UNIVERSITÄT BASEL

Thorsten Möller
Maarten Schenk - maarten.schenk@unibas.ch
Michael Plüss - m.pluess@unibas.ch

UNI BASEL

Programming Paradigms – C++ FS 2019

Exercise 3 Due: 28.04.2019 23:59:59

Upload answers to the questions **and source code** before the deadline via courses.cs.unibas.ch. In addition, running programs have to be demonstrated during the exercise slots until **Friday, 03.05.2019** the latest. Also note, of all mandatory exercises given throughout the course, you must score at least 2/3 of their total sum of points to get accepted in the final exam.

Modalities of work: The exercise can be completed in groups of at the most 2 people. Do not forget to provide the full name of all group members together with the submitted solution.

Question 1: Constructors, Destructors (12 points)

In this question you will implement a AVL tree based on a binary tree. If you don't know what a AVL tree or a binary tree is or how they work, you can look them up here:
`https://en.wikipedia.org/wiki/AVL_treee`
`https://en.wikipedia.org/wiki/Binary_tree`
We are going to use this tree to save zip codes with their respective places. You can use any zip codes, but if you want real data:
`http://www.geonames.org/postalcode-search.html?q=&country=CH`

a) Write a class `AVLNode` (with a header file) implementing a single node of a AVL tree with a key as integer and a value as string and an integer balance.

   Each element should contain a integer key, its string value, integer balance and `pointers` to the parent, the left child and the right child nodes. Each node should also have functions to get its string value and to check if it is a leaf node in the tree.

   (3 points)

b) Write two constructors for the `AVLNode` class:

   1. A constructor taking the key/value pair and the pointer to the parent node of an AVL node as an argument.
   2. A constructor taking the key/value pair, the pointers to the parent node, left child and right child node of an AVL node as an argument.

(2 points)

c) Write a class `AVLTree` (with a header file) that uses the `AVLNode` class internally to store the key/value pairs.

Your implementation should contain:

- A pointer to the root of the tree stored in a private variable.
- A method `insert` which adds key/value pairs to the AVL tree.
- A method `search` which returns the value to the corresponding key or returns an empty string if the value was not found inside the tree.
- A method `rebalance` which checks depending on the node, if the tree needs to be rebalanced and if true does the rebalancing.
- A method `printBalance` which prints the balance of each node in layers of the tree (use | to denote a layer), if a node is not a leaf but has only one child print an x for the missing child. (E.g. 0 | 1 0 | 0 x 0 0 |)
- A destructor that makes sure all nodes are deleted.

Note: It's adviced to write also a method for each of the four different rotations (rotateLeft, rotateRight, rotateLeftThenRight, rotateRightThenLeft). You don't have to implement a delete method.

(6 points)

d) Write a main method to test your implementation.

You should insert at least 10 different zip codes with ther respective place and print the balance of the tree. Also make some searches for at least one value which is inside the tree and for one that isn't.

(1 points)

Question 2: Classes and Inheritance (12 points)

In this task you will implement classes for different sorting algorithms based on inheritance.

a) Create a header file `sort.h` declaring a class `Sort`, that should act like a Java interface, containing only purely virtual functions `getName` taking no arguments and returning a string, and `sort` taking and returning a vector of integers.

(1 points)

b) Create a class `CocktailShakerSort` that inherits from `Sort` and implements the cocktail shaker sort algorithm. The cocktail shaker sort algorithm works like a bidirectional bubble sort, where after every pass of the normal bubble sort, a pass in the other direction of the list is performed. The details can be found here:
`https://en.wikipedia.org/wiki/Cocktail_shaker_sort`

`getName` should return the name of the sorting algorithm and `sort` should return the sorted vector using your CocktailShakerSort implementation.

(4 points)

c) Create a class `QuickSort` that inherits from `Sort` and implements the quick sort algorithm. The choice of the pivot element can greatly influence the efficiency of the algorithm. For example, always chosing the last element of an already-sorted array will result in $O(n^2)$ performance. A common approach to resolve this issue, is to first shuffle the array. More details on the exact implementation possibilities can be found here:
`https://en.wikipedia.org/wiki/Quicksort`

`getName` should return the name of the sorting algorithm and `sort` should return the sorted vector using your QuickSort implementation.

(4 points)

d) Write a main function to test your sorting classes.

