iFollow Test

Q1. Alarm System

The purpose of this exercise is to propose an API and an implementation (in C++ or Python) to play different alarm "audio" sequences. Usually, in material devices, there are 3 different level of alarms : LOW, MEDIUM and HIGH priority.

You can consider the audio sequences as follows:

- LOW: one beep every 30s, with a beep duration of 1s. Repeat continuously
- MEDIUM: one beep every second, with a beep duration of 250ms. Repeat continuously
- HIGH: five beeps every 500ms with duration of 250ms each, then wait for 2s with no beep. Repeat continuously

There are different rules regarding the alarms management:

- There can be only one alarm sequence at a time being played
- Only the highest priority active alarm is played at a time
- They can be started and stopped

If a higher priority alarm is stopped, the lower one might be played if active.

What you need to do:

a. You need to provide an API (objects, functions, methods) for the client application to start and stop alarms. The simpler, the better.

b. Implement unit tests to show that how to use the API and to show that your code works as expected.

c. From within your `main` fuction, use your API to start the alarm system.

d. You need to provide an appropriate architecture (classes, threads if needed, . . .). It has to be robust (we're talking about alarms here), as simple as possible and readable.

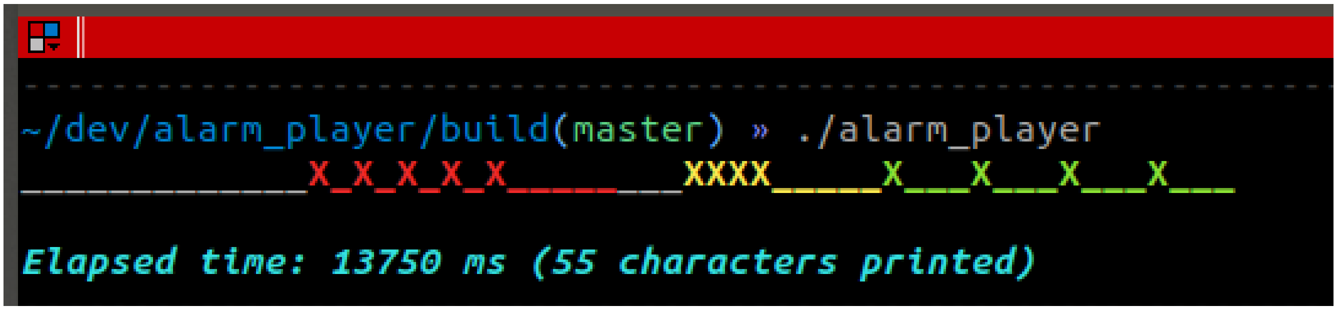
e. Use `git` as a versioning tool (hosting your project on gitlab/github). Create clear concise commits and provide a `README.md` file with the documentation explaning how to build, test and use your program. Give as well any extra information to help understand how the code works.

f. If coding in C++, the project shall be "cmake based" (it has to include a `CMakeLists.txt`) and it should allow us to run the main application as well as run the unit tests using `cmake` and `make` commands. You can ignore this if using Python.

The application input/output:

- When the main application is executed, the output should be continuously displaying characters simulating the buzzer state (you can add a real buzzer too if you like). One character represents 250ms, '`_`' is no beep and '`X`' is beep.
- Pressing on `h`: activates/deactivates (toggles) high priority alarm
- Pressing on `m`: activates/deactivates (toggles) medium priority alarm
- Pressing on `l`: activates/deactivates (toggles) low priority alarm
- Pressing on `ctrl-c`: politely kills the application showing some type of summary

Bellow we show an output example as some combination of the keys `h`, `m` and `l` are pressed ending with the `ctrl-c` key stroke:



```
-----------------------------------------------------------
~/dev/alarm_player/build(master) » ./alarm_player
_____________X_X_X_X_X________XXXX_____X___X___X___X___

Elapsed time: 13750 ms (55 characters printed)
```

Figure 1: Alarm System output example