We would like to implement the following three modules:

- **The first one** will generate byte arrays of random length [1...100] containing random bytes [0...255]. The byte arrays are generated continuously and delivered to the **second module**.

- **The second module** will receive the generated byte arrays from the first one and will look for a match of a specific constant byte sequence (for example 0x00, 0x01, 0x02) into the byte array. If a match is found in the array, it will deliver the array to the **third module**, otherwise it will just discard the array. The module will be able to receive up to 100 byte arrays from the first module using an input buffer.

- **The third module** will receive the byte arrays from the second one and record them. Each time a byte array is received, the module will print to the standard output ALL the records received to that moment ordered by UTC time. The module will be able to receive up to 100 byte arrays from the second module using an input buffer.

NOTES:

- The code must be implemented in C++ for linux.
- Any C++ version is accepted (C++98/11/14…)
- Only STL is accepted (no BOOST).
- All the modules implement the same interface (**IModule**). The interface is defined by the developer.
- The main application **cannot** access to the modules definition. It can only access to **IModule** interface.
- Each module will have an internal thread to make the module work.
- This is a pseudo-code of the main application:

```
int main(void) {

    //Create modules
    IModule *m1 = (get object of type module 1).
    IModule *m2 = (get object of type module 2).
    IModule *m3 = (get object of type module 3).

    //Set m1 to deliver data to m2

    //Set m2 to deliver data to m3

    //Start modules

    //Wait 100 seconds

    //Stop modules

    //Release modules
    delete m1;
    delete m2;
    delete m3;
    return 0;
}
```

- We will consider:
  - Clean code.
  - Correct architecture.
  - Correct memory management.
  - Correct use of threads and synchronization.
  - Correct use of containers (maps, vectors, strings, etc…)