Advanced Programming

Homework Assignment 8

Templates, STL Containers

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 continues the implementation of a grant managing system for a higher education institution that we started designing in homework 6 and 7. Therefore, the descriptions of the classes Student, BA, MA and PHD are based on the requirements described in homework 6 and 7.

In this exercise, we will perform the following operations:

- Input all types of students into a STL linked list.
- Move each type of students into a separate STL vector.
- Sort each vector according to a particular student-type-related criterion.
- Use a STL stack and a STL queue to print the student data in descending order.

Part A: Modifications to Homework 7

- Change the access level of the method studType in all the classes from protected to public.
- Define a virtual destructor in Student. Implement **fully** the Rule of 5 in the class that has dynamic allocations (BA) – the STL containers make use of all 5.
- Add to the method print in Student the output "*** Scholarship ***" if the student is eligible to receive a grant (See execution example).
- Add in the global function addStudent the input option 0 for ending the input (See execution example). In this case the function is to return nullptr.

Part B: Add the operator<

- In the class Student declare the operator< as pure virtual:
```
virtual bool operator < (const Student& rhs) const = 0;
```

- In the class BA implement operator< with the following logic:
    - If the student types in the two operands are different, an exception is thrown: “cannot compare students of different types”.
    - If the left student is **ineligible** to receive a grant and the right student is **eligible** to receive a grant, return **true**.
    - If the left student is **eligible** to receive a grant and the right student is **ineligible** to receive a grant, return **false**.
    - If both students are eligible or both students are ineligible to receive a grant, true is returned if the left grade average is less than the right grade average. Otherwise return false.
- The class MA uses the implementation from class BA.
- The class PHD implementation is the same as the class BA implementation except that if both students have the same eligibility to receive a grant, true is returned if the left has less research hours than the right.

Part C: Rewrite main

Rewrite the main function so that the following operations are performed:

- Use the function addStudent to input students of all types until 0 is inputted. The inputted students are to be inserted into a STL list (What must the element type be?).
- Declare three separate STL vectors, one for each type of student. Copy each element from the list into the correct vector. Use studType to determine where the element should go. Free the memory of the element in the list.
  Note: The elements in the vectors must be instances and not addresses so that the sorting will work (Which casting is used when moving the elements?).
- Sort each of the vectors using the function sort from the library <algorithm>. The function sort receives an iterator to the beginning and an iterator to the end of some collection and sorts according to the operator<. The sorting will be in non-descending order.
- Declare two STL queues – one for the BA instances and one for the MA instances. Declare a STL stack for the PHD instances. Enqueue the instances of BA and MA using the const reverse iterator (from last to first) so that the best students will be in the front of the queue. Push the PHD students in order (from first to last) so that the best students will be at the top of the stack.
- Print all the instances, first BA, then MA, then PHD, while dequeueing and popping them from their queues and stack to result with empty queues and an empty stack.

As a result, all the inputted students will be printed in three lists, each degree separately, sorted according to the defined criteria.

Note: After freeing the allocations that were in the list no more freeing is required because all the rest of the dynamic allocations are within the classes and will be freed by their respective destructors.

Below is an execution example that demonstrates the expected behavior and output of the program. Make sure your outputs match those in the example.

Execution example:

```
Enter degree: 1 for BA, 2 for MA, or 3 for PhD, 0 to finish
1
enter id, first name, last name, number of courses
123 Reuven Reuveni 10
and enter a list of student grades
100 100 100 100 100 100 100 100 100 100
Enter degree: 1 for BA, 2 for MA, or 3 for PhD, 0 to finish
1
enter id, first name, last name, number of courses
234 Shimon Shimoni  10
and enter a list of student grades
95 95 95 95 95 95 95 95 95 95
Enter degree: 1 for BA, 2 for MA, or 3 for PhD, 0 to finish
1
enter id, first name, last name, number of courses
345 Levi Levi 10
and enter a list of student grades
99 99 99 99 99 99 99 99 99 99
Enter degree: 1 for BA, 2 for MA, or 3 for PhD, 0 to finish
1
enter id, first name, last name, number of courses
444 Ploni Almoni 0
and enter a list of student grades
Enter degree: 1 for BA, 2 for MA, or 3 for PhD, 0 to finish
2
enter id, first name, last name, number of courses
555 Yehuda Yehuda 7
and enter a list of student grades
95 95 95 95 95 95 95
enter 1 if the student does research and 0 if not
1
Enter degree: 1 for BA, 2 for MA, or 3 for PhD, 0 to finish
2
enter id, first name, last name, number of courses
678 Dan Dani 7
and enter a list of student grades
97 97 97 97 97 97 97
enter 1 if the student does research and 0 if not
1
Enter degree: 1 for BA, 2 for MA, or 3 for PhD, 0 to finish
3
enter id, first name, last name, number of courses
789 Naphtali Naphtali 2
enter number of research hours:
26
Enter degree: 1 for BA, 2 for MA, or 3 for PhD, 0 to finish
7
Exception: no such degree
Enter degree: 1 for BA, 2 for MA, or 3 for PhD, 0 to finish
```

```
3
enter id, first name, last name, number of courses
890 Gad Gadi 2
enter number of research hours:
32
Enter degree: 1 for BA, 2 for MA, or 3 for PhD, 0 to finish
3
enter id, first name, last name, number of courses
901 Asher Asher 1
enter number of research hours:
36
Enter degree: 1 for BA, 2 for MA, or 3 for PhD, 0 to finish
0
------------------------------------------------------------
BA student *** Scholarship ***
ID:     123
Name:   Reuven Reuveni
Grades: 100 100 100 100 100 100 100 100 100 100
Average: 100

BA student *** Scholarship ***
ID:     345
Name:   Levi Levi
Grades: 99 99 99 99 99 99 99 99 99 99
Average: 99

BA student
ID:     234
Name:   Shimon Shimoni
Grades: 95 95 95 95 95 95 95 95 95 95
Average: 95

BA student
ID:     444
Name:   Ploni Almoni
Grades:
Average: 0

MA student *** Scholarship ***
ID:     678
Name:   Dan Dani
Grades: 97 97 97 97 97 97 97
Average: 97
research:       YES

MA student *** Scholarship ***
ID:     555
Name:   Yehuda Yehuda
Grades: 95 95 95 95 95 95 95
Average: 95
research:       YES
```

```

PhD student *** Scholarship ***
ID:       890
Name:    Gad Gadi
Number of research hours:       32

PhD student *** Scholarship ***
ID:       789
Name:    Naphtali Naphtali
Number of research hours:       26

PhD student
ID:       901
Name:    Asher Asher
Number of research hours:       36
```

Cont. in the next page…

A trick to speed up the testing process: Prepare a text file with the expected input and copy from there into the console. Even if you put a breakpoint before all the input is read, the rest of the input will remain in the buffer and be read when the execution is continued. Such a text file can be reused and save the necessity of rewriting the input every time. If there are a number of test cases they can be saved in separate text files.

For example, the input for the execution example above will look like this:

```
1
123 Reuven Reuveni 10
100 100 100 100 100 100 100 100 100 100
1
234 Shimon Shimoni  10
95 95 95 95 95 95 95 95 95 95
1
345 Levi Levi 10
99 99 99 99 99 99 99 99 99 99
1
444 Ploni Almoni 0
2
555 Yehuda Yehuda 7
95 95 95 95 95 95 95
1
2
678 Dan Dani 7
97 97 97 97 97 97 97
1
3
789 Naphtali Naphtali 2
26
7
3
890 Gad Gadi 2
32
3
901 Asher Asher 1
36
0
```

Good Luck!