Exercise 2.1 Set Implementation, Constructor, Destructor

Files for this Question:
RBTree/Node.cpp
RBTree/RBTree.cpp
RBTree/Insert.cpp
RBTree/Search.cpp
RBTree/PrintTree.cpp
(and the corresponding .h files)

To compile (on MacOS):

```
g++ RBTree.cpp Node.cpp Insert.cpp Search.cpp PrintTree.cpp -o RBTree
```

Output:
It should print the tree three times (constructed with the three different constructors) and print out the search result of 4052, 4132 and 5050.
The tree is printed from left to right. So the root node is the leftmost node while the leaves are on the right side. It also prints B or R at every node indicating its color.

Sources / Credits:
All code is written by myself. I used this website: geeksforgeeks.org/insertion-in-red-black-tree as basis for my implementation and rewrote the given Java code in C++.

Exercise 2.2 Classes and Inheritance

Files for this Question:
cars/Car.cpp
cars/Vehicle.cpp
(and the corresponding .h files)

To compile (on MacOS):
```
g++ Car.cpp Vehicle.cpp -o car
```

Output of part a-d:
```
--- CAR INFO ---
Driver: Torvalds
Plate: Mr. Linux, DAD OF 3, KING OF GEEKS
Color: Yellow
Producer: Mercedes Benz
Type: SLK
-----------------
```

Output of part e:
```
C1 - f2
C1 - f1
*********
```
–> `aa` is of Type `C1*`, so `aa->f1()` calls `C1::f1()` which calls `C1::f2()`

```
C2 - f2
C2 - f1
*********
```
–> `bb` is of Type `C2*`, so `bb->f1()` calls `C2::f1()` which calls `C2::f2()`

```
C2 - f2
*********
```
–> `bb` is of Type `C2*`, so `bb->f2()` calls `C2::f2()`

```
C2 - f2
*********
```
–> `ab` is of Type `C1*` but points to an object of `C2*`, so `ab->f2()` calls `C1::f2()` but because that is a virtual function it calls the correct implementation based on the dynamic object type which is `C2` so in the end `C2::f2()` is called. (static type is C1*, but dynamic type is C2*)

```
C2 - f2
C2 - f1
C2 - f3
*********
```
–> `bb` is of Type `C2*` so `bb->f3()` calls `C2::f3()` which calls `C2::f1()` which inside calls `C2::f1()`.

```
C2 - f2
C3 - f1
```
–> `ac` is of Type `C1*` but points to an object of `C3`. So `ac->f1()` calls `C3::f1()` because its dynamic type is C3. `C3::f1()` then calls `f2()` which is not implemented in `C3` and because the static type is `C1`, it goes up one step and uses the implementation of its parent: `C2::f2()`

Exercise 2.3 Operator Overloading

Files for this Question:
Overloading/Complex.cpp
(and the corresponding .h file)

I am aware the it is technically possible to divide by 0 in this implementation. To do that however, the user needs to do the error of actively trying to divide by 0. The program itself does not crash and correctly returns NaN when asked to divide by 0 which is the important part!

Exercise 2.4 Templates

Files for this Question:
Templates/DynamicArray.h
Templates/main.cpp

The Code for the Template DynamicArray is only generated when its instantiated for a specific type. So the compiler needs to have access to the full definition of the Template whenever he needs to instantiate it for some type. Therefore the full definition has to either be in the .h already, or the .h file needs to `#include` the corresponding .cpp file at the end. Here I went with the first option.

To compile:
```
g++ main.cpp -o main
```

Read comments in the code for more details on the functionality of the DynamicArray.

Exercise 5 and 6 Tic Tac Toe, The Ultimate Game

Files for this Question:
tictactoe/tictactoe.py

Run: `python3 tictactoe.py`

The Game is self-explanatory in the GUI. It is still possible to play in the terminal if you wish.

Commands for the commandline:

`quit`	->	Quit the game.
`mode`	->	Switch between AI and 2 Player mode.
`restart`	->	Restart the game.
`[1-9]`	->	Mark the field. 1 is bottom right, 9 is top left.