CLASS WITH A RESOURCE

Workshop 6 (10 marks – 3.75% of your final grade)

In this workshop, you are to design and code a class type that accesses a resource.

LEARNING OUTCOMES

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

- define a copy constructor
- define an assignment operator
- define a destructor
- avoid duplication in coding these special member functions
- 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 a class named `Contact`, in namespace `sict`. This class holds information about a person and their phone numbers. The maximum number of characters in the person's name is stored in an unmodifiable variable named MAX_CHAR. The type `Contact` contains the following information:

- an array of characters of size MAX_CHAR (excluding the null byte `'\0'`) that holds the name of the contact;
- a pointer to a **dynamically allocated** array of phone numbers. A valid phone number has one to two digits for the country code (cannot be zero), **exactly** three digits for the area code (cannot start with zero) and **exactly** seven digits for the number (cannot start with zero);
- the number of phone numbers currently stored in the **dynamically allocated** array;

Your type exposes the following public member functions:

- **default constructor** (a constructor with no parameters): this constructor sets the current object to a safe empty state;
- **constructor with 3 parameters**: This constructor receives the address of an unmodifiable C-style string that holds the name of the contact, the address of an unmodifiable array of phone numbers and the number of phone numbers stored in the array. This constructor allocates enough memory dynamically to hold **only the valid** phone numbers from the array received and copies the valid numbers into the allocated array;
- **destructor**: the destructor deallocates any memory that has been allocated dynamically;
- `bool isEmpty() const`: a query that returns false if the current object has valid data; true if the object is in a safe empty state;
- `void display() const`: a query that inserts into the standard output stream the data stored by the object applying the following format:

  If the object is in a safe empty state, this function displays the following message:

```
Empty contact!<ENDL>
```

If the object is not in a safe empty state, this function displays the following message:

```
CONTACT_NAME<ENDL>
(+COUNTRY_CODE) AREA_CODE NNN-NNNN<ENDL>
(+COUNTRY_CODE) AREA_CODE NNN-NNNN<ENDL>
•••
(+COUNTRY_CODE) AREA_CODE NNN-NNNN<ENDL>
```

Introduce as many private member functions as necessary to avoid any duplication of code in your `Contact` module. Store your class definition in a header file named `Contact.h` and your function definitions in an implementation file named `Contact.cpp`. Avoiding duplication will reduce the time that you will need to spend on the at home section.

Using the sample implementation of the `w6_in_lab.cpp` main module listed below, test your code and make sure that it works on Visual Studio. The expected output from your program is listed below this source code. The output of your program should match **exactly** the expected one.

IN-LAB MAIN MODULE

```
#include <iostream>
#include "Contact.h"

using namespace std;
using namespace sict;

int main()
{
    cout << "----------------------------------------" << endl;
    cout << "Maximum no of characters in a name: "<< sict::MAX_CHAR << endl;
    cout << "----------------------------------------" << endl;
    cout << "Testing the default constructor!" << endl;
    cout << "----------------------------------------" << endl;
    sict::Contact empty; // sict:: intentional
    empty.display();
    cout << "----------------------------------------" << endl << endl;

    cout << "----------------------------------------" << endl;
    cout << "Testing an invalid contact!" << endl;
    cout << "----------------------------------------" << endl;
    Contact bad(nullptr, nullptr, 0);
    bad.display();
    Contact alsoBad("", nullptr, 0);
    alsoBad.display();
    cout << "----------------------------------------" << endl << endl;

    cout << "----------------------------------------" << endl;
    cout << "Testing the constructor with parameters!" << endl;
    cout << "----------------------------------------" << endl;
```

```
    Contact temp("A contact with a very looooong name!", nullptr, 0);
    temp.display();
    cout << "----------------------------------------" << endl << endl;

    cout << "----------------------------------------" << endl;
    cout << "Testing a valid contact!" << endl;
    cout << "----------------------------------------" << endl;
       long long phoneNumbers[] = { 1416123456LL, 14161234567LL, 1416234567890LL,
              14162345678LL, -1LL, 124163456789LL,
              14161230002LL };
       Contact someContact("John Doe", phoneNumbers, 7);
    someContact.display();
    cout << "----------------------------------------" << endl << endl;

    return 0;
}
```

