UNIVERSITÄT BASEL

Lecturer: Thorsten Möller - **thorsten.moeller@unibas.ch**
Tutors: Miruna Muntean - **miruna.muntean@unibas.ch**
Mark Starzynski - **mark.starzynski@unibas.ch**

Programming Paradigms – C++ FS 2023

Exercise 2 Due: 30.04.2022 23:55:00

Upload your answers to the questions **and source code** on Adam before the deadline.

Text : For answers to questions, observations and explanations, we suggest writing them in LaTeX. Please hand-in your answers as a **single PDF** file (independent of what tools you use, LaTeX, Markdown etc.).

Source-Code : For coding exercises, the source-code must be provided and has to be **commented in detail** (e.g. how it works, how it is executed, comments on conditions to be satisfied).

Upload : Please archive multiple files into a **single compressed zip-file**. If you upload an updated version of your solutions, the file name should contain a clear and intuitive versioning number. Only the latest version will be graded.

Requisit : In order to take the final exam, you must score at least 2/3 of all available points throughout the mandatory exercises.

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: Set Implementation, Constructor, Destructor (15 points)

In this task you will implement a red-black tree, a special form of a binary search tree. If you do not know what a red-black tree is you can look it up here:

`https://en.wikipedia.org/wiki/Red%E2%80%93black_tree`
`https://en.wikipedia.org/wiki/Binary_tree`

We are going to use this tree to save zip codes with their respective places. If you feel up to the challenge, you can implement the whole tree on your own. However, for this task you are free to use a code reference and adjust it accordingly to your needs (as always credit sources). Your implementation needs to include the following:

a) Write a class `Node` implementing a single node of a red-black tree with a key as integer, a value as string and an integer color. Furthermore each element should contain `pointers` to the parent, the left child and the right child nodes. The key-value pair should be private and the node should provide the necessary public getter and setter functions, namely `getKey(), setKey(int k), getValue(), setValue(string s)`.

(3 points)

b) Write a class `RBTree` that uses the `Node` 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 red-black 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 to rebalance the tree if needed (insertion!).
- A method `printTree` which outputs a simplistic view of the balanced tree.
- Three constructors. A default constructor, a constructor that takes one key/-value pair and a constructor that takes an array of key and an array of the respective values (and builds a sorted tree with them).
- A destructor that makes sure all nodes are deleted.

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

(10 points)

c) Bonus Task: Implement a delete function for your tree as well.

(0 points)

d) Test all the functionality of your tree. You can use the provided lists:

```
int zips[14] = {1005, 9000, 1204, 6060, 5034, 8057, 8805, 2740, 3005, 4052, 3920, 4132, 6005, 3604};

string names[14] = {"Lausanne", "St. Gallen", "Geneve", "Sarnen", "Suhr", "Zurich", "Richterswil", "Moutier",
"Bern", "Basel", "Zermatt", "Muttenz", "Luzern", "Thun"};
```

(2 points)

Question 2: Classes and Inheritance (9 points)

In this task you will implement a simple vehicle management application.

a) Create a header file `Car.h` for a class `Car`, where the class `Car` has three private fields:

   `String driverLastName, String licensePlate, String color`

   For each of those fields there should be a getter and setter method. Write a method `printInfo` that writes the values of the three fields to the console.

   (2 points)

b) Create an instance `testCar` of the class `Car`. Set the values of the fields with the setter method and print your car to the console using `printInfo` method created before.

   (1 points)

c) Create a new class `Vehicle` with two private fields:

   `String producer, String carType`

   Implement appropriate getter and setter methods for those fields.

   (2 points)

d) Let `Car` inherit from `Vehicle` and extend your main program with the new fields.

