Part C - Encapsulation

Class with a Resource

Workshop 5 V1.0

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

LEARNING OUTCOMES

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

- code a copy constructor
- code an assignment operator
- code a destructor
- reflect on what you have learned in 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. If you do not attend the workshop, you can submit the "in-lab" section along with your "at-home" section (a 20% late deduction will be assessed). The "at-home" portion of the lab is **due the day before your next scheduled workshop**

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

You are responsible to regularly backup your work.

CONTACT CLASS

Download or clone workshop 5 from https://github.com/Seneca-244200/OOP-Workshop5

Open Workshop5/in_lab directory and view the code in `PhoneNumber.h` and `PhoneNumber.cpp`.

PhoneNumber is designed and coded to receive, hold and display information about a phone number. These information are: type of number (work, mobile, home), ,areacode and number.
PhoneNumber has the many methods implemented, but for now those important for you are the following:
void read(). Gets a phone number and its type from console in following format: Type, areacode number (Home, 555 4564567).

void display(). Displays the phone number on the screen with no newline after:
Home, 555 456-4567

Your task is to complete the code of a class called Contact that holds a person's name and her phone numbers. (home, office, mobile and etc). Since the number of phone numbers are unknown, the Contact class should keep them in a dynamic array of PhoneNumbers.

The class **Contact** has the following three data members:

- **name**_: A C-style string holding the name of the **Contact** to the maximum of 40 characters excluding the terminating null.
- **phoneNumber_**: A **PhoneNumber** pointer that will be used to point to the dynamic array of **PhoneNumbers**.
- **noPN**_: An integer indicating the number of PhoneNumbers in a **Contact** class.

Contact has the following member functions (methods) that are already implemented in **Contact.cpp.**

- **bool read()** – Receives the Contact information from console.
- **void setEmpty()**- sets the object to a safe empty state.
  Safe Empty state: sets the name to an empty string, _noPN to 0 and finaly _pn (PhoneNumber pointer) to null
- **bool isEmpty()** – return true if the object is in a safe empty state.

IN-LAB SECTION CONSTRUCTORS, DESTRUCTOR AND DISPLAY(60%)

Write the following for the **Contact** class:

- **Default constructor:** Sets the **Contact** to an empty safe state.
- **A two argument Constructor:** Accepts a const char* name and the number of phone numbers for the contact.

This constructor will allocates enough memory to hold the phone numbers and keeps the address in the **_pn** pointer. Then it will copy the name into the **_name** attribute and set **_noPN** to the number of phone numbers.

- **`Destructor`**
  Write a destructor for the **`Contact`** class making sure that the allocated memory is deallocated when the object goes out of scope.
- **`void display() const`** - a query that displays the contact's name and then lists all the phone numbers kept in class. To list the phone numbers, this function uses the display of the **`PhoneNumber`** class and prints them one by one from the array of PhoneNumbers held by **_pn**.

  Contact with no (or no valid) phone numbers
  ```
  John Doe
  ----------------------------------------
  ```
  Contact with phone numbers are displays like this:
  ```
  John Doe
  Home..........., 555 123-1234
  Work Mobile...., 555 345-3456
  Office........., 555 432-2000
  ----------------------------------------
  ```

CLIENT MODULE

Here is a sample of implementation file for the **`w5_in_lab.cpp`** main module that you should use to test your implementation:

```
// OOP244 Workshop 5: Classes with resources
// File: w5_in_lab.cpp
// Version: 1.0
// Date: 2016/02/15
// Author: Fardad Soleimanloo
// Description:
// This file tests in-lab section of your workshp
///////////////////////////////////////////

#include <iostream>
using namespace std;
#include "Contact.h"
using namespace sict;

int main(){
  Contact c("Empty Contact", 3);
  c.display();
```

```
  cout << "Enter Contact information: " << endl;
  c.read();
  c.display();
  cout << endl;
  for (int i = 0; i < 1000000; i++){
    Contact temp("Testing the contact with a looooong "
                 "name that should be taken care of", 20);
    if (!(i % 100000)){
      cout << i << ":" << endl;
      temp.display();
    }
  }
  return 0;
}
```

