[Practical Task] FreeRTOS: Getting Started

In this part, you are required to create simple task instances in a simulated real time environment with the help of FreeRTOS. You are required to use C programming for your implementation.

Before getting started, you should install your toolchain.

- On linux systems
  - Install CMake (3.24 or newer)
  - Install GCC
- On Windows
  - Install minGW (`https://sourceforge.net/projects/mingw/`)
    - mark 'mingw32-base' and 'mingw32-gcc-g++' for installation
    - in the 'Installation' menu, press 'Apply Changes'
    - Add minGW to PATH
  - install CMake (`https://cmake.org/download/`)
  - make sure CMake is added to the PATH
  - make sure the installation worked by running `'gcc --version'` and `'cmake --version'`

When you are using a Windows OS you can use the Linux toolchain via WSL (`https://learn.microsoft.com/en-us/windows/wsl/install`). Note, that you can use a different toolchain if you like. However, we do not provide support for any other toolchain than the ones mentioned above.

As an IDE you can use whatever you like. We suggest using either VS Code (`https://visualstudio.microsoft.com/vs/community/`, free) or CLion (`https://www.jetbrains.com/clion/`, educational license available).

Having checked that your toolchain is up and running, open the project from the provided .zip-file (chatterbox.zip). The provided .zip-file contains the folder structure given in Figure 6.

```
chatterbox
├── chatterbox_app
│   ├── main_exercise.c
│   ├── main.c
│   └──
├── example
├── src
├── CMakeLists.txt
└── README.md
```

Figure 6: Folder structure for Chatterbox-Application.

The `main()`-function in the file `main.c` is the entry-point to the application. In this exercise, this `main()`-function initializes the heap and calls the `main_exercise()`-function from the file `main_exercise.c`. Your task is to implement the missing parts of

the "chatterbox"-application in the provided `main_exercise.c`-file. The "chatterbox"-application basically consists of 3 instances of the "chatterbox"-task. Every time the chatterbox-task is executed, it outputs a string (in this exercise the string is simply printed to the console output). The behavior of a task instance is defined at its creation by two values: a) the output string and an integer flag which can have either value 0 (infinite task instance executions) or 1 (task instance shall only be executed 5 times).

To implement the chatterbox-app, extend the `main_exercise.c`-file as follows:

a) Think of a suitable data structure which can be used to pass the output string and the integer flag to the C function which implements the task instances (see part b.). Initialize the necessary data structures for three task instances in the `main_exercise()`-function as follows: The output string of each task instance is `TaskX` where `X` is the number of the task instance (ranging from 1 to 3). Task instance number 1 and 2 shall be repeated indefinitely, whereas task instance with number 3 shall only be executed 5 times.

b) Extend the `main_exercise.c`-file with the C function which implements the behavior of the chatterbox task instances. Chatterbox task instances shall print their output string every `mainTASK_CHATTERBOX_OUTPUT_FREQUENCY_MS` ticks. Use the `vTaskDelayUntil()`-function, to block a task instance until the time it has to generate the next output. After *Task instance number 3* has printed its output string for the final time it shall additionally print a string indicating its termination and delete its task instance.

c) In the `main_exercise()`-function create the three task instances using the code from the previous sub-questions. Give task instance number 3 higher priority than task instance number 1 and 2. Start the scheduler, so that the three task instance are executed concurrently.