   (1 points)

e) On the next page you'll find a small test program. Write down the output without running it. (PS: Disregard the memory leaks ☺)

```cpp
class C1 {
public:
    virtual void f1() {
        f2();
        cout << "C1 - f1" << endl;
    }
    virtual void f2() {
        cout << "C1 - f2" << endl;
    }
};
class C2 : public C1 {
public:
    void f1() {
        f2();
        cout << "C2 - f1" << endl;
    }
    void f2() {
        cout << "C2 - f2" << endl;
    }
    virtual void f3() {
        f1();
        cout << "C2 - f3" << endl;
    }
};
class C3 : public C2 {
public:
    void f1() {
        f2();
        cout << "C3 - f1" << endl;
    }
};
int main() {
    C1* aa = new C1();
    C2* bb = new C2();
    C1* ab = bb;
    C1* ac = new C3();

    aa->f1(); cout << "*********" << endl;
    bb->f1(); cout << "*********" << endl;
    bb->f2(); cout << "*********" << endl;
    ab->f2(); cout << "*********" << endl;
    bb->f3(); cout << "*********" << endl;
    ac->f1();
    return 0;
}
```

(3 points)

Question 3: Operator Overloading (10 points)

In this exercise you will implement your own complex number class, define complex calculation functions and overload various operators in order to be able to use them with your class.

a) Implement your own complex number class and define four functions that perform the complex addition, subtraction, multiplication and division. The method prototype for the operations could look like this:

```
COMPLEX operation(COMPLEX a, COMPLEX b);
```

You can use the following example to test your implementation:

$$(1+5i) - \frac{(5+3i)+(2-3i)}{(2-4i)} = (0.3+3.6i)$$

(4 points)

b) A more elegant and easily readable approach to the same problem would be to overload the for your class relevant operators. This would look like so:

```
COMPLEX a,b;

cout << "Operator overloading: " << endl;
cout << "-------------------------------------------" << endl;
cout << "Addition:        a + b = " << a + b << endl;
cout << "Subtraction:     a - b = " << a - b << endl << endl;
cout << "Multiplication:  a * b = " << a * b << endl;
cout << "Division:        a / b = " << a / b << endl;
cout << "Conjugation:     a = " << a << " and a! = " << !a << endl;
// Output example for conjugation: a = 7 + 5i and a! = 7 - 5i
```

Use operator overloading to enable all 4 basic operators for your class and additionally overload an unary operator for complex conjugation. Also keep in mind that in order to display your copmlex numbers properly in the console you need to overload the << operator as well.

(6 points)

Question 4: C++ Templates **(6 points)**

a) Write a generic version of a `dynamic array` class and test your implementation in a main function. Showcase your dynamic array implementation as a char, int and double array respectively.

(6 points)

b) Bonus Task: In question one of this exercise you have implemented a red-black tree for zip codes and their places. Write a generic version of the `Node` and the `RBTree` class from question one and test your implementation in a main function.

(0 points)

Question 5: Tic-Tac-Toe (Python) **(10 points)**

In this task you have to implement the game Tic-Tac-Toe [1] on the command line written **entirely** in the Python programming language.

Write a Python program that allows you to play the game on the command line.

Your program should have the following features:

- Starting player is chosen at random.
- If the board is filled completely without a winner, the game starts anew with the opposite player starting.
- Player can choose whether to play with another human player or against a computer AI.
- How you implement the AI is up to you, at its simplest you can make the computer play at random.
- Game conditions like restart, quit game, change mode (AI or 2nd player) should be implemented.
- Inputting the field of choice can be done via the numpad, e.g. for the lower left field you can press `1+ENTER` and for the middle field you press `5+ENTER` etc (sum of row and column number).

Game session start :

```
Turn of Player X
    1   2   3
  -------------
6 |   |   |   |
  ----+---+----
3 |   |   |   |
  ----+---+----
0 |   |   |   |
  -------------
```

Within a game session:

```
Turn of Player O
    1   2   3
  -------------
6 |   | X | O |
  ----+---+----
3 |   |   | X |
  ----+---+----
0 |   |   |   |
  -------------
```

[1]**https://en.wikipedia.org/wiki/Tic-tac-toe**

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 6: 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 previous game implementation with your own GUI or add colors to the command line to make it look prettier.