IN-LAB OUTPUT

```
----------------------------------------
Maximum no of characters in a name: 20
----------------------------------------
Testing the default constructor!
----------------------------------------
Empty contact!
----------------------------------------

----------------------------------------
Testing an invalid contact!
----------------------------------------
Empty contact!
Empty contact!
----------------------------------------

----------------------------------------
Testing the constructor with parameters!
----------------------------------------
A contact with a ver
----------------------------------------

----------------------------------------
Testing a valid contact!
----------------------------------------
John Doe
(+1) 416 123-4567
(+1) 416 234-5678
(+12) 416 345-6789
(+1) 416 123-0002
----------------------------------------
```

IN-LAB SUBMISSION

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

If not on matrix already, upload `Contact.h`, `Contact.cpp` and `w6_in_lab.cpp` to your matrix account. Compile and run 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_w6_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 your professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

AT-HOME (30%)

For the *at-home* part, copy the `Contact` module that you created for your *in-lab* submission. Declare the following additional member functions in the class definition and implement them in the .cpp file:

copy constructor: this constructor receives an unmodifiable reference to a `Contact` object copies that object into the newly created instance;

copy assignment operator: this operator receives an unmodifiable reference to a `Contact` object and copies the content of that object into the existing current object and returns a reference to the current object, as updated.

an overloaded += operator: this operator receives a new phone number as its parameter, validates the number received and, if valid, resizes the phone number array to hold all the existing numbers plus the received one and finally adds this new number to the array. If the number is not valid, this operator retains the original array and does not add the number. In either case, this operator returns a reference to the current object.

NOTE: When you implement the copy assignment operator, make sure that the `Contact` instance is not being set to itself.

Using the sample implementation of the `w6_at_home.cpp` main module listed below, test your code and make sure that it works on Visual Studio. The expected output from your program is listed below this source code. The output of your program should match **exactly** the expected one.

AT-HOME MAIN MODULE

```
#include <iostream>
#include "Contact.h"

using namespace std;
using namespace sict;

int main() {
    cout << "----------------------------------------" << endl;
    cout << "Maximum no of characters in a name: "<< sict::MAX_CHAR << endl;
    sict::Contact theContact("John Doe", nullptr, 0); // sict:: intentional
    theContact.display();
    theContact += 14161234567LL;
    theContact += 14162345678LL;
```

```
    theContact += 14163456789LL;
    theContact += 114164567890LL;
    theContact.display();

    cout << endl << "Testing! Please wait:" << endl;

    for (int i = 1; i <= 5000000; i++)
    {
        Contact temp = theContact;
        theContact = temp;
        theContact = theContact;
        if (!(i % 10000))
            cout << ".";
        if (!(i % 500000))
            cout << endl;
    }
    cout << endl;
    theContact.display();

       theContact = Contact("", nullptr, 0);
       theContact += 14161230002LL;
       theContact.display();

    return 0;
}
```

AT-HOME EXPECTED OUTPUT

```
----------------------------------------
Maximum no of characters in a name: 20
-John Doe
John Doe
(+1) 416 123-4567
(+1) 416 234-5678
(+1) 416 345-6789
(+11) 416 456-7890

Testing! Please wait:
.................................................
.................................................
.................................................
.................................................
.................................................
.................................................
.................................................
.................................................
.................................................
.................................................

John Doe
(+1) 416 123-4567
(+1) 416 234-5678
(+1) 416 345-6789
(+11) 416 456-7890
Empty contact!
```

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) Why does the copy assignment operator check for self-assignment before doing anything else? What could go wrong if the operator doesn't check for self-assignment?
2) List examples of how you avoided code duplication.

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`, `Contact.h`, `Contact.cpp` and `w6_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_w6_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.