TP1 This exercise aims to revise classes and code organisation (.cpp, .h and makefile).
Show each completed step of this exercise to the tutor.

1. In a *student.h* file, write the prototype of a **Student** class containing:

   - The name of the student (limited to 127 characters: we will not use a string).
   - The date of birth of the student, encoded in an integer YYYYMMDD format.

   The constructor of this class will have to take this data as an argument.

   Add to this class these functions (only declarations!):

   - Name (), which returns the name (a const char *).
   - Birthday (), which returns an integer representing the student's birthday in the MMDD format (so if it was born on the 23/07, we return 723)

2. In a *student.cc* file, write the implementation of this class.
   Indications:

   - Do not forget to include the .h at the beginning.
   - The syntax for defining a class function outside the class looks like:

     ```
     type Class::Method(type_arg arg1, type_arg arg2) {
           ...
     }
     ```

   Copy the string (the name), character by character. Do not keep only the pointer!
   You can for example use `strncpy` but be careful to add the character '\ 0' if necessary!

   Check by compiling your files: `g++ student.cc -c -o student.o`

3. In 2 files *class.h* and /class.cc], write the prototype and the implementation (i.e. declaration and definition) of a class **Class** containing:

   - The name of the course (unlimited size), for example "XML", and a textbfName() function returning it.
     This time, use string instead of `char *` or `char [ ]`! You will need $\#include\ <string>$ and using $std :: string$;
   - The maximum number of students that can enroll in this course (which will never exceed 1000), and a `MaxNumberOfStudents ()` function that returns it. The list of students enrolled in this course:
     - The number of students enrolled: a NumStudents () function.
     - The possibility to enroll a student in the course: a function `AddStudent (Student * student)`, which returns the index of the student (0 for the first student, 1 for the second, etc.), or -1 if the course is already filled to the maximum.
     - Consult a registered student: a function `GetStudent (int student_index)`. What will she send back?

   N.B.

   - Remember to include the student.h file.
   - To build a Class, you will take its name and maximum number of students, and you will have no student enrolled at the beginning.

Check your code compiling it!

4. In two files, namely *io.h* and *io.cc* files, write the declaration and then the definition of the functions:

   - `void PrintStudent (const Student &)`, which "displays" the description of a student (with cout $<<$ ...) on a line.
   - `void PrintClass (const Class &)` which displays the description of a class (on one line) and all the students who are registered (one per line).
   - `Student * EnterStudent ()` which creates and fills a Student (name, date of birth) with the `cin` interactive input, and using the `new` operator.
   - `Class * EnterClass ()` which does the same thing for a class, then asks for the number of students to enroll, types them one by one and finally add them.

   Include the *student.h* and *class.h* files in the *io.h* file.

   N.B. Remember to use a `#ifdef` guard in .h files. If not you will probably have a compilation error of "duplicate declaration".

5. In a `main.cc` file, write a *main ()* function that, using all the function created above, will request the interactive input of a Class, and then display it.

   Compile it in separate steps:

   - For each module done previously: g ++ -c module.cc -o module.o
   - For the main.cc, to choose:
     - g ++ module1.o module2.o ... main.cc [-o main]
     - Same, but with the -c option, and with -o main.o, and then g ++ module1.o module2.o .. main.o [-o main]

   Run the program.

   Alternatively, you can compile it using the IDE Eclipse installed on the computer.

6. Complete the `Makefile` you find in moodle that will script all that, with a binary `main` that will be the default target , and a `module.o` target per each module.

   Text it with the `make` command.