Advanced Programming

Homework Assignment 9

STL

General guidelines:

a. Maximize readability and ensure indentation.
b. Do exactly as required in the questions.
c. Every class in every question must be submitted in two seperate files – a header file (.h) and an implementation file (.cpp).
d. Add functions and helper methods as necessary to ensure readability.
e. Submission must be done according to the submission guidelines – a document published in the course website – read them carefully!
f. Use relevant names.
g. Use comments to document your functions and complex code parts. **Add an output example at the end of your file!**
h. Individual work and submission – paired submission is not allowed.

Important note: Unless otherwise specified, ever homework has no more than one week for submission.
Open submission boxes past the deadline are not permission for late submission.

Question 1 (and only):

This exercise deals with the soldier medal awarding decisions following the "Shomer Hachomot" operation.
In the operation there were three ranks of soldiers: privates, commanders and officers.
The following criteria were defined for medals to each rank:

- Private: Was part of at least 10 military operations and the average of his operation performance grades is greater than 95.
- Commander: Was part of at least 7 military operations and the average of his operation performance grades is greater than 90. Additionally must be assigned to a combat unit.
- Officer: Was part of at least 2 military operations and his sociometric grade from his soldiers is at least 92.

In order to represent the given ranks, define the hierarchy described below.

Define an abstract class called Soldier that represents a soldier.

The class must have the following fields: Id number, first name, last name, number of operations the soldier participated in.

The class must have (at least) the following methods:

- A constructor that initializes the fields.
- A virtual destructor.
- medal – a boolean method that checks if the soldier is eligible to receive a medal.
- print – prints the data of the soldier. The data is printed one field at a time, each field printed in one line preceded by a string describing it (see example below). Note: If there are no performance grades the output grades is not to be printed.
- soldierType – returns a string describing the soldier type – one of the strings "private", "commander" and "officer".

Define the following inheriting classes:

- PrivateSoldier – inherits from Soldier and represents a private. The class must contain apart from the inherited fields also an address of an array of performance grades for the operations the soldier participated in. The grades range between 0-100. Note: The class has dynamic fields and must implement the necessary methods to allow for deep copying.
- Commander – inherits from PrivateSoldier and represents a commander. The class must contain also a boolean field that is true if the commander is assigned to a combat unit.
- Officer – inherits from Soldier and represents an officer. The class must contain apart from the inherited fields also a field containing his sociometric grade.

Note: All aspects of the classes must be declared and implemented correctly, including constructors (e.g. copy and move constructors), getters (setters are not required) and virtual or pure virtual methods where relevant.

The soldier data is to be printed in the following format:

```
PrivateSoldier
ID id number
first name: first name
last name: last name
num operations: number of ops
if the number of ops > 0
grades: the grades separated by
spaces
```

```
Commander
ID id number
first name: first name
last name: last name
num operations: number of ops
if the number of ops > 0
grades: the grades separated by
spaces
combat: yes/no
```

```
Officer
ID id number
first name: first name
last name: last name
num operations: number of ops
sociometric score: sociometric score
```

Complete the main program provided below to contain the following components:

- Declaration of a STL list or STL vector (your choice) that can contain soldiers of **<u>all</u>** ranks.
- Complete the declaration and implement the following global functions that perform operations on the STL container of soldiers of your choice:
  - add – receives the container and adds to it a new soldier.
    The implementation must make use of the following outputs only:

```
cout << "choose a soldier\n";
cout << "enter 1 to add a private\n";
cout << "enter 2 to add a commander\n";
cout << "enter 3 to add an officer\n";
cout << "enter id, first name, last name and number of operations\n";
cout << "enter "<<numOfOperations<<" grades\n";
cout << "enter 1 if the soldier is combat and 0 if not\n";
cout << "enter the sociometric score\n";
```

  - medal – receives the container and prints only those soldiers who are eligible to receive medals.
  - highestSociometricScore – receives the container and returns the address of the soldier who is an officer with the highest sociometric grade. If there are no officers the function returns NULL.
- Add a one-line operation to every place that is marked for completion. Make use of the STL algorithms and lambda expressions to achieve your goals.

