Object-oriented and softw. design - C++

Fall 2025

Lab worksheet #3

15.10.2025

*** In these exercises, you will only use the notions of C++ presented up to lecture 3 ***

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

- **Working in teams:** You **have to form teams of 2 or 3 students**. I you attended not the 1st class, you have to work on this worksheet on your own.
  Heads up: There will be a **2nd graded lab sheet** where you have to **form teams with students different to this group**!
- **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 **04.11.2025**, one minute before midnight. This is a **strict** deadline!

© Université Grenoble Alpes - 2025

1) Limitations of integer encoding

Write a simple C++ (or C) program that takes as an argument a positive integer in `x`, computes and prints the value of `factorial(x)`, for example:

```
$ my_factorial 6
The value of factorial(6) is 720
```

1.1) Integer limits

Up to which value of its argument does this application compute the expected result? Can you explain why?

Feel free to use the following information:

Factorials

x	x!
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800
11	39916800
12	479001600
13	6227020800
14	87178291200
15	1307674368000

Powers of 2

$2^0 = 1$	$2^{10} = 1\,024$	$2^{20} = 1\,048\,576$	$2^{30} = 1\,073\,741\,824$
$2^1 = 2$	$2^{11} = 2\,048$	$2^{21} = 2\,097\,152$	$2^{31} = 2\,147\,483\,648$
$2^2 = 4$	$2^{12} = 4\,096$	$2^{22} = 4\,194\,304$	$2^{32} = 4\,294\,967\,296$
$2^3 = 8$	$2^{13} = 8\,192$	$2^{23} = 8\,388\,608$	$2^{33} = 8\,589\,934\,592$
$2^4 = 16$	$2^{14} = 16\,384$	$2^{24} = 16\,777\,216$	$2^{34} = 17\,179\,869\,184$
$2^5 = 32$	$2^{15} = 32\,768$	$2^{25} = 33\,554\,432$	$2^{35} = 34\,359\,738\,368$
$2^6 = 64$	$2^{16} = 65\,536$	$2^{26} = 67\,108\,864$	$2^{36} = 68\,719\,476\,736$
$2^7 = 128$	$2^{17} = 131\,072$	$2^{27} = 134\,217\,728$	$2^{37} = 137\,438\,953\,472$
$2^8 = 256$	$2^{18} = 262\,144$	$2^{28} = 268\,435\,456$	$2^{38} = 274\,877\,906\,944$
$2^9 = 512$	$2^{19} = 524\,288$	$2^{29} = 536\,870\,912$	$2^{39} = 549\,755\,813\,888$

1.2) Integer types

Try out the following types for encoding integer numbers:
`char`, `short int`, `int`, `long int`, `long long int`, `__int128`

- Print their size in bytes using `sizeof([type])`. How does this size correlate to the representable solution of the factorials?
- Investigate how the range changes if you use `unsigned` variants of these types. What do you observe?

Hints:

- Regarding the 128 bit integers, `__int128` are is not part of C++ but supported by certain compilers such as g++. They are (typically) not supported by `std::cout`. Therefore, please write your own printing routine for them.
- You can use the precompiler directive of the form `#define T [type of int]` to change the type of `T` throughout your entire program.

2) Trip planning continued

We get back to our previous exercises (which you should have adequately finished so far!) on the trip planning program we developed before.

2.1) Overloading

Modify your class `Date` so that it overloads the following operators:

- the operator < should be used to call and return the value of the function `before(...)`.

- the operator - should be used to call and return the value of the function `difference(...)`.

2.2) Check implementation

Use the class `Trip` and the `main` function of exercise 2 of the lab worksheet #1 (they should be unchanged) to check this new implementation.

2.3) Independent functions

Implement the overloaded methods as independent functions (outside the class).

3) Generics

3.1) Skeleton

Define a generic class `MyCollection` that represents a set of elements of type `T`. These elements are stored in a C-style array, which is dynamically allocated by the constructor and deallocated by the destructor.

Hints:

- Use `template` to realize generic types.
- Note that all implementations of the templated classes must be made available in the header file and not in the `.cpp` file. This contrasts with the standard C++ programming style to split the declaration and the implementation/definition into `.h` header and `.cpp` files.

3.2) Attributes/member variables

The attributes of this class are

- the address of the array,
- the maximum number of elements that can be stored in the array (constant, initialized by the constructor), and
- the current number of elements stored in the array.

3.3) Insert & get methods

It has at least a method `insert_elem(...)` to insert a new element at the end of the collection and a way `get_elem(...)` that gets the element stored at a specific index `i`.

3.4) std::cout method

Also, define an overloading of the << operator for this class to print out some information. Have a look at the lecture notes for more information and the following program snippet.

```
1000 // We first need to tell the compiler that there is a templated
        class MC
     template<class T> class MC;
1002
```

Université Grenoble Alpes - 2025

```
// Then, we provide the implementation of the << operator for this
    class
template <class T>
std::ostream &operator<<(std::ostream &o, const MC<T> &p)
{
  [...]
  return o;
}

// Finally, we can make the << operator implementation to be a
template<class T>
class MC {
  public:
  friend std::ostream &operator<< <>(std::ostream &o, const MC<T> &
   arr);
};
```

3.5) Factorial program

Develop the following two independent functions:

```
void init(MyCollection<int> &c, int k);
void apply_fact(const MyCollection<int> &c, MyCollection<int> &res);
```

The function `init(...)` initializes the collection `c` with `k` integers (chosen pseudo-randomly), and the function `apply_fact(...)` fills the collection `res` with the factorials of the elements of collection `c`. Write a `main(...)` function to check these functions.

3.6) Higher precision runs

Test variants of these programs with higher precision support than before.

3.7) Using std::vector

Adapt the functions of the previous assignment to use the generic class vector of the STL (`http://www.cplusplus.com/reference/vector/vector/`) instead of `MyCollection`.

In the end you should have both versions implemented and usable (potentially based on a precompiler directive or whatever you prefer).

4) Template class (optional, exercise of lecture 3)

4.1) Warmup

Define classes `Dollar`, `Euro`, and `Pound` to represent respectively an amount of money in dollars, euros, and pounds.

Each class should have a constructor that takes as a parameter a float to initialize the amount, a method `get_value()` that returns the amount, and an overloading of the << operator.

4.2) Valid currency

Write a templated function `bool check_type(Type_currency object)` which returns a boolean depending on a valid currency (Dollar, Euro, or Pound) being used.

4.3) Bank account

Define a templated class `Bank_account` with generic type `Type_currency`. This class has two attributes:

- a character string `owner_name`, and
- a `Type_currency` balance.

This class has a constructor, a copy constructor, a destructor, and a method `credit_balance()` that returns `true` if the amount of balance is positive.

`class Bank_account` should be usable with the example main given in the lecture.

4.4) std::cout

Define an overloading of the << operator for this class.

Université Grenoble Alpes - 2025