Object-oriented and softw. design - C++

Fall 2025

Lab worksheet #5

19.11.2025

*** In these exercises, you will only use the notions of C++ presented up to lecture 5 ***

A **submission** is requested for this session.

- **Working in teams:** You **have to form teams of 3 students**, and you can only form teams of less than 3 if no further students can join your team.

  Important: You're not allowed to work with the same team members as you did for the first submission. You have to **form teams with different students**!

- **Files to submit:** Each submission consists of 2 parts:
  - **Source code:** Its **ease of use** and **readability** of the code will be part of the grading. It must be possible to compile all the code from the main directory in the submitted file just by typing "make".
  - **Presentation slides:** Write on the 1st page of the presentation slides **who of you contributed to which parts of this worksheet**. You will only get points if you contribute enough.
    The **submission format** of the report is solely a **.pdf** file (no Libreoffice/Word or any other format!).
  - **Tarball:** Provide the **report** as well as the **source code** within a **single .zip** or **.tar.bz2** file using **well structured directories** within it.
- **Grading:**
  - **Regarding the source code**, also its **ease of use** and **readability** of the code will be part of the grading. It's better to have fewer things working correctly rather than having many things "done", but not working correctly.
  - **Regarding the presentation**, create **presentation slides** with up to 10 slides briefly describing what you did. The writing quality, the quality of figures, source code, and readability will be taken into account in the grading. The presentation should be self-explanatory without the source code. **Grammar and spelling will be taken into account**. There are no excuses for not using at least a spellchecker!
- **Submission:** Only submissions via the course webpage will be accepted. Submissions via Email are **not allowed** unless there are any technical issues at (and only at) UGA's site. You can do multiple uploads, and the **last upload will be used for grading**.
- **Deadline:** The deadline for the submission is **03.12.2025**, one minute before midnight. This is a **strict** deadline!

The boss of IM2AG bank was quite happy with your banking software and discussed this with all CEOs. They are now interested in some feature upgrades.

1) Polymorphism

The classes that represent the banking accounts should be improved:

- First, we like to take into consideration that **debiting an amount of money from an account of the most general class cannot be defined** (no relevant definition), this functionality is only relevant for current accounts and savings accounts.
- Second, it should be possible to **display the attributes** of any kind of account using the stream operator <<.

This can be realized with features known as polymorphism.

- Recap on why polymorphism is useful in this case.
- Which declarations and definitions must be modified, and why?
- Modify the files `account.h` and `account.cpp` accordingly and test this new version.

2) Client update

The IM2AG bank wants to offer more flexibility to its customers. Therefore, the client's data should be partly extended to include the following data:

- The **name** (a string)
- The **identifier** (an unsigned integer)
- A **dynamic array of pointers** (**not** an `std::array`!) to **multiple** accounts (whatever the type of the account is)
- The **maximum** number of accounts
- The **current** number of accounts.

Whenever a new client is created, the name, identifier, and maximum number of accounts will be given.

For every client, a method `createAccount` will allow the creation and insertion of a new account (of any type, specified interactively by the user) in the array. Every new account will have a balance of 0.

It should be also possible to:

- **Credit** any of the accounts given by an index in the array of an amount of money X.
- Try to **debit** any of the accounts given by an index with an amount of money X and display the attributes (including the attributes of his accounts).

Here is an example of expected interaction for the creation of accounts for Smith and displaying the attributes of Smith:

```
1000 Creation of accounts for Smith:
     Creation of a current account(1), unblocked savings account(2), or blocked savings account(3)?
1002 2
     Creation of a current account(1), unblocked savings account(2), or blocked savings account(3)?
1004 1

1006 Client name: Smith and id = 1234 and his/her accounts :
```

```
** Savings account **
Account number = 0, balance = 0 euros
and Interest rate = 2%
** Current account **
Account number = 1, balance = 0 euros
```

You will realize this in the following assignments.

2.1) Declaration

Give the declaration of this new class `Client`: modify/add attributes and methods.

2.2) Methods and functions

Define all the methods and functions declared before.

2.3) Copy constructor

Define a copy constructor for the class `Client`.

2.4) Update of main

The class `Bank` is the same as in the previous lab worksheet. Adapt your main function to use all your new classes.

2.5) Input / output

Inputs/outputs can also be performed using files. To deal with files, output and input streams of types `std::ofstream` and `std::ifstream` are used (note that it is necessary to include `<fstream>`).

A file is opened using the `open` method and closed using the `close` method.

Look at the documentation about text files given here: `http://www.cplusplus.com/doc/tutorial/files/` and adapt the `main` function to write the output to a file.

3) Polynomials

(Exercise 16 of lecture notes)

This exercise aims to deal with vectors of heterogeneous functions of one variable (for example, polynomials). We will define a general class `Function`, and concrete classes (`Poly0`, `Poly1`, ...) which inherit from it.

3.1) Function

Define a class `Function` with an attribute `name` which is a string. This should help identify the function type (e.g. “cosine”, “polynomial of degree 1”).

This class also contains the following methods:

- `eval` computing the value of the function at some point x_0,
- `derivative` returning a pointer to a newly allocated `Function` class representing the

derivative, and
- `print` enabling the display of the characteristics of a class instance.

3.2) Polynomials

From this general class, derive specific subclasses to represent polynomials:

- class `Poly0` represents $p(x) = k_0$,
- class `Poly1` represents $p(x) = k_0 + k_1 x$, and
- class `Poly2` represents $p(x) = k_0 + k_1 x + k_2 x^2$

Hint: Consider that `Poly0` includes the null polynomial.

3.3) Testing

Define a `main` function in which you declare a vector of pointers to `Function`, and you insert the following functions:

- $p(x) = 6$
- $p(x) = 2 + 5x$
- $p(x) = 10 + 3x + 2x^2$

Then, you iterate on this vector to

- display the characteristics of each polynomial, and
- evaluate its derivative at point 1.

You should get the following result:

```
$ ./exercise3
++ Content of the vector ++
This function is a Polynomial of degree 0
p(x) = 6
The value of its derivative at point x=1 is 0
This function is a Polynomial of degree 1
p(x) = 2 + 5.x
The value of its derivative at point x=1 is 5
This function is a Polynomial of degree 2
p(x) = 10 + 3.x + 2.x^2
The value of its derivative at point x=1 is 7
```