UNIVERSITÄT BASEL

Thorsten Möller
Salome Müller - salo.mueller@stud.unibas.ch
Maarten Schenk - maarten.schenk@unibas.ch

UNI
BASEL

Programming Paradigms – C++ FS 2020

Exercise 3 Due: 03.05.2020 23:59:59

Upload answers to the questions **and source code** before the deadline via courses.cs.unibas.ch. Due to the measures taken to curb the Coronavirus pandemic, programs do **not** have to be demonstrated during the exercise slots. Instead, for **every** task and subtask you **must explain** your solution in detail. If a task involves writing or completing some code, you **must** provide code that is **commented in detail** (how it works, things that need to be done to make it working, things that must be satisfied, and so on).

Also note, of all mandatory exercises given throughout the course, you must score at least 2/3 of the total sum of their points to be allowed to take 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 binary search tree based on a binary tree. If you don't know what a BST or a binary tree is or how they work, you can look them up here:
`https://en.wikipedia.org/wiki/Binary_search_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 `BSTNode` (with a header file) implementing a single node of a BST tree with a key as integer and a value as string.

Each element should contain a integer key, its string value 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.

(3 points)

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

1. A constructor taking the key/value pair and the pointer to the parent node of an BST 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 BST node as an argument.

(2 points)

c) Write a class `BSTTree` (with a header file) that uses the `BSTNode` 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 BST 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 `printBST` which prints the tree in a simplistic way.
- A destructor that makes sure all nodes are deleted.

(6 points)

d) Write a main method to test your implementation.

You should insert at least 10 different zip codes with their respective places and print 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 `GnomeSort` that inherits from `Sort` and implements the Gnome sort algorithm. The Gnome sorting algorithm is similar to insertion sort in that it works with one item at a time but gets the item to the proper place by a series of swaps, similar to a bubble sort. The details can be found here:
`https://en.wikipedia.org/wiki/Gnome_sort`

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

(4 points)

c) Create a class `MergeSort` that inherits from `Sort` and implements the merge sort algorithm. The merge sort is a divide and conquer algorithm. Conceptually, a merge

sort works in two steps, divide the unsorted list into n sublists, each containing only one element. Then repeatedly merge sublists to produce new sorted sublist until there is only one sublist remaining. More details on the exact implementation can be found here:
`https://en.wikipedia.org/wiki/Merge_sort`

`getName` should return the name of the sorting algorithm and `sort` should return the sorted vector using your MergeSort 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 `GnomeSort` and `MergeSort` with the `()` operator such that they can be used as indicated in the following example code:

```
#include <vector>

#include "GnomeSort.h"
#include "MergeSort.h"

using namespace std;

int main() {
    GnomeSort gnomeSort;
    MergeSort mergeSort;

    vector<int> list;

    // Fill vector here

    vector<int> sortedGnome = gnomeSort(list);
    vector<int> sortedMerge = mergeSort(list);

    cout << sortedGnome << endl;
    cout << sortedMerge << endl;

    return 0;
}
```

(2 points)

Question 3: C++ Templates (6 points)

In question one of this exercise you have implemented a BST tree for zip codes and their places. If it turned out that you needed a BST tree which has the year as a key and the average temperature (as double) of this given year as value in addition to the BST tree for zip codes then a naive approach would be to write an entirely new class and copy-paste the BST 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 `BSTNode` and the `BSTTree` 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 Battleships (12 points)

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

Write a Python program that allows you to play the game Battleship on the command line. You don't need to implement the announcement of a sunk ship.

Your program should have the following features:

- The entire battleship board should be printed before each turn.
- Players input their moves during runtime through the command line.
- Players take turns to make moves. Use the provided clear() method to clear the console and ask the current player for confirmation that only he is present before showing any board with visible ships.
- At the start of a session each player is asked to place his ships (use coordinates like A 1 E/S, E = eastswards, S = southward)
- A Player is able to continue if he successfully hits, until he misses.
- Use x as a hit, S as a shiptile, ˜ as water, o as a miss (see reference below).
- Every player has 5 ships, a carrier (5 tiles), a battleship (4 tiles), a destroyer (3 tiles), a submarine (3 tiles) and a patrol boat (2 tiles).
- On your board your own ships, the hits and misses of your enemy should always be visible.
- On your enemy's board should always be visible where you already have shoot (miss = o and hit = x). Of course you should not be able to see his ship placement!
- You do have to program a win condition. And should be checked after every move.

An example start game session might start like this:

```
   ---- YOUR BOARD ----  |   ---- ENEMY BOARD ----
   A B C D E F G H I J   |    A B C D E F G H I J
                         |
0  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~   | 0  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
1  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~   | 1  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
2  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~   | 2  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
3  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~   | 3  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
4  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~   | 4  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
5  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~   | 5  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
6  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~   | 6  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
7  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~   | 7  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
8  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~   | 8  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
9  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~   | 9  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
```

This is how it could look like during a game session:

```
   ---- YOUR BOARD ----  |   ---- ENEMY BOARD ----
   A B C D E F G H I J   |    A B C D E F G H I J
                         |
0  o ~ ~ ~ ~ ~ ~ ~ ~ ~   | 0  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
1  x ~ ~ ~ ~ ~ ~ ~ ~ ~   | 1  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
2  x ~ S ~ ~ S S S ~ ~   | 2  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
3  x ~ S ~ ~ ~ ~ ~ ~ ~   | 3  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
4  x ~ S ~ ~ ~ ~ ~ ~ ~   | 4  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
5  x ~ ~ ~ ~ ~ ~ ~ S S   | 5  ~ o x x x x o ~ ~ ~
6  ~ ~ ~ ~ o ~ ~ ~ ~ ~   | 6  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
7  ~ ~ ~ S S S S ~ ~ ~   | 7  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
8  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~   | 8  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
9  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~   | 9  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
```

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 battleship implementation from the previous task such that it announces a ship destruction and introduce a single player mode with a AI based on randomness or surprise us.

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!