Object-oriented and softw. design - C++

Fall 2025

Lab worksheet #1

17.09.2025

*** In these exercises, you will only use the notions of C++ presented up to lecture 1 ***

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

Hint: You need **only pointers to char variables** in this worksheet, but **no pointers or references** in general unless the code snippet is provided!

Development environment:
You will clearly use a more advanced development environment at work later on.
However, to learn programming properly, you **are only allowed to use any AI assistance** to edit your files within this lab. This will also help you to write less error-prone code within the tests.

Attendance of labs:
Lab attendance is mandatory according to the study rules (available in the UGA cloud).

1) Extension of exercise 1 of lecture 1

1.1) Warmup

- Recall the role of a constructor in a class.
- Recall how class instances communicate (how their member functions are executed/called) and give an example.

1.2) Date

Define a class `Date`, with 3 integer attributes: day, month, year.

- Create a constructor with the same 3 attributes (day, month, year) as parameters. Check that the parameters represent the correct date. Otherwise, use the Unix Epoch (1 January 1970).

- Create a second constructor with a parameter given by a variable of type `time_t` (as it can be returned by the function `time`, see man page printouts on the last page). The constructor initializes the attributes using the fields `tm_mday`, `tm_mon` and `tm_year` of the struct `tm` returned by `localtime` (see man page below). We will get back to pointers later. Feel free to use the following snippet in your code.

```
// Get the current time in seconds
time_t t = time(NULL);

...

// Created a copy of the detailed time information to the
   structure t2 using
// the reference & and pointer * to which we will familiarize
   ourselves later.
struct tm t2 = *localtime(&t);

// Next, we can access the variables in the structure by using
   ".",
// e.g., by printing out the month:
std::cout << t2.tm_mday << std::endl;
```

- Write a method `print_date` to print the date. In this method, use a `switch` statement to print out the corresponding month, e.g., 1 → "Jan", 2 → "Feb", 3 → "Mar", ...

1.3) "main" function

Define a function `main` that creates an instance of `Date` using the current time (second constructor) and prints it. Check that your methods work correctly.

1.4) "happy birthday"

Write a method `happy_birthday` that receives as parameters a name `(char *) name` and a `Date b` (the day of birth of `name`) and that wishes a happy birthday by also naming the person if the date equals his/her birthday, and prints his/her age.

1.5) "main"

Update the `main` function that enables the corresponding command to receive 4 parameters: the name, day, month, and year of his/her birth. It creates a `Date` that corresponds to today and wishes a happy birthday to the specified person if this date is his/her birthday. Remember to check that the command receives the expected number of parameters!

Here are example executions:

```
$ ../bin/ex1_date Nestor
Wrong number of arguments: ../bin/ex1_date name day month year
$ ../bin/ex1_date Nestor 27 9 1995
Happy birthday Nestor! You are 24 years old
```

Hints:

- To convert char arrays to integers, you can use the `atoi()` function. Check out its manpage!
- C++ files are compiled with `g++` and not with `gcc` which is for C files.

1.6) Makefile (optional in this worksheet)

This assignment is optional, but you should give it a try. We will use Makefiles repeatedly in the following worksheets.

Organize your files as in the structure given below and write the associated `Makefile`. Ensure the executable will be generated in the directory `bin`.

Introducing such a directory structure is a little bit too much for this small example, but the idea is to get used to such a structure, which you'll see in larger projects.

```
example_1/
├── bin
├── include
│   └── date.h
├── Makefile
└── src
    ├── date.cpp
    └── datemain.cpp
```

Hints:

- Make sure that the header files are available in the "include path" of the compiler. See `g++` manpage and its option `-I`.
- You might have seen some examples of Makefiles, which are for C files. However, we have C++ files here. Make sure that you're setting up the correct variables. E.g., it's not `CFLAGS`, but it's `CXXFLAGS`.

2) Extension of exercise 2 of lecture 1

2.1) Trip

Define a class `Trip`, with 3 attributes: beginning and end dates of the trip, and price (float).

- Create a **constructor with 7 parameters**: day, month, and year of the beginning date, day, month, and year of the end date, and price.
- Create a second **constructor with 3 parameters**: beginning and end dates, and price.
- Write a method `print_trip` to print out some information of the trip.

