CLASSES AND PRIVACY

Workshop 3 (10 marks – 3.75% of your final grade)

In this workshop, you are to code a class with private data members and public member functions. The class will represent a book and can be used to manage a collection of books that a person has in her possession.

LEARNING OUTCOMES

Upon successful completion of this workshop, you will have demonstrated the abilities to

- define a class type;
- privatize data within the class type;
- instantiate an object of class type;
- access data within an object of class type through public member functions;
- use standard library facilities to format data inserted into the output stream;
- truncate a C-style null-terminated string to fit in memory
- describe what you have learned in completing this workshop.

SUBMISSION POLICY

The *in-lab* section is to be completed during your assigned lab section. It is to be completed and submitted by the end of the workshop period.

If you attend the lab period and cannot complete the *in-lab* portion of the workshop during that period, ask your instructor for permission to complete the *in-lab* portion after the period. You must be present at the lab in order to get credit for the *in-lab* portion.

If you do not attend the lab, you can submit the *in-lab* section along with your *at-home* section (see penalties below). The *at-home* portion of the lab is due on the day that is four days after your scheduled *in-lab* workshop (@23:59) (even if that day is a holiday).

All your work (all the files you create or modify) must contain your name, Seneca email and student number.

You are responsible to back up your work regularly.

LATE SUBMISSION PENALTIES:

- *In-lab* portion submitted late, with *at-home* portion: **0** for *in-lab*. Maximum of **7**/10 for the entire workshop.
- If any of *in-lab*, *at-home* or *reflection* portions is missing, the mark for the workshop will be **0**/10.

IN-LAB (30%)

Design and code a module named `CRA_Account`. The application that uses your module is listed below. Store your class definition in a header file named `CRA_Account.h` and your function definitions in an implementation file named `CRA_Account.cpp`.

Include a **compilation safeguard** in the header file. Enclose all declarations and definitions within the `sict` namespace.

In your `CRA_Account.h` header file, predefine the following constants as integers:

`max_name_length` with a value of 40. This value represents the maximum number of characters for any name of the account holder (not including the null byte `'\0'`).

`min_sin` with a value of 100000000. This is the smallest social insurance number that is acceptable.

`max_sin` with a value of 999999999. This is the largest social insurance number that is acceptable.

Your `CRA_Account` module validates any SIN that it receives. For this in-lab submission, your implementation consider a SIN valid if it is a 9-digit number between the predefined limits.

Define the `CRA_Account` class with private members that hold the following data for the account:

- the family name
- the given name
- the social insurance number (SIN)

Declare the following public member functions in your class definition:

`void set(const char* familyName, const char* givenName, int sin)`: This function checks if `sin` is a valid SIN. If valid, this function stores the family and given names on the account along with `sin`. If `sin` is not valid, this function stores values that identify an empty state. It is up to you to select the data values that identify the state as empty. In designing your function make sure that it does not overflow the space

allocated for the family and given names. Store only as many characters as there is space available. Use the `strncpy(...)` function in the `<cstring>` library for this purpose. In using this function, make sure that you set the last byte in the memory statically allocated for the name to the null byte.

`bool isEmpty() const`: This function returns true if the object is in an empty state, false if the object is not empty.

`void display() const`: If the object is not empty, this function inserts into the standard output stream the object's data in the following format:

```
Family Name: FAMILY_NAME<ENDL>
Given Name : GIVEN_NAME<ENDL>
CRA Account: SIN<ENDL>
```

If the object is empty, your function only inserts the following message:

```
Account object is empty!<ENDL>
```

IN-LAB MAIN MODULE

```
#include <iostream>
#include "CRA_Account.h"
#include "CRA_Account.h" // intentional

using namespace std;
using namespace sict;

int main() {
    sict::CRA_Account myCRA;
    int sin;
    bool quit = false;
    char familyName[sict::max_name_length + 1];
    char givenName[sict::max_name_length + 1];

    cout << "Canada Revenue Agency Account App" << endl
         << "==================================" << endl << endl;
    cout << "Checking constants" << endl
         << "----------------------------------" << endl
         << "max_name_length: " << sict::max_name_length << endl
         << "        min_sin: " << sict::min_sin << endl
         << "        max_sin: " << sict::max_sin << endl
         << "----------------------------------" << endl << endl;
    cout << "Please enter your family name: ";
    cin >> familyName;
    cin.ignore();
    cout << "Please enter your given name: ";
    cin >> givenName;
    cin.ignore();
```

```
    do {
        cout << "Please enter your social insurance number (0 to quit): ";
        cin >> sin;
        cin.ignore();
        if (sin != 0) {
            myCRA.set(familyName, givenName, sin);
            if (myCRA.isEmpty())
                cout << "Invalid input! Try again." << endl;
            else
                quit = true;
        }
        else
            quit = true;
    } while (!quit);

    cout << "----------------------------------------" << endl;
    cout << "Testing the display function" << endl;
    cout << "----------------------------------------" << endl;
    myCRA.display();
    cout << "----------------------------------------" << endl;

    return 0;
}
```