Sort a vector of at least length 20 that is not already sorted and print the result to the standard output.

(1 points)

e) Functors in C++ are classes and structs that overload the `()` operator and can therefore be called like a function even though they are variables.

In this task you will expand your sorting classes to be functors, so they can be used like functions. If you were unable to complete the previous exercise you can implement a new class and use the C++ sorting method `sort`.

Extend your classes `CocktailShakerSort` and `QuickSort` with the `()` operator such that they can be used as indicated in the following example code:

```
#include <vector>

#include "CocktailShakerSort.h"
#include "QuickSort.h"

using namespace std;

int main() {
    CocktailShakerSort shakerSort;
    QuickSort quickSort;

    vector<int> list;

    // Fill vector here

    vector<int> sortedShaker = shakerSort(list);
    vector<int> sortedQuick = quickSort(list);

    cout << sortedShaker << endl;
    cout << sortedQuick << endl;

    return 0;
}
```

(2 points)

Question 3: C++ Templates (6 points)

In question one of this exercise you have implemented a AVL tree for zip codes and their places. If it turned out that you needed a AVL tree which has the year as a key and the average temperature (as double) of this given year as value in addition to the AVL tree for zip codes then a naive approach would be to write an entirely new class and copy-paste the AVL tree and replace the `string` value with a `double` value. Of course, solving the problem this way is very bad coding practice, which is why templates exist in C++ – a concept for generic programming.

A *generic class* is a class in which variables and function return types can be defined to be template types. These template types can only be used in ways any type could be used. Because of this, the compiler can substitute in the right type and create individual classes for different types without the programmer having to write the same code multiple times.

Write a generic version of the `AVLNode` and the `AVLTree` class from question one and test your implementation in a main function as described in question 1 `c)`. You can use made up values for your test, but use atleast 8 values.

Question 4: Python Go (12 points)

In this task you have to implement a simplified version of Go on the command line written **entirely** in the Python programming language (`https://en.wikipedia.org/wiki/Go_(game)`).

Write a Python program that allows you to play the game Go on the command line. You don't need to implement the Ko and Suicide rules. It is sufficient to only implement the basic rule (the rule of liberty) and area scoring to decide who's winning. A normal Go board has a grid size of 19x19. For our purposes, the smaller version of a 9x9 board suffices.

Your program should have the following features:

- The entire Go board should be printed after or before each turn.
- Players take turns to make moves.
- A player can place one of their pieces during their turn onto an intersection.
- After each placement of a stone, the rule of liberty should be checked and groups of stones that do not obey the rule have to be removed.
- A player cannot place pieces on already occupied intersections.
- Players input their moves during runtime through the command line.
- After each turn, the score of each player should be calculated via area scoring and the scores should be displayed.
- You do not have to program a win condition. It is sufficient to treat it as an endless game.

An example game session might start like this:

```
   A B C D E F G H I
0  + + + + + + + + +  0
1  + + + + + + + + +  1
2  + + + + + + + + +  2
3  + + + + + + + + +  3
4  + + + + + + + + +  4
5  + + + + + + + + +  5
6  + + + + + + + + +  6
7  + + + + + + + + +  7
8  + + + + + + + + +  8
   A B C D E F G H I

Scores: player 0: 0, player 1: 0
Current player: 0
Choose where to place your stone: E5
```

```
   A B C D E F G H I
0  + + + + + + + + +  0
1  + + + + + + + + +  1
2  + + + + + + + + +  2
3  + + + + + + + + +  3
4  + + + + + + + + +  4
5  + + + + 0 + + + +  5
6  + + + + + + + + +  6
7  + + + + + + + + +  7
8  + + + + + + + + +  8
   A B C D E F G H I

Scores: player 0: 5, player 1: 0
Current player: 1
Choose where to place your stone: D5
```

If you need help to get started with Python, take a look at the Python tutorial:
`https://docs.python.org/3/tutorial/index.html`

Question 5: The Ultimate Game (Optional) (0 points)

In addition to the grand sum of 0 points, you will also earn the tutor's respect for completing this optional exercise.

Extend your Go implementation from the previous task such that the full rules are implemented. As such, it should not be possible to repeat previous constellations (Ko rule).

Also, use colors on the command line instead of the passionless white on black that is standard. Or, if you are really ambitious, you may even code a GUI for the game. Be creative!