PROGRAMMING EXERCISES - OBJECT ORIENTED PROGRAMMING USING C++

Introduction to classes

1.

Consider the following definition of class **Date** that will be used to represent dates:

```
class Date
{
public:
  Date(unsigned int year, unsigned int month,unsigned int day);
  Date(string yearMonthDay); // yearMonthDay must be in format "yyyy/mm/dd"
  void setYear(unsigned int year) ;
  void setMonth(unsigned int month) ;
  void setDay(unsigned int day) ;
  void setDate(unsigned int year, unsigned int month, unsigned int day) ;
  unsigned int getYear() ;
  unsigned int getMonth() ;
  unsigned int getDay() ;
  string getDate();  // returns the date in format "yyyy/mm/dd"
  void show();       // shows the date on the screen in format "yyyy/mm/dd"
private:
  unsigned int year;
  unsigned int month;
  unsigned int day;
};
```

a) Write the code of all the methods of the class. For now, consider that all the dates represented in objects of class `Date` are valid.
b) Write the `main()` function of a program that tests all the methods of the class.
c) Add the following methods to the class:
- `isValid()` - a method that returns a boolean value indicating whether the date is valid or not; suggestion: implement a private method that returns the number of days of a given month-year pair;
- `isEqualTo(const Date &date)` - a method that returns a boolean value indicating whether the date represented in an object is equal to the date received as parameter;
- `isNotEqualTo(const Date &date)` - a method that returns a boolean value indicating whether the date represented in an object is equal to the date received as parameter;
- `isAfter(const Date &date)` - a method that returns a boolean value indicating whether the date represented in an object is after date received as parameter;
- `isBefore(const Date &date)` - a method that returns a boolean value indicating whether the date represented in an object is before the date received as parameter.

Note: in the future you will learn how to overload the operators `==`, < and > so that they can be used to compare objects of type `Date`, as well as operator << to output the attributes.

d) Modify the `main()` function of the program in order to test the new methods.
e) Modify the implementation of the class so that the atributes (`year`, `month` and `day`) are represented in a single string, using the format "`yyyymmdd`". Verify that the `main()` function that you developped in **d)** does not have to be modified.
f) Explain why the following variable declaration is not possible: `Date d1;` .
g) Modify the definition of class `Date` so that the definition in **f)** is possible. Suggestion: the default constructor must construct an object representing the current date; you must search for a way of obtaining the current date from your computer.

2.

(*from 1st season exam - 2014/15*) The following class was defined to store the relevant information about a student's final grade in a specific course. The final grade depends on the grades obtained in the short exam, the project and the exam of the course.

```
class Student {
public:
  Student();
  Student(const string &code, const string &name);
  void setGrades(double shortExam, double project, double exam);
  string getCode() const;
```

```
    string getName() const;
    int getFinalGrade() const;
    // other get and set methods
    bool isApproved() const; // is the student approved or not ?
    static int weightShortExam, weightProject, weightExam;  // weights in percentage (ex:20,30,50)
  private:
    string code; // student code
    string name; // student complete name
    double shortExam, project, exam; // grades obtained by the student in the different components
    int finalGrade;
  };
```

a) Write the code of the `setGrades` method. This method should also set the value of the `finalGrade` attribute, taking into account the values of its parameters, and the weights of the `shortExam`, `project`, `exam` components in the `finalGrade` value, which are, respectively, `weightShortExam`, `weightProject`, `weightExam`. The calculated value must be rounded to the nearest integer; when the decimal part of the calculated value is 0.5 the `finalGrade` must be rounded up.
Justify the use of `static` qualifier in the attributes `weightShortExam`, `weightProject`, `weightExam` and define them with the values 20, 30 and 50, respectively.

b) Write a piece of code that reads, from the keyboard, the code, name and grades of a student (in the short exam, project and exam) and creates an object **s** of type `Student`, having the attributes read from the keyboard. On the right, you can see an example of a dialog with the user, during the execution of this piece of code.