IN-LAB EXPECTED OUTPUT

```
Canada Revenue Agency Account App
=================================

Checking constants
---------------------------------
max_name_length: 40
        min_sin: 100000000
        max_sin: 999999999
---------------------------------

Please enter your family name: Doe
Please enter your given name: Jane
Please enter your social insurance number (0 to quit): 234
Invalid input! Try again.
Please enter your social insurance number (0 to quit): 1234567890
Invalid input! Try again.
Please enter your social insurance number (0 to quit): 193456787
----------------------------------------
Testing the display function
----------------------------------------
Family Name: Doe
Given Name : Jane
CRA Account: 193456787
----------------------------------------
```

IN-LAB SUBMISSION

To test and demonstrate execution of your program on Visual Studio use the same data as the output example above.

Upload `CRA_Account.h`, `CRA_Account.cpp` and `w3_in_lab.cpp` to your matrix account. Re-compile and rerun your code and make sure everything works properly.

Then, run the following command from your account (use your professor's Seneca userid to replace `profname.proflastname`, and your section ID to replace `XXX`, i.e., SAA, SBB, etc.):

`~profname.proflastname/submit 244XXX_w3_lab``<ENTER>`

and follow the instructions generated by the command and your program.

IMPORTANT: Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

AT-HOME (30%)

The *at-home* section of this workshop upgrades your `CRA_Account` module to improve the validation and to track the balance owing or refund due on the account over several years. Copy the original module to your at-home directory and add the following:

In your `CRA_Account.h` header file, predefine the following constant as an integer:

`max_yrs` with a value of 4. This value represents the maximum number of tax years that a `CRA_Account` object can remember.

Add the following attributes to your `CRA_Account` class:

a) An array of size `max_yrs` to hold the tax return years.
b) An array of size `max_yrs` to hold the balance owed or refund due on the account for each tax return year remembered.

c) The number of years for which tax return data is stored

Declare the following public member function in your class definition:

`void set(int year, double balance)`: If the object is not empty and has room to store tax return data, this function stores `year` and `balance` in the object. If not, this function does nothing.

Modify the definitions of the following public member functions:

`void set(const char* familyName, const char* givenName, int sin)`: validate the data received: if `sin` is valid and if the family and given names are not empty, this function stores the family and given names on the account along with the SIN *and initializes the number of years for which data is stored to 0*. If any part of the data received is invalid, this function stores values that identify an empty state. For this submission, your function considers a SIN valid if it is a 9-digit number between the predefined limits and has digits that satisfy the following rule:

A Canadian Social Insurance Number (SIN) has nine digits. The rightmost digit is a check digit that validates the other digits. For an integer to be a valid SIN, it must contain nine digits and the weighted sum of all digits including the check digit must be divisible by 10.

To obtain the weighted sum, take the digit to the left of the check digit and then every second digit leftwards (as shown below). Add each of these digits to itself. Then, add each digit of each sum to form the weighted sum of the even positioned digits. Add each of the remaining SIN digits (except the check digit) to form the sum of the odd positioned digits. Add the two sums and subtract the next highest number ending in zero from their total. If this number is the check digit, the whole number is a valid SIN; otherwise, the whole number is not a valid SIN.

For example:

```
SIN  193 456 787
               |
check digit is 7
add first set of alternates to themselves
   9  4  6  8
   9  4  6  8
  18  8 12 16
add the digits of each sum 1+8+8+1+2+1+6 = 27
add the other alternates   1+3+5+7       = 16
total                                    = 43
Next highest integer multiple of 10      = 50
Difference                               =  7
 Matches the check digit, therefore this number is a valid SIN
```

`void display() const`: if the object is not empty, this function inserts into the standard output stream the object's data in the following format:

- If the balance owing is > $2

```
Family Name: FAMILY_NAME<ENDL>
Given Name : GIVEN_NAME<ENDL>
CRA Account: SIN<ENDL>
Year (YEAR) balance owing: AMOUNT<ENDL>
```

- If the refund due is > $2

```
Family Name: FAMILY_NAME<ENDL>
Given Name : GIVEN_NAME<ENDL>
CRA Account: SIN<ENDL>
Year (YEAR) refund due: AMOUNT<ENDL>
```

- In all other cases

```
Family Name: FAMILY_NAME<ENDL>
Given Name : GIVEN_NAME<ENDL>
CRA Account: SIN<ENDL>
Year (YEAR) No balance owing or refund due! <ENDL>
```

If the object is empty, your function inserts the following message:

```
Account object is empty!<ENDL>
```

Make sure that the printed amount has exactly two digits.

