Lesson 4

If you find any open questions in some of the presented exercises then feel free to make your own decisions. However, your decisions should be reasonable and must not contradict any of the conditions explicitly written for the exercise. Please, write comments in the programs that clarify your assumptions/decisions, if any.

Exercise 1

Write a program that performs the following steps, by the indicated order.

1. Generate a sequence of 31 distinct random numbers in the interval $[1, 95]$ (i.e. the generated sequence should not have repeated values).
2. Display the first 20 generated random values, sorted increasingly.
3. Display the last 11 generated random values, sorted increasingly.

Use **`STL`** to solve this exercise. Use standard algorithms, instead of hand-written loops (such as **`for`**-loop, **`while`**-loop, or **`do`**-loop), whenever possible.

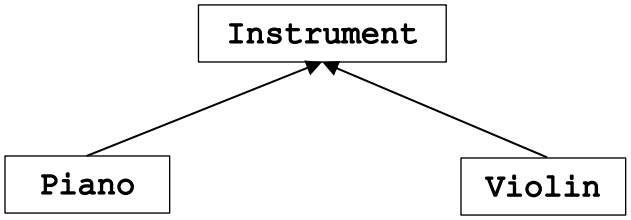
An example is shown below.

Example

```
First 20 values: 4 6 7 13 17 22 26 38 42 50 51 65 68 70 75 78 79 82 91 92
Last 11 values: 2 8 33 37 41 44 47 54 58 59 67
```

Exercise 2

Consider the musical instruments hierarchy given in the file **`exerc2_V0.cpp`**. The hierarchy is depicted in the figure below.



The program in the file **`exerc2_V0.cpp`** is ready to be compiled and executed.

a) Modify the hierarchy such that

   i. Each instrument should have a unique **`id`** that is automatically assigned when an instrument is created and, thereafter, it should not be possible to change the **`id`**. Note that an **`id`** is always a non-negative integer and it is not provided by the client code. For instance, to create an instance of **`Piano`** then the client code only needs to declare

```
Piano P1("Yamaha");
```

   ii. Add a function **`get_identification`** that returns a string with the format "*`inst_brand`*`<`*`inst_id`*`>`", where "*`inst_brand`*" and "*`inst_id`*" are respectively the instrument's brand and the instrument's id.

   iii. Add a function **`set_brand`** that allows to modify the brand of an instrument.

   iv. Each violin should store information about whether it is made in wood. When a violin is created, if no information about whether the violin is made in wood is given then by default it is assumed to be a wooden violin. For instance,

```
Violin V1("Stradivarius");
```

   should create a wooden violin.

   v. Add a function **`is_wooden`** to test whether a violin is made in wood.

b) Confirm that it is possible to use the copy constructor to create an instrument that is a copy of another existing instrument (e.g. **`Piano P3(P1);`**), while it is not possible to assign one instrument to another (e.g. **`P1 = P2;`** does not compile). Motivate why it is not possible to use the assignment operator.

c) Since each instrument is unique, we do not wish that it is possible to create copies of an instrument by using the copy constructor. To this end define first a new class, named **`Uncopyable`**, to represent objects that cannot be copied, either by using the copy

constructor or the assignment operator. Then, make sure that all instruments are **`Uncopyable`**. Note that it should not be possible to create **`Uncopyable`** objects that are not instances of some other class, like **`Instrument`**.

d) Investigate what possible use can client code have of pointers of the type **`Uncopyable*`**. For instance, consider the pointer below

```
Uncopyable *ptr = new Violin("XXX");
```

Are there any member functions of the **`Instrument`**s hierarchy that can be called out of the pointer **`ptr`** (e.g. **`ptr->is_wooden()`**)?

Lycka till!!