Assignment 4 - Dynamic Memory Allocation and Inheritance

- 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 4.1 *Boxes* (1 point)

Presence assignment, due by 18:30 h today

Design and write an object-oriented program for the computation of the volume of boxes (having a height, a width and a depth). Your solution should have methods for setting and getting the height, the width and the depth of a box as well as constructors (an empty constructor and one which sets the data members) and a destructor.
Name your files `testbox.cpp`, `Box.cpp` and `Box.h`.
The program testing your code (`testbox.cpp`) should do the following:

a) Enter from the keyboard the number of boxes that will be entered.

b) Dynamically create an array of boxes.

c) Enter from the keyboard the height, width and depth of each box one after the other.

d) Loop across all boxes, calculate and print on the screen the volume of each box.

e) Print on the screen the total volume of all boxes (sum of all volumes).

You can assume that the input will be valid.

Problem 4.2 *Inheritance with creatures* (1 point)

Download the file:
`https://grader.eecs.jacobs-university.de/courses/320142/cpp/creature.cpp`
Extend the example program `creature.cpp` by adding two other type of creatures (i.e., two new classes derived from `Creature`) with some example properties and methods (use your imagination for doing this). Use/implement at least one property and at least one method for each derived class.
Each constructor, destructor and method should be implemented in such a way that each call is being documented via messages printed on the screen.
Create one instance of `Wizard`, two instances of the two other classes, and call for each object methods of the particular instances (as in the original example).
You can assume that the setting values are valid.

Problem 4.3 *Inheritance split code* (1 point)

Reorganize your previous file's (`creature.cpp`) content into three parts: declaration, implementation of the classes and your test program. Name the files `Creature.h`, `Creature.cpp` and `testcreature.cpp`.

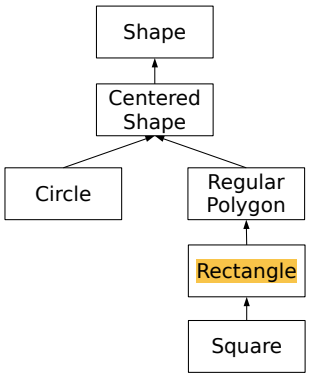
Problem 4.4 *Inheritance and pointers* (1 point)

Change your previous testing program (`testcreature.cpp`) such that it runs in an endless loop and waits for input from the keyboard. If '`w`', '`1`', or '`2`' is entered, a wizard, your other `object1` or your other `object2` should be dynamically created (via `new`), the corresponding method is being called and the object is then destroyed (via `delete`). Entering '`q`' should stop the execution of your loop using `break`.

Name the files `Creature.h` (same as above), `Creature.cpp` (same as above) and `dyncreature.cpp`.
You can assume that the input will be valid.

Problem 4.5 *Shapes* (1 point)

Download the files:
`https://grader.eecs.jacobs-university.de/courses/320142/cpp/Shapes.h`
`https://grader.eecs.jacobs-university.de/courses/320142/cpp/Shapes.cpp`
`https://grader.eecs.jacobs-university.de/courses/320142/cpp/testshapes.cpp`
Extend the `Shapes` example by adding two new classes `Rectangle` and `Square` according to the following inheritance diagram.



Implement the following:

a) default constructors for all classes by initializing the properties to default values,

b) setter and getters for all classes,

c) the constructors `Rectangle(const string& n, double nx, double ny, double nwidth, double nheight)` and `Square(const string& n, double nx, double ny, double nside)`,

d) copy constructors for all classes,

e) the methods `double perimeter()` and `double area()` for the classes `Circle`, `Rectangle` and `Square` for returning the perimeter and area of corresponding instances,

f) in your testing program create instances of `Circle`, `Rectangle` and `Square`, and then compute and print on the screen their perimeter and area.

Name your files `Shapes.h`, `Shapes.cpp` and `testshapes.cpp`.
You can assume that the setting values are valid.

Bonus Problem 4.6 *A `Vector` class for doubles* (2 points)

Create a class named `Vector` for storing and managing vectors of doubles. The properties of a vector are the size of the vector and a `pointer to a double` pointing to a memory location where the components of the vector are stored. The class has to provide a default constructor, another constructor for setting the properties, a copy constructor and a destructor. The non-default constructors should dynamically allocate memory for a vector and the destructor should release it. Provide suitable setter and getter methods for each property and a method for printing the components of a vector on the screen separated by spaces. Also provide methods for computing the norm of a vector, the sum, the difference and the scalar product of two vectors. The class declaration has to be placed into `Vector.h`, the class definition has to be placed into `Vector.cpp` and the test program where the instances are created has to be in `testvector.cpp`. The test program should create three instances of the `Vector` class using the different constructors (the second instance should be the copy of the first one). Then the norm of the first vector should be computed and printed on the screen, the scalar product, the sum and the difference of the first and third vectors (in this order) should be also computed and printed on the screen.

Use the `const` modifier properly for parameters and methods.
The prototypes should have the following form:

```
double norm() const;
Vector add(const Vector) const;
```

The usage should be:

```
Vector v1, v2;
(v1.add(v2)).print();
```

Note: If $v_1 = (a_1, a_2, \ldots, a_n)$ and $v_2 = (b_1, b_2, \ldots, b_n)$ with $n \in \mathbb{N}^*$. Then:
$\|v_1\| = \sqrt{\sum_{i=1}^{n} a_i^2}$ (the norm of a vector),
$v_1 \cdot v_2 = \sum_{i=1}^{n} a_i \cdot b_i$ (the scalar product of two vectors),
$v_1 + v_2 = (a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n)$, and
$v_1 - v_2 = (a_1 - b_1, a_2 - b_2, \ldots, a_n - b_n)$.

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
   a4_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 Wednesday, November 28th, 10:00 AM.