AT-HOME MAIN MODULE

```
#include <iostream>
```

```cpp
#include "CRA_Account.h"
#include "CRA_Account.h" // intentional

using namespace std;
using namespace sict;

int main() {
    CRA_Account myCRA;
    int sin;
    bool quit = false;
    char familyName[max_name_length + 1];
    char givenName[max_name_length + 1];

    cout << "Canada Revenue Agency Account App" << endl
         << "==================================" << endl << endl;

    do {
        cout << "Please enter your family name: ";
        cin.getline(familyName, max_name_length);
        cout << "Please enter your given name: ";
        cin.getline(givenName, max_name_length);
        cout << "Please enter your social insurance number (0 to quit): ";
        cin >> sin;
        cin.ignore();
        if (sin != 0) {
            myCRA.set(familyName, givenName, sin);
            if (myCRA.isEmpty()) {
                cout << "Invalid input! Try again." << endl;
            }
            else {
                int year;
                double balance;
                do {
                    cout << "Please enter the year of your tax return (0 to quit): ";
                    cin >> year;
                    cin.ignore();
                    if (year != 0) {
                        cout << "Please enter the balance owing on your tax return (0
to quit): ";
                        cin >> balance;
                        cin.ignore();
                        myCRA.set(year, balance);
                    }
                } while (year != 0);
                quit = true;
            }
        }
        else {
            quit = true;
        }
    } while (!quit);
    cout << "----------------------------------------" << endl;
    cout << "Testing the display function" << endl;
    cout << "----------------------------------------" << endl;
    myCRA.display();
    cout << "----------------------------------------" << endl;
```

```
    return 0;
}
```

At-Home Expected Output

```
Canada Revenue Agency Account App
=================================

Please enter your family name: Doe
Please enter your given name:
Please enter your social insurance number (0 to quit): 193456787
Invalid input! Try again.
Please enter your family name: Doe
Please enter your given name: Jane
Please enter your social insurance number (0 to quit): 123456789
Invalid input! Try again.
Please enter your family name: Doe
Please enter your given name: Jane
Please enter your social insurance number (0 to quit): 193456787
Please enter the year of your tax return (0 to quit): 2014
Please enter the balance owing on your tax return (0 to quit): 34.56
Please enter the year of your tax return (0 to quit): 2015
Please enter the balance owing on your tax return (0 to quit): -1234
Please enter the year of your tax return (0 to quit): 2016
Please enter the balance owing on your tax return (0 to quit): 1.23
Please enter the year of your tax return (0 to quit): 2013
Please enter the balance owing on your tax return (0 to quit): -0.12
Please enter the year of your tax return (0 to quit): 0
---------------------------------------
Testing the display function
---------------------------------------
Family Name: Doe
Given Name : Jane
CRA Account: 193456787
Year (2014) balance owing: 34.56
Year (2015) refund due: 1234.00
Year (2016) No balance owing or refund due!
Year (2013) No balance owing or refund due!
---------------------------------------
```

Reflection (40%)

Study your final solution, reread the related parts of the course notes, and make sure that you have understood the concepts covered by this workshop. **This should take no less than 30 minutes of your time.**

Create a file named `reflect.txt` that contains your **detailed description of the topics that you have learned** in completing this workshop and mention any issues that caused you difficulty. Include in your explanation—**but do not limit it to**—the following points:

1) Explain what is the difference between a null string and an empty string?
2) Your class declares two member functions named `set(...)`. In C, this would generate an error. Name the feature of C++ that allows this.
3) For the *at-home* portion you had to change the logic that validates a SIN. How would you design your class in such a way that if a new update to the validation logic is necessary, you don't have to change anything in the function `CRA_Account::set(...)`?
4) What have you learned in completing this workshop?

Quiz Reflection

Add a section to `reflect.txt` called Quiz X Reflection. Replace the X with the number of the last quiz that you received and list the numbers of all questions that you answered incorrectly.

Then for each incorrectly answered question write your mistake and the correct answer to that question. If you have missed the last quiz, then write all the questions and their answers.

At-Home Submission

To submit the *at-home* section, demonstrate execution of your program with the exact output as in the example above.

Upload `reflect.txt`, `CRA_Account.h`, `CRA_Account.cpp` and `w3_at_home.cpp` to your matrix account. Compile and run your code and make sure everything works properly.

To submit, run the following command from your account (use your professor's Seneca userid to replace `profname.proflastname`, and your section ID to replace XXX, i.e., SAA, SBB, etc.):

~profname.proflastname/submit 244XXX_w3_home<ENTER>

and follow the instructions generated by the command and your program.

IMPORTANT: Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.