Programming in C++ I, Rotation 4
Jacobs University Bremen
Dr. Kinga Lipskoch

Course: CH08-320142
November 22nd, 2018

Assignment 3 - Classes and Constructors

- The problems of this assignment must be solved in C++.
- The TAs are grading solutions to the problems according to the following criteria: https://grader.eecs.jacobs-university.de/courses/320142/2018_2r2/Grading-Criteria-C++.pdf

Problem 3.1 *A `City` class* (1 point)

Presence assignment, due by 18:30 h today
Create a class named `City`. Assume that a city has a name, a number of citizens (inhabitants of the city), a location (belonging to a country) and a POI (point of interest).
Then create three instances of this class with the name property containing: *Bremen*, *Hamburg* and *Berlin*. Provide suitable setter and getter methods for each property. The class declaration has to be placed into `City.h`, the class definition has to be placed into `City.cpp` and the test program where the instances are created has to be in `testcity.cpp`.
You can set the needed data from the `main()` function or read it from the keyboard.

Problem 3.2 *Constructors for `Critter`* (1 point)

Add three constructors to the class `Critter`. Each constructor should also print a simple informational message on the screen such that one can see when and which constructor has been called.
You should be able to create an instance of the `Critter` class using three types of constructors: 1) without supplying any properties (which should set the name to “`critter`”, the height to 10 and the rest to 0), 2) by only supplying a name as parameter (which should set the height to 10 and the rest to 0), and also 3) by supplying *name*, *hunger*, *boredom* and *height* all as parameters. You should also be able to create an instance of the `Critter` class without specifying the *height*. If the *height* is not supplied, the critter should have the default height of 15.
Write a test program which creates four instances of the `Critter` by using these three different constructors (the last one in two ways). Set their *hunger* levels to 5 by using appropriate method and/or constructor calls. The critters’ properties should then be printed on the screen.
Name the files `Critter.h`, `Critter.cpp` and `testcritter.cpp`.

Problem 3.3 *Information hiding I* (1 point)

A game developer crew has decided to rather use a percentage scale (double value between 0.0 and 1.0) to represent the *hunger* level of a critter. Change the internal structure of the class to reflect this. However, your **existing test program should run without any modifications** (therefore the public class interface stays the same) and you will need to find a way to convert the current *hunger* levels from integer values (which are from 0 to 10) to doubles and then from doubles back to integers. Use separate methods for doing this which will be called in constructors and other methods.
Use a simple mapping scheme like 10 is 100%, 9 is 90%, 8 is 80%, . . . , 1 is 10%, and 0 is 0%.
Name the files `Critter.h`, `Critter.cpp` and `testcritter.cpp` (**must remain unchanged**).
The implementation for the conversions needs to be put into `Critter.cpp` and should not be part of the public interface.
The client class `testcritter.cpp` from the previous problem **must remain unchanged**. The *hunger* levels of the critters should be “internally” at 50%.
You can assume that the setting values are valid.

Problem 3.4 *Information hiding II* (1 point)

Next a *thirst* level (as double value) should be added to the properties of a critter. Add a new constructor that takes five parameters for setting all properties of a critter. Make sure that the existing constructors will still work. For the existing constructors, the *thirst* level should be set to the same level as the *hunger* level.

Your existing `testcritter.cpp` must still be able to run **in its unchanged form**. So the already existing constructors need to support the change. Name the files `Critter.h`, `Critter.cpp` and `testcritter.cpp`.
Finally, you should adapt the print method for printing on the screen also the value of the thirst level as a double. The client program `testcritter.cpp` may contain two additional lines, where the constructor taking five parameters is being called and the object is printed.
You can assume that the setting values are valid.

Problem 3.5 *Copy constructor* (1 point)

Download the file:
`https://grader.eecs.jacobs-university.de/courses/320142/cpp/copyconstructor.cpp`
Based on the source code of `copyconstructor.cpp` implement the method `funcByref()`. Change all constructors (including the copy constructor) such that **you can clearly see when and which of them is invoked** by adding a message which is printed on the screen.
Then in your `main()` function create at least two objects using the different constructors, call `funcByVal()`, `funcByRef()`, and print the results on the screen. Then make sure that the memory occupied by the objects will be released by the end of the program.
You can assume that the setting values are valid.

Problem 3.6 *A `Complex` class* (2 points)

Create a class named `Complex` for storing and managing complex numbers. A complex number has an real part and an imaginary part. The class has to provide a default constructor initializing the properties by 0, another constructor for setting the properties with specific values and an empty destructor. Provide suitable setter and getter methods for each property and a method for printing the complex number on the screen in its mathematical form (e.g., $1 + 2i$, $3 - 5i$). Also provide methods for the conjugation of a complex number, and for adding, subtracting and multiplying two complex numbers. The class declaration has to be placed into `Complex.h`, the class definition has to be placed into `Complex.cpp` and the test program where the instances are created has to be in `testcomplex.cpp`. The test program should create at least two instances of the `Complex` class, the data for the properties should be read from the keyboard.
Then:

a) the conjugate of the first instance should be determined and printed on the screen;

b) the sum of the two instances should be determined and printed on the screen;

c) the difference between the first and second instance (in this order) should be determined and printed on the screen;

d) the multiplication of the two instances should be determined and printed on the screen.

The prototypes of the methods for adding, subtracting and multiplying must have the following form:

```
Complex Complex::add(Complex);
```

Then the usage will be the following:

```
Complex c1, c2, c3;
...
c3 = c1.add(c2);
```

Note: If $z = a + bi$ then $\overline{z} = a - bi$. If $z_1 = a + bi$ and $z_2 = c + di$ then $z_1 + z_2 = (a + c) + (b + d)i$, $z_1 - z_2 = (a - c) + (b - d)i$ and $z_1 \cdot z_2 = (a \cdot c - b \cdot d) + (b \cdot c + a \cdot d)i$.

You can assume that the input will be valid.

How to submit your solutions

- Your source code should be properly indented and compile with `g++` without any warnings (You can use `g++ -Wall -o program program.cpp`). Insert suitable comments (not on every line . . .) to explain what your program does.
- Please name the programs according to the suggested filenames (they should match the description of the problem) in Grader.

  Each program **must** include a comment on the top like the following:

  ```
  /*
     CH08-320142
     a3_p1.cpp
     Firstname Lastname
     myemail@jacobs-university.de
  */
  ```
- You have to submit your solutions via *Grader* at

  `https://grader.eecs.jacobs-university.de`.

  If there are problems (but **only** then) you can submit the programs by sending mail to `k.lipskoch@jacobs-university.de` **with a subject line that begins with CH08-320142**.

  It is important that you do begin your subject with the coursenumber, otherwise I might have problems to identify your submission.
- Please note, that after the deadline it will not be possible to submit any solutions. It is useless to send late solutions by mail, because they will not be accepted.

This assignment is due by Tuesday, November 27th, 10:00 AM.