Example of the execution of the piece of code:

```
Student code? up20141007
Student name? Ana Silva
Short exam grade? 13.5
Project grade? 17
Exam grade? 15.7
```

(*in the code, after this dialog, object s should be created*)

c) Considering that the information about the students that took a course is stored into a `vector<Student>`, write the code of a function that receives as parameters an ouput stream and a vector of that type and writes to the output stream, the name and final grade of the students that were approved. The names and grades must be vertically aligned; consider that the maximum length of a name is 50 chars.

d) Implement the remaining methods and develop a program to test them (*not included in the exam*).

3.

(*adapted from Big C++ book*) Develop a program to print out an invoice. An invoice describes the charges for a set of products, acquired in certain quantities. Complexities such as dates, taxes, and invoice and costumer numbers may be omitted. The program simply prints the billing address, all the items (description, unit price, quantity ordered and total price), one item per line, and the total amount due. For simplicity, no user interface is required; simply use a test harness that adds items to the invoice and then prints it. Your program must include the classes: `Client` (describes the client name and address), `Product` (describes a product with a description and price), `Item` (describes a line of the invoice; see example below) and `Invoice` (describes an invoice for a set of purchased products).

Example of output:

```
DEI- FEUP
Rua Dr. Roberto Frias, s/n
4200-465 Porto

Description           Price   Qty    Total
-------------------- ------ ----- --------
Computer             999.90    10  9999.00
Printer              149.90     1   149.90

Amount due: 1148.90 euro
```

4.

Develop a small program for the management of a library. The program must allow the management of the books and of the users, as well as the borrowed books. All the information must be permanently saved in files.
For simplicity, you may assume that books and users are never removed from the system, although books may be "lost" and users may be "not active" (use an attribute to store those properties).
Notes:
- use `const` qualifier, both in methods and in method parameters, whenever you find they should be used;
- you may improve your program by adding to each book the date when it was taken by the user, using objects of classe `Date`, from a previous exercise, for that purpose.

5.

Define a class **Person** for representing persons. Consider that a person has the following attributes: name, gender and birthdate. Use an object of class **Date** (see problem 1) to represent the birthdate. Develop your program using separate compilation.

6.

Write a template function that determines the minimum and maximum values of vector of numbers that it receives as parameter.

7.

a) Suppose you didn't have class **vector** from the STL. Define and implement a template class, **Vector**, that emulates the functioning of class **vector** from the STL. Your class **Vector**, defined below, must only implement a limited set of functionalities; the functionality of the methods is the same as in **vector** (from the STL). The memory for the elements of **Vector** must be allocated dinamically. **Suggestions**: use function **malloc()** for dynamic memory allocation; to implement method **push_back()** investigate the use of function **realloc()** to increase the space allocated for the buffer.

```
template <class T>
class Vector
{
public:
  Vector();
  Vector(unsigned int size);
  Vector(unsigned int size, const T & initial);
  Vector(const Vector<T> & v);
  ~Vector();
  size_t size() const;
  bool empty() const;
  T & front();
  T & back();
  T & at(size_t index);
  void push_back(const T & value);
  void pop_back();
  void clear();
private:
  T * buffer;
  size_t bufferSize;
};
```

b) In the **vector** class of the STL, the reallocation of memory is not done every time a new element is pushed back; instead, it is allocated in new chunks. The capacity of a **vector** tells us how many elements the vector could hold before it must allocate more space. Modify the definition and implementation of the class methods in order to include a new attribute **bufferCapacity** and two new methods: **capacity()** that tells the current capacity of the **Vector** object, and **reserve()** that requests that the vector capacity be at least enough to contain **n** elements, where **n** is a parameter of the method. Before doing your implementation, try to infer how the capacity of a vector is increased in the STL; for that, start with an empty **vector** object, make a lot of push backs, and determine its capacity after each one of the push backs.