2.2) "main"

Define a function `main` that makes use of the methods above to see whether they are working.

2.3) `price_per_day`

Now, we want to write a method `price_per_day` that computes the price per day of the trip. For this, it is necessary to be able to compute the duration of the trip. Before defining `price_per_day`, write the following independent functions (put them in a file `utils.cpp`):

- `bool before(Date d1, Date d2);`
  Return `true` if `d1` is before `d2`, otherwise `false`.
- `int difference(Date d1, Date d2);`
  Return the number of days between `d1` and `d2` by assuming that `d2` is before `d1`, hence computing "`d1-d2`". To simplify things, feel free to assume that **every month is 30 days long**.
- `int duration(Date d1, Date d2);`
  Return the number of days between `d1` and `d2` (it calls `difference(...)` depending on the fact that `d1` precedes `d2` or vice versa).

Add getters in class `Date`, if necessary.

2.4) "main"

Define a function main that enables the corresponding command to receive the parameters day, month, and year of the beginning of a trip and the day, month, and year of its end. The program interactively asks the user to input the trip's price if the command receives the expected number of parameters. Then, it prints the characteristics of the trip and the price per day.

2.5) Makefile (optional, but recommended)

Organize your files as described below and write the associated Makefile.

Again, writing a Makefile is optional in this assignment, but try it! We'll get used to this more over the following labs. The directory `bin` will receive the generated executable.

```
example_3/
├── include
│   ├── classes
│   │   ├── date.h
│   │   └── trip.h
│   └── others
│       └── utils.h
└── src
    ├── date.cpp
    ├── Makefile
    ├── trip.cpp
    ├── tripmain.cpp
    └── utils.cpp
```


3) Alternative version of trip

3.1) Alternative version

Create an alternative version of the application of exercise 2, in which the functions of task 2.3 are now methods of class `Date`:

- `bool Date::before(Date d);`
- `int Date::difference(Date d);`
- `int Date::duration(Date d);`

Recall that the keyword `this` is used in C++ to represent the pointer to the object on which the member function is being called.

3.2) Getters?

Do you still need getters with this version? Why?

3.3) Cleanup

Adapt all the files accordingly, as well as the organization and the Makefile.

Manpage of **time**:

```
TIME(2)                    Linux Programmer's Manual                    TIME(2)

NAME
time - get time in seconds

SYNOPSIS
#include <time.h>

time_t time(time_t *t);

DESCRIPTION
time()  returns the time as the number of seconds since the Epoch,
1970-01-01 00:00:00 +0000 (UTC).
If t is non-NULL, the return value is also stored in the memory pointed to by t.

RETURN VALUE
On success, the value of time in seconds since the Epoch is returned.
On error, ((time_t) -1) is returned, and errno is set appropriately.
```

Manpage of **localtime**:

```
LOCALTIME(3)                Linux Programmer's Manual                LOCALTIME(3)
NAME
localtime - transform date and time to broken-down time

SYNOPSIS
#include <time.h>

struct tm *localtime(const time_t *timep);

DESCRIPTION
The localtime() function takes an argument of  data  type  time_t,
which represents calendar time.  When interpreted as an absolute
time value, it represents the number of seconds elapsed since the
Epoch, 1970-01-01 00:00:00 +0000 (UTC).

Broken-down time is stored in the structure tm, which is defined in
<time.h> as follows:
struct tm {
    int tm_sec;    /* Seconds (0-60) */
    int tm_min;    /* Minutes (0-59) */
    int tm_hour;   /* Hours (0-23) */
    int tm_mday;   /* Day of the month (1-31) */
    int tm_mon;    /* Month (0-11) */
    int tm_year;   /* Year - 1900 */
    int tm_wday;   /* Day of the week (0-6, Sunday = 0) */
    int tm_yday;   /* Day in the year (0-365, 1 Jan = 0) */
    int tm_isdst;  /* Daylight saving time */
};
The members of the tm structure are:
...
tm_mday   The day of the month, in the range 1 to 31.
tm_mon    The number of months since January, in the range 0 to 11.
tm_year   The number of years since 1900.

The localtime() function converts the calendar time timep to  broken-down  time
representation,  expressed  relative  to  the  user's  specified timezone.
```