Once all of the people have been generated, the server should be updated to reflect and list the new people that were just created so that the user knows the process has completed.

So, how exactly are person records randomly generated? You are to create a randomizer class for this purpose. This class will have a method for generating a name, a method for generating an address, and a method for generating a phone number. (It could have other methods as well, but it at least needs those methods.) Unlike your other classes which will require objects instantiated, this class could work as a static class without requiring instantiation prior to use. (This is up to you.) Each type of data will be generated as follows:

- To generate names, you will have a static, hard-coded list of strings that are first names, and a second such list containing last names. You should have at least ten of each. How you wish to set these up is up to you. Generating a name is then as simple as randomly selecting a first name and randomly selecting a last name, packing those into a name object and returning things from there.

- Generating an address is similar. You will have a static, hard-coded list of strings for each of street names, city names, and provinces. You should have at least ten street names, ten city names, and the ten Canadian provinces. To generate the street address, you simply select a random number as the street number and add a randomly selected street name to it, producing something like "123 Fake Street". The city name and province are selected randomly from your lists, and the country is always "Canada". This data is then packed into an address object and returned from there.
- Phone numbers are generated randomly. Ten random digits are picked and placed into the ###-###-#### format, though the first and fourth numbers should not be a 0. This is packed into a phone number object and returned from there.

REST Interface

In addition to displaying generated address book information in the User Facing Interface above, it will also be provided in JSON format through a simple RESTful web service interface. Constructing such a thing in Wt is fortunately not difficult. A tutorial with examples for doing so can be found here, with sample code illustrating how to start up your User Facing Interface along side this shown here. (In essence you are adding another entry point to your application, instead of starting it with WRun as in some of the examples. This linkgives more information as well if needed.)

The interface provided here will operate as follows:

- Instead of having a resource called "/resource" as in the above example, you will instead have one called "/people". When that URL is visited, it will provide a list of all the people in the address book, but won't provide the full information on each person. Instead, it will only provide their integer position in your list (1, 2, 3, ...) as an id and their first and last names. This way, you can quickly get a full list, and only acquire full information on people as you need it. For example, a sample list could be:

```
{
  "people": [
    { "id": "1", "lastname": "Doe", "firstname": "John" },
    { "id": "2", "lastname": "Smith", "firstname": "Bob" },
    ...
  ]
}
```

- Of course, there are other ways to name, format, and nest things. As long as it's valid JSON in the end capturing the list of people, that's the main thing.
- To access a person's full information, you would use "/people/1" for the first person, "/people/2" for the second, and so on. This would provide their id (as above), name, address, and phone number as generated above. For example, a sample person record could be:

```
{
  "id": "1",
```

```
    "lastname": "Doe",
    "firstname": "John",
    "streetaddress": "123 Fake Street",
    "city": "London",
    "province": "Ontario",
    "country": "Canada",
    "phone": "555-555-5555"
  }
```

- Again, there are other ways to name, format, and nest things. As long as it's valid JSON in the end, that's the main thing.

Most methods of storing your people objects in C++ will give you a numeric position that can be used as an id. (Either from the data structure itself, or when you are iterating through it.) If your structure does not allow you to do this, you will then need to attach an integer id into your person objects as you are creating them. (For the most part, this should not be necessary.)

To generate JSON for your address book information, your should include a "to JSON" method in your person, name, address, and phone number classes. You can then generate things in chunks in a reasonably clean and simple fashion. (They can be packed to strings or wherever else necessary, depending on your implementation.) You are free to do this manually yourself, composing the JSON as needed, or you can use the JSON library provided by Wt. The manual process is likely simpler and more straightforward for the amount of data in this case, though using the library is likely more elegant. (It won't make a difference to your overall grade, so do what works best for you.)