Output Example:
(Your output should exactly match the following)

```
Empty Contact
----------------------------------------
Enter Contact information:
Contact Name: John Doe
Please enter 3 phone numbers:
1: Home, 123 1234567
2: Cell, 234 2345678
3: Work, 345 3456789
John Doe
Home..........., 123 123-4567
Cell..........., 234 234-5678
Work..........., 345 345-6789
----------------------------------------

0:
Testing the contact with a looooong name
----------------------------------------
100000:
Testing the contact with a looooong name
----------------------------------------
200000:
Testing the contact with a looooong name
----------------------------------------
300000:
Testing the contact with a looooong name
----------------------------------------
400000:
```

```
Testing the contact with a looooong name
----------------------------------------
500000:
Testing the contact with a looooong name
----------------------------------------
600000:
Testing the contact with a looooong name
----------------------------------------
700000:
Testing the contact with a looooong name
----------------------------------------
800000:
Testing the contact with a looooong name
----------------------------------------
900000:
Testing the contact with a looooong name
----------------------------------------
```

IN-LAB SUBMISSION (60%)

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**, **PhoneNumber.h**, **PhoneNumber.cpp** and **w5_in_lab.cpp** to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account:

```
Sections SAA and SBB:
~edgardo.arvelaez/submit w5_in_lab <ENTER>
Section SCC and SDD:
~fardad.soleimanloo/submit w5_in_lab <ENTER>
Section SEE and SFF:
~eden.burton/submit w5_in_lab <ENTER>
```

and follow the instructions.

AT-HOME SECTION: COPY CONSTRUCTOR, ASSIGNMENT OPERATOR AND REFLECTION (40%)

Add the following to the Contact class:

- **Copy Constructor –** This constructor should safely make a copy of an already existing contact.
  Details:
  If the contact which is being copied is empty, then set the current object to empty, otherwise, copy the name, then allocate a dynamic array of PhoneNumbers to the exact size of the array of the object that is being copied. Then set each PhoneNumber of the current object to the corresponding PhoneNumber of the contact which is being copied.

- **The assignment operator:**
  `Contact& operator=(const Contact& C)`
  Sets the current object to another Contact.
  Details:
  First make sure that this **Contact** is not being set to itself. If so, abort the copying and just return the current **Contact**.
  Otherwise, first delete the dynamic array that is currently being pointed by _pn and then do exactly as you have done in the copy constructor. (Make sure you are reusing your logic and not repeating the same code).

CLIENT MODULE

Here is a sample of implementation file for the **w5_at_home.cpp** main module that you should use to test your implementation:

```
// OOP244 Workshop 5: Classes with resources
// File: w5_at_home.cpp
// Version: 1.0
// Date: 2016/02/15
// Author: Fardad Soleimanloo
// Description:
// This file tests in-lab section of your workshop
///////////////////////////////////////////

#include <iostream>
using namespace std;
#include "Contact.h"
using namespace sict;

int main(){
  Contact c("Empty Contact", 3);
  c.display();
  cout << "Enter Contact information: " << endl;
  c.read();
```

```
  c.display();
  cout << "Please wait";
  for (int i = 0; i < 1000000; i++){
    Contact cp = c;
    c = cp;
    if (!(i % 100000)) cout << ".";
  }
  cout << endl;
  c.display();
  return 0;
}
```

Output Example:
(Your output should exactly match the following)

```
Empty Contact
----------------------------------------
Enter Contact information:
Contact Name: John Doe
Please enter 3 phone numbers:
1: Home, 123 1234567
2: Cell, 234 2345678
3: Work, 345 3456789
John Doe
Home..........., 123 123-4567
Cell..........., 234 234-5678
Work..........., 345 345-6789
----------------------------------------
Please wait.........
John Doe
Home..........., 123 123-4567
Cell..........., 234 234-5678
Work..........., 345 345-6789
----------------------------------------
```

AT-HOME SUBMISSION (30%) AND REFLECTION (10%)

Please provide brief answers to the following questions in a text file named **reflect.txt.**

1. Explain how allocateAndCopy() function works in PhoneNumber.cpp
2. Explain why usually a copy constructor and assignment operator is needed when a class uses dynamic memory allocation to hold its resources.

SUBMISSION

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

If not on matrix already, upload **Contact.h**, **Contact.cpp**, **PhoneNumber.h**, **PhoneNumber.cpp** and **w5_at_home.cpp** to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account:

```
Sections SAA and SBB:
~edgardo.arvelaez/submit w5_at_home <ENTER>
Section SCC and SDD:
~fardad.soleimanloo/submit w5_at_home <ENTER>
Section SEE and SFF
~eden.burton/submit w5_at_home <ENTER>
```

and follow the instructions.