Université Grenoble Alpes 2018-2019
M1 MSIAM

Algorithms and software tools - C++
Lab 1

Exercise 1. *Extension of the lecture exercise (Date)…*

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

- Create a constructor with 3 parameters. Check that the parameters represent a correct date, otherwise use the Unix Epoch (1 January 1970).

- Create a second constructor that takes as parameter a variable of type *time_t* (as it can be returned by *time*, see below). The constructor initializes the attributes using the fields tm_mday, tm_mon and tm_year of the *struct tm* returned by *localtime* (see below).

- Write a method *print_date* to print the date. In this method, use a switch statement to convert the month (int) into the corresponding char* (1 → "Jan", 2 → "Feb", 3 → "Mar",…).

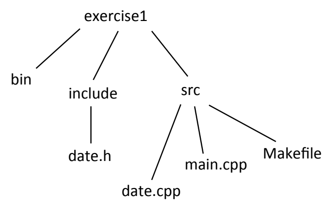
2. Define a function *main* that creates an instance of *Date* using the current time (second constructor), and prints it. Check that your methods work properly.

3. Write a method *happy_birthday* that receives as parameters a name n and a date b (the day of birth of n) and that wishes an happy birthday to n if the date equals his/her birthday, and also prints his/her age.

4. Define a function *main* that enables the generated command to receive 4 parameters which are a name and the day, month and year of his/her birth. It creates a Date that corresponds to today, and wishes an happy birthday to the specified person if this date is his/her birthday.
Do not forget 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 25 9 1995
Happy birthday Nestor! You are 23 years old
```

5. Organize your files as described to the right, and write the associated Makefile. Note that the executable will be generated in directory *bin*.



```
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.
```

```
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.
```

Exercise 2. *Extension of the lecture exercise (Trip)…*

1. 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 the characteristics of the trip.

2. Define a function *main* that checks the methods above.

3. Now we want to write a method *price_per_day* that computes the price per day of the trip. To that goal, 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);* that returns true if d1 precedes d2, otherwise false (to simplify, assume that d1 and d2 have the same value for the year attribute)

- *int difference(Date d1, Date d2);* that symbolizes d1 - d2 i.e., returns the number of days between d1 and d2 (to simplify, assume that every month is 30 days long)

- *int duration(Date d1, Date d2);* that returns the number of days between d1 and d2 (it calls *difference* appropriately, depending on the fact that d1 precedes d2 or vice versa).

Of course, add getters in class *Date* if necessary.

4. Define a function *main* that enables the generated command to receive as parameters the day, month and year of the beginning of a trip, and the day, month and year of its end. Provided that the command receives the expected number of parameters, the programs interactively asks the user to input the price of this trip, then it prints the characteristics of the trip and the price per day.

5. Organize your files as described below:
- a directory *include* contains 2 sub-directories: *classes* that contains *date.h* and *trip.h*, and *others* that contains *utils.h*
- a directory *src* contains files *utils.cpp* (this file contains the functions of question 3), *date.cpp*, *trip.cpp*, and *main.cpp*, as well as the *Makefile*
- a directory *bin* that will receive the generated executable,

and write the associated Makefile.

Exercise 3. *Alternative version of Trip*

Create an alternative version of the application of exercise 2, in which the functions of question 3 are now methods of class *Date*:

- *bool Date::before(Date d);*
- *int Date::difference(Date d);*
- *int Date::duration(Date d);* - to write this method, you need to know that the keyword ***this*** is used in C++ to represent the pointer to the object on which the member function is being called.

With this alternative version, do you still need getters?

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

Exercise 4. *Simple classes for polynomials*

1. Define a class *Poly0* that represents polynomials of degree 0. This class has only one (private) attribute of type float[1]. It has also a constructor to initialize the attribute, as well as a method *val* that computes the value of the polynomial at a point *x*.

2. Define a class *Poly1* that represents polynomials of degree 1. This class has two private attributes of type float. This class must define at least:
- a constructor to initialize the attributes
- a method *val* that computes the value of the polynomial at a point *x*
- a method *zero* that computes the root (if any) of the polynomial in an interval [a,b] specified by parameters
- a method *derivative* that returns the derivative polynomial (of type *Poly0*)

3. Define a function *main* that:
- declares and initializes an array of size N that contains instances of *Poly1*
- computes and displays the roots of these polynomials (given an interval[a,b])
- computes and displays the derivatives of the polynomials.

As before, organize your application with different directories for headers, .cpp files, and the executable.

[1] Remark: strictly speaking, the null polynomial is not of degree 0. You can either define an other class *PolyNull* to handle this case, or consider that *Poly0* includes the null polynomial.