Object-oriented and softw. design - C++

Fall 2025

Lab worksheet #2

01.10.2025

*** In these exercises, you will only use the notions of C++ presented up to lecture 2 ***

This worksheet provides the fundament for the next worksheets. After the lab session is over, make sure that you understand these concepts and that you can develop code using them. **If you do not understand them, practice it!** Otherwise, you will have more and more trouble with the next labs.

1) References vs. pointers

In this first part, we will familiarize ourselves with the difference between references, pointers, and values.

1.1) Program output

Which outcome do you expect for the execution of the simple program below and why?

```
1000 #include <iostream>

1002 int make_computation(int x) {
       int i;
1004   for (i=1; i<=4; i++)
         x = x * i;
1006   return x;
     }
1008
     int main(void) {
1010   int k;
       printf("Give an integer for the computation  ");
1012   std::cin >> k;
       std::cout << "Function make_computation returns " <<
        make_computation(k) << " and the value of k is " << k << std::
        endl;
1014   return 0;
     }
```

Transfer this program to a C++ file, compile it, and execute it. Did you get what you expected?

1.2) Pass-by-reference

Modify this program to use *pass-by-reference* for the parameter of `make_computation`.

What is now the outcome? How do you explain that?

1.3) Pass-by-pointer

Modify the previous program so that the function `make_computation` receives a *pointer to integer* as a parameter.

What is the outcome? How do you explain that?

2) Lab sheet 1 and pass-by-reference

This exercise is an extension of exercises 1 and 2 of lab 1 where we only used pass-by-value. This has several drawbacks (e.g., classes are always copied) and should be avoided in many cases (unless you know what you're doing). These assignments will be about updating the previous lab sheet to pass-by-reference, which should improve the computational performance significantly.

2.1) Date

Modify the class `Date` so that it has an additional attribute, which is a `char*` storing the day of the week ("Monday", "Tuesday", ...). (For testing purposes, set this pointer to an arbitrary weekday depending on the day of the month (which does not need to be the right one; see the following assignment). This is to test the implementation.)

Create a **copy constructor** and a **destructor** for this class (use functions `strlen(...)` and `strcpy(...)`).

2.2) Accurate dates (optional)

We do not care about the realistic solution of our weekdays and accurate days of the month since these assignments are intended to familiarize us with object-oriented programming in C++. Anyhow, if you like to implement more realistic solutions, take a look at the function `mktime` (see here: `http://www.cplusplus.com/reference/ctime/mktime/`)

2.3) Pass-by-reference, step 1

Based on the previous worksheet, modify the independent functions `before`, `difference`, and `duration` so that they use pass-by-reference parameters (instead of pass-by-value).

2.4) Pass-by-reference, step 2

Adapt the entire application accordingly and compare the creation of objects in the case of pass-by-reference and the case of pass-by-value.

How many times - and when - will the copy constructor and destructor be called, and why?

Use text output in the constructors to identify these cases!

3) Points

In this exercise, we will familiarize ourselves with using pointers in C/C++. These examples are purely artificial but are intended to help you understand pointers and how they are used in C++.

3.1) PPoint

Define a class `PPoint`.

3.1.a) Attributes

Add two attributes *given by pointers to integers* **`x`** *and* **`y`** (coordinates of the point).

Ensure these are pointers to integers since we like to practice pointer utilization!

3.1.b) Constructor

Create a constructor that takes as arguments *two integers* for initialization of these coordinates.

Since we have pointers to coordinates, you also need to allocate memory in the constructor to store the values of both coordinates.

3.1.c) Typeid

Modify this constructor so that it also prints the type name of the class instance that is being created. This is the occasion to become familiar with the `typeid` operator:

This operator can take a variable as an argument (use it here with the instance being built) and returns an instance of class `type_info`, which stores information about its type. This class has several methods, among them a method `name()` that returns a character string representing the implementation-defined name of the type.

Note that you must include `<typeinfo>` to use `typeid`.

More information is available at `https://en.cppreference.com/w/cpp/language/typeid`.

3.1.d) Copy constructor & destructor

Create a copy constructor and a destructor for this class.

Make sure that the destructor also frees all previously allocated memory!

3.1.e) Setting of coordinates

Write an additional method `set_coordinates` in the class `PPoint` with two arguments x and y. This method should set the (x, y) coordinates of the point to those provided as arguments.

3.1.f) Adding coordinates of two points

Create a method `add` with an argument of type `PPoint`. This method should element-wise add the coordinates of the point in the argument to the current point.

