SC1008 Assignment 2 of C++: Class, Inheritance and STL

1. **(Shape Areas) [10 Marks]** Implement a simple class hierarchy to represent two-dimensional (2D) shapes. Your abstract base class, `Shape`, will define a common interface through a pure virtual function, `calArea()`, which will be implemented in the derived class to compute the area of different 2D shapes. It also has a member variable called `area`, and a member function called `getArea().` You will then create two derived classes, `Circle` and `Rectangle`, which override the `calArea()` function to calculate their specific areas.
   Here are the detailed members in each class:
   - Class `Shape`
     - Attribute: `double area`
     - Constructor: `Shape()`
       - It initializes the area as 0.0 and prints out “Shape Constructor!”
     - Destructor: `~Shape()`
       - It prints out “Shape Destructor!”
     - **A pure virtual function**: `void calArea()`
     - A member function: `double getArea()`

   - Class `Circle` (inherits from `Shape`)
     - Attribute: `radius`
     - Constructor: `Circle(double r)`
       - It initializes the radius and prints out “Circle Constructor!”
     - Destructor: `~Circle()`
       - It prints out “Circle Destructor!”
     - Function: `void calArea()`

   - Class Rectangle (inherits from `Shape`)
     - Attribute: `width, height`
     - Constructor: `Rectangle(double w, double h)`
       - It initializes the width and height and prints out “Rectangle Constructor!”
     - Destructor: `~Rectangle()`
       - It prints out “Rectangle Destructor!”
     - Function: `void calArea()`

   Below is the starting code.

```
#include <iostream>
#include <cmath>  // For M_PI
#include <type_traits>  // Required for std::is_abstract

// Abstract base class
class Shape {
protected:
    double area;
```

```
public:
    // TO-DO: Please implement the constructor, the destructor and the calArea()
function here
    //
    //


    // Member function to get the area
    double getArea() const {
        return area;
    }
};

// Derived class: Circle
class Circle : public Shape {
private:
    double radius;

public:
    // TO-DO: Please implement the constructor, the destructor and OVERRIDE the
calArea() function here
    //
    //


};

// Derived class: Rectangle
class Rectangle : public Shape {
private:
    double width;
    double height;

public:
    // TO-DO: Please implement the constructor, the destructor and OVERRIDE the
calArea() function here
    //
    //


};

int main() {
    std::cout << std::boolalpha;
    std::cout << "Is Shape abstract? " << std::is_abstract<Shape>::value <<
std::endl<< std::endl;

    Shape* shape1 = new Circle(5.0);
    Shape* shape2 = new Rectangle(4.0, 6.0);
    std::cout<<std::endl;

    shape1->calArea();
    shape2->calArea();

    std::cout << "Area of Circle: " << shape1->getArea() << std::endl;
    std::cout << "Area of Rectangle: " << shape2->getArea() << std::endl;
    std::cout<<std::endl;

    // Clean up
    delete shape1;
    delete shape2;

    return 0;
}
```

The sample outputs should be:

```
Is Shape abstract? true

Shape Constructor!
Circle Constructor!
Shape Constructor!
Rectangle Constructor!

Area of Circle: 78.5398
Area of Rectangle: 24

Circle Destructor!
Shape Destructor!
Rectangle Destructor!
Shape Destructor!
```

2. **(Phonebook) [10 Marks]** You are creating a simple phone book application to map a person's name (string) to their phone number (string). Users should be able to add entries, remove entries, and look up phone numbers by name. Create a class `PhoneBook` by using STL container `map`, which has the following members:

   - **Private Members:**
     - `map<string,string> contacts`: stores name -> phoneNumber pairs.

   - **Public Members:**
     - A constructor that initializes an empty map.
     - A function `addContact(const string &name, const string &number)` that adds or updates a contact in the map.
     - A function `removeContact(const string &name)` that removes the contact if it exists.

- A function `findContact(const string &name)` that returns the phone number if the contact exists, or **a reminder message “Not Found!”** if it does not.
- A function `displayAllContacts()` that prints every name -> phoneNumber pair in the map.

Here is the code with missing implementation for you get started:

```cpp
#include <iostream>
#include <map>
#include <string>

using namespace std;

class PhoneBook {
private:
    map<string, string> contacts; // Maps names to phone numbers

public:
    // TO-DO: Implement the constructor
    //

    // TO-DO: Implement addContact(const string &name, const string &number)
    //


    // TO-DO: Implement removeContact(const string &name)
    //


    // TO-DO: Implement findContact(const string &name)
    //


    // TO-DO: Implement displayAllContacts()
    //

};

int main() {
    PhoneBook pb;
    pb.addContact("Alice", "12345678");
    pb.addContact("Bob", "23456789");
    pb.addContact("Charlie", "34567890");

    // Display contacts
    cout << "All Contacts:" << endl;
    pb.displayAllContacts();
    cout<<endl;

    // Find a contact
    string searchName = "Charlie";
    cout << "The contact number of " << searchName << ": "
         << pb.findContact(searchName) << endl <<endl;

    searchName = "David";
    cout << "The contact number of " << searchName << ": "
            << pb.findContact(searchName) << endl <<endl;

    // Remove a contact
    pb.removeContact("Bob");
    cout << "After removing Bob, contacts are:" << endl;
    pb.displayAllContacts();

    return 0;
}
```

The sample output should be as follows:

```
All Contacts:
Alice -> 12345678
Bob -> 23456789
Charlie -> 34567890

The contact number of Charlie: 34567890

The contact number of David: Not Found!

After removing Bob, contacts are:
Alice -> 12345678
Charlie -> 34567890
```