1. Consider the following class hierarchy:

```
struct Vehicle {
virtual ~Vehicle() = default;
virtual void Move() {}
};
struct WheelVehicle: Vehicle {
virtual void TurnRight() {}
};
struct WingVehicle: Vehicle {
virtual void TakeOff() {}
};
// A plane is a vehicle with wheels and wings
struct Plane: WheelVehicle, WingVehicle {};
Plane plane;
```

What does it happen if your try to do:

```
Plane p;
Vehicle& v = p;
```

Compilation will fail.

Why?

```
Vehicle                     Vehicle
   |                           |
WheelVehicle            WingVehicle
             \       /
              Plane
```

The problem is the compiler finds out there are two different Vehicles in its inheritance path and when calling Move, which of both methods will be called?

How would you call the Move() method from a plane?

Now is not possible unless minor changes done:

For example declare the Move method inside plane (since its virtual) and inside Plane select which Move method we want to call:

```
struct Plane: WheelVehicle, WingVehicle {
        void Move() {
        WheelVehicle::Move();
        };
};
```

How would you solve this problem?
A better solution is:
One way could be defining inheritance as virtual, this way the compiler knows only one Vehicle will be available in the inheritance path.

```
                    Vehicle
               /               \
WheelVehicle              WingVehicle
                   \        /
                    Plane
```

```
struct WheelVehicle: virtual Vehicle {
virtual void TurnRight() {}
};
struct WingVehicle: virtual Vehicle {
virtual void TakeOff() {}
};
```

2. how would you correct this code? What's wrong with it?

```
ev_t *car;
/* … work with car, car points to a defined ev_t object ... */
memset(car, 0, sizeof(car));
```

Depending on what you want to do:
Initialize the pointer to NULL:

```
memset(&car, 0, sizeof(car));
```

Initialize the instance pointing by car to all zeros:

```
memset(car, 0, sizeof(ev_t));
```

(Most probably 2nd option)

3. Find ordered pairs of positive numbers (x,y) given the variables O, P, A and B, such that their sum is divisible by both A and B. And 1<=x<=O and 1<=y<=P

Please Check ex_3 folder

4. Implement a Int function F that calculates the sum of positive values less than 1000 multiples of 3 and positive values less than 1000 multiples of 5.

Please Check ex_4 folder

5. Given a file list of files, implement a function that returns the name
of the earliest modified file, owned by the administrator, executable
and with a size smaller than 14*220.

If no file matches the requirement, it shall return “NO FILES”
Example list of files:
admin -wx 29 Sep 1983 833 source.h
admin r-x 23 Jun 2003 854016 blockbuster.mpeg
admin --x 02 Jul 1997 821 delete-this.py
admin -w- 15 Feb 1971 23552 library.dll
admin --x 15 May 1979 645922816 logs.zip
jane --x 04 Dec 2010 93184 old-photos.rar
jane -w- 08 Feb 1982 681574400 important.java
admin rwx 26 Dec 1952 14680064 to-do-list.txt

Please Check ex_5 folder

6. Given a set of coin throws (1=heads, 0=tails), implement a function that that determine the minimum amount of coins that have to be reversed to make the throw sequence alternating (example: 0,1,0,1)
Example coin throw:
(1, 0, 1, 0, 1, 0, 0, 1, 1, 0)

Please Check ex_6 folder

7. Data packets are received from 3 different channels. The packets must reach two objects that handle several aspects of a process. How would you design the class hierarchy

At first sight I would wrap the access of the data offering the same interface for each one of the channels. I would use the Observer pattern to notify each one of the two objects when a new data has come. If processes handled by objects consume too much time I could consider saving incoming data in a Priority Queue to be handled when idle time is available.

8. What does the Visitor pattern do? What is it useful for? What principle of software design does it comply with? What are the advantages of using design patterns?

Visitor pattern is used to operate over a set of objects that share some functionality and let the operation algorithm to future uses, separating the algorithm independent from the objects and following the open-close principle (one of the SOLID principles).

Design patterns are a set of designs that can be used to solve well known problems when designing a system. Using not only will help following the SOLID principles but also following these patterns will help programmers and designers to exchange ideas and standardize code which will be more easily understood.

9. What are the major categories of the standard library? What are the advantages of using it?

Main Categories of the STL are:

- Containers
- Iterators
- Algorithms
- Funcions

Inside the STL there are tons of good quality code ready to be used which has been checked, maintained and reviewed for lots of years (Depending on the implementation, of course) so I cannot imagine a bad reason not to be used (except for compatibility issues and even in that case normally you can find a reduced version of the library for those devices.

10. Given this class

```
class ElectricVehicle {
    public:
    ElectricVehicle(std::string p_brand, std::string p_model, int p_energy) :
    brand(p_brand), model(p_model), energy_capacity(p_energy) {}
    int getMaxEnergy() { return energy_capacity; }
    Private:
    std::string brand;
    std::string model;
    int energy_capacity;
    };
```

define a vector of cars with this data:
Brand: Tesla, Model: Model 3, Battery Capacity: 75
Brand: Kia, Model: Soul, Battery Capacity: 64
Brand: BMW, Model: i3, Battery Capacity: 42.2
Brand: Renault, Model: Zoe, Battery Capacity: 52
Brand: Peougeot, Model: e-208, Battery Capacity: 50
Order the vector by battery capacity in one single line.

Please Check ex_10 folder

11. Given this vector of integers, return a vector of the same size where the values are doubled if the value is less than 100 and divided by 4 if it is greater than 600.

```
vector<int> v1 {50,200,800,35,75,150,190,750,900}
```

Please Check ex_11 folder

12. Imagine you have a class that represents the configuration of a device. You have to read and save this configuration to disk. You also have to send it through a REST API to the server and show it on a screen. What would be your approach to this design? Can you implement a skeleton version of it? Have you applied any design principle?

Please Check ex_12 folder

I simply applied the strategy pattern, but it can actually be easily changed, the important thinking for me is every class has a concrete use and software can easily be escalated to other needs (In future I may need a YAML conversion class and it may be done and used without modifying any of the code developed)

13. Imagine you have a class that controls the state machine of a device. You need to connect an RFID reader to this device and integrate its driver. You've been told that there's a chance that in 6 months from now the model of the RFID reader will change. How would you design the class hierarchy so the future change is smooth and painless? What can you say about the unit testability of your design?

The main idea would be you use abstraction to have access to the RFID, this means that in the first phase of the design a good abstraction of the access to the RFID must be done. After this abstraction is designed, the interface must be written following it and this interface is what must be stable over the years. With the interface written now is time to write the concrete implementations of the driver, first one for these days, and in 6 months change the implementation for the new RFID.
Unit test should not change from one version to another only if interface changes should be added.