E.g., $(x^{point1}, y^{point1}) := (x^{point1}, y^{point1}) + (x^{point2}, y^{point2})$.

3.1.g) Print

Create a method `print` that prints the point.

3.2) Array of points

Define a class `Array_of_PPoint`.

3.2.a) Member variables/attributes

Add the following attributes: a dynamic array of `PPoint`, and an `unsigned integer` that represents the length of this array.

3.2.b) Constructor

Create a constructor that takes as arguments an `unsigned integer len` and allocates memory for an array of length `len`. Do you need to modify something regarding the constructor(s) of class `PPoint`? What and why?

3.2.c) Initialization

Now, we want to improve the constructor of class `Array_of_PPoint` so that it also initializes the coordinates of each element of the array with random values, for instance, in $[0, 5] \times [0, 5]$.

Use the `set_coordinate` function of `PPoint` in the constructor of class `Array_of_PPoint` to perform assignments to random values for the coordinates of the elements of the newly created array.

Use the random number generator `rand`, see also `https://www.tutorialspoint.com/c_standard_library/c_function_rand.htm` and the lecture.

3.2.d) Add

Create a method add of `Array_of_PPoint` with argument of type `PPoint` that adds the coordinates of the argument to every point stored in the array of points.

3.2.e) Print tabs

Add a method `print_tabs` to print the content of the array by also using tabs for a nice-looking layout.

3.3) Calling constructors and destructors

Extend the copy constructor and destructor to print messages on the screen (for instance, "`Calling copy constructor`" and "`Calling destructor`" respectively).

How many messages will be printed when executing the following main function, according to your definition of `add`?

```
1000 int main(){
       Array_of_PPoint a(4);
1002   PPoint p(2,6);
       a.add(p);
1004   a.print_tab();
```

```
  return 0;
}
```

3.4) Cleanups and Makefile

Please clean up your code, add a Makefile, etc.

Don't forget to add a `clean` target that removes all object file and the executable.

4) Code documentation (with Doxygen)

Documentation is the backbone of future-proof software development. Software that is not well documented might need to be replaced entirely in the future since it might not be understandable anymore by those who did not develop the code.

Therefore, we would like to get familiar with code documentation.

4.1) Warmup

Start documenting the code of exercise 3 using the conventions of the Doxgen tool (see `http://www.doxygen.nl/`). The tool Doxygen supports the automatic generation of (HTML, Latex, etc.) documentation for code written in C, C++, and many other languages.

4.2) Step 1

Add a target `doc:` to your `Makefile` to automatically generate the documentation in a `doc/` directory.

4.3) Step 2

Comments which are written in a special style in the code are processed by Doxygen, e.g.,

```
#ifndef _CONVERT_H_
#define _CONVERT_H_

/**
 *\file convert.h                    // DO NOT FORGET THIS!!!
 *\brief Conversion functions
 */

#define RATE 1.28
#define EURO 0
#define DOLLAR 1

/**
 *\fn void dollar_euro(double dollar)
 *\brief Conversion function dollar -> euro
 *\param dollar value in dollars to be converted
 *\return returns nothing, displays the value in euros
 */
```

```
1018 void dollar_euro(double dollar);

1020 ...
```

For more details, see the Doxygen website or `https://www.star.bnl.gov/public/comp/sofi/doxygen/docblocks.html`

4.4) Step 2: Configuration file

Generate the configuration file e.g. by calling

```
1000 $ doxygen -g doxy-convert.conf
```

and edit this file correspondingly

4.4.a) Step 3: Execution of Doxygen

Execute Doxygen

```
1000 $ doxygen doxy-convert.conf
```


5) Exceptions (optional)

Exceptions are a way to cope with situations that should rarely occur, e.g., in case of an error. Some people like exceptions in C++, and some don't. Certain companies (G...e) also avoid using exceptions in C++ since they are not directly compatible with other languages. If you write code purely in C++, they can be highly beneficial.

5.1) Overview of exceptions

C++ supports the notion of exceptions. Read the brief overview at `https://www.tutorialspoint.com/cplusplus/cpp_exceptions_handling.htm`.

5.2) Exercise

Apply these notions in the classes of the "Points exercise" as follows:

- Modify the method `add` of class `PPoint` so that it throws the exception "`unexpected value`" if at least one of the coordinates of the computed result is not inside $[0, 20]$.
- Modify the method `add` of class `Array_of_PPoint` accordingly.
- Check this solution with a `main` that creates an `Array_of_PPoint` and adds a `PPoint` to it.

6) Homework for next time

Finish the exercises.