Note: The implementation of highestSociometricScore must make use of the getSociometric getter from the Officer class. Since the elements in your collection are not Officers (necessarily) a direct call to the getter could cause a compiler error. This problem can be solved as follows:

1. Make sure the instance is actually an officer.
2. Use downcasting to the correct address type.
   For example:
   Given the declaration:
   ```
   Soldier *soldier = new Officer();
   ```
   Write:
   ```
   if (soldier->soldierType()==”officer”)
               ((Officer *) soldier)->getSociometric();
   ```

In the main function, after declaring your container of choice, the program runs a loop that terminates when the user chooses STOP (0). The loop displays a menu with the following choices:

0. Stop.
1. Add a new soldier.
2. Print the medal eligible soldiers.
3. Print the full name of the officer with the highest sociometric score.
4. Print the number of privates eligible to receive medals.
5. Print the full names of all the non-combatant commanders.
6. Test (and print a relevant message) if there exists a soldier who participated in more than 15 military operations.
7. Remove from the container all the officers who participated in no military operations.

Note: Every marked place in the main function must contain a single line of code only (possibly longer than the marked line). This can and must be done using combinations of STL algorithms and lambda expressions, as seen in the lectures and labs. You may consult the documentation in the website: https://www.cplusplus.com/reference/

The provided main program snippet:

```
#include <iostream>
#include <list>
#include <vector>
#include <algorithm>
#include <string>
using namespace std;

 enum option {
        EXIT,
        ADD_NEW_SOLDIER,
        DESERVES_MEDAL,
        HIGHEST_SOCIOMETRIC,
        PRIVATE_MEDAL_COUNT,
        NONCOMBAT_COMMANDER,
        SUPER_SOLDIER,
        REMOVE_OFFICER,
        };

// Complete the following declarations with your container of choice and implement
void add(_________ );
void printMedalList(_________ );
Soldier* highesttSociometricScore (_________ );

int main()
{
        Soldier* s;
        ________________    // Declare a STL vector/list that can contain soldiers
        int op;
        cout<<"enter 0-7\n";
```

```
        cin>>op;
        while(op!=EXIT)
        {
                switch (op)
                {
                case ADD_NEW_SOLDIER:add( ____________ );
                        break;
                case DESERVES_MEDAL:printMedalList( ____________ );
                        break;
                case HIGHEST_SOCIOMETRIC:
                        s=highestSociometricScore( _____ );
                        cout<<"Officer with the highest sociometric score: ";
                        cout <<s->getFirstName()<<' '<<s->getLastName()<<endl;

                        break;
                case PRIVATE_MEDAL_COUNT:
                        cout<<"number of privates that received medals: ";
                        _______________
                        cout<<endl;
                        break;
                        case NONCOMBAT_COMMANDER:
                        cout << "list of noncombat commanders: ";
                        _______________
                        cout<<endl;
                        break;
                case SUPER_SOLDIER:
                        if( _____________ )
                                cout << "there is at least one soldier that did
more than 15 operations\n";
                        else
                                cout << "no soldier did more than 15
                        operations\n";
                        break;
                case REMOVE_OFFICER:
                _______________ // Remove
                _______________ // Print the updates container
                        break;
                };
                cout<<"enter 0-7\n";
                cin>>op;
        }
        return 0;
}
```

Cont. below...

Execution example:

```
enter 0-7
1
choose a soldier
enter 1 to add a private
enter 2 to add a commander
enter 3 to add an officer
1
enter id, first name, last name and number of operations
111 aaa aaa 3
enter 3 grades
100 95 98
enter 0-7
1
choose a soldier
enter 1 to add a private
enter 2 to add a commander
enter 3 to add an officer
2
enter id, first name, last name and number of operations
222 bbb bbb 0
enter 1 if the soldier is combat and 0 if not
1
enter 0-7
1
choose a soldier
enter 1 to add a private
enter 2 to add a commander
enter 3 to add an officer
3
enter id, first name, last name and number of operations
333 ccc ccc 0
Enter the sociometric score
100
enter 0-7
3
Officer with the highest sociometric score: ccc ccc
enter 0-7
4
number of privates that received medals: 0
enter 0-7
5
list of noncombat commanders :
```

```
enter 0-7
6
no soldier did more than 15 operations
enter 0-7
7
private
ID: 111
first name: aaa
last name: aaa
num operations: 3
grades: 100 95 98
commander
ID: 222
first name: bbb
last name: bbb
num operations: 0
combat: yes
enter 0-7
0
```

Good Luck!