Thank you for showing interest in our C++ Plugin Developer opening!

The task for this exercise is to implement a simple command interpreter library in C++, in a CMake project. The exercise involves both defining an interface to the library according to some requirements and implementing the behaviour of the library.

Requirements

The following functions are already provided as part of the library code project. Their signatures may not be changed.

- Initialise(): Is called to initialise any resources in the library.
- ShutDown(): Is called to release any resources in the library. Any long-running commands should be terminated cleanly.
- Poll(): Is called frequently to perform processing within the library. You may assume that this function is called at least once every 100ms, but the actual interval may not be regular.

You should implement the functions above to initialise and clean up any resources your library uses, and to perform any regular processing when Poll() is called.

In addition to the three functions above, you should also define and implement an interface for the command interpreter's functionality. The exact design of this interface is up to you, but must fulfil the following requirements:

- The user must be able to call a function to provide a command to the interpreter as a string. You may assume that only one command is provided per call to this function.
    - Commands are provided in the format "command arg1 arg2 ...", where the command name is mandatory and any arguments are dependent on the context of the command.
    - The command name and arguments are delimited by whitespace (which may not include newlines), and quoted arguments do not need to be supported.
- The user must be able to provide a single, static callback function to the interpreter, which serves to pass back output from executed commands. The output should be provided as a string.
- The user must be able to provide some arbitrary context data to the input function when they enter their command. When the command is processed and output is emitted, this context should be passed back through the output callback in addition to the output string.
- Commands should only be processed in response to a call to Poll(). The user may provide multiple commands to the interpreter before calling Poll(). In this case, these arguments should be processed in the order in which they were received, but do not have to all be processed in a single call to Poll().
- The interface functions will always be called from a single thread. The user should be able to call the command input function safely from code within their output callback.

FACEIT LIMITED – Millbank Tower, 25th Floor, 21-24 Millbank, London, SW1P4QP, United Kingdom Tel: +44 77 8994 3776 / Fax: +44 (0) 20 7100 8144 Reg
VAT Number: GB131331074, the Registrar of Companies for England and Wales

The following arguments must be supported by the interpreter:

- "echo [args] ...": Echoes all command arguments verbatim to the output callback, delimited by single spaces. If no arguments are specified, echoes an empty line.
- "countdown <secs>": Counts down from the given number of seconds, logging a message each second. The number of seconds is specified by exactly one argument.
    - When the command is first processed, "<n> seconds remaining" should be output from the interpreter, where <n> is the number of seconds remaining in the countdown. The message should then be repeatedly output once a second (with a precision of 100ms or better), while the number of seconds remaining is greater than zero.
    - Once <n> reaches zero, "Countdown complete" should be output. If the number of seconds originally provided to the command is zero, this message should be output immediately, and no messages regarding the number of remaining seconds should be output.
    - The valid range for the number of seconds to count down is between 0 and 10 inclusive. If any value outside of this range is specified, or a non-numerical value is specified, "Invalid countdown period of <x> seconds" should be output, where <x> is the argument that was provided.
    - Any arguments to the command after the number of seconds to count down should be ignored.

Any unrecognised command should output "Unrecognised command: <x>" to the output callback, where <x> is the name of the submitted command.

Other Considerations

The library must compile on both Windows and Linux, so should use portable and standards-compliant code. Which C++ standard you target is up to you, but be prepared to give reasons for your choice.

/W4 will be used to compile the library on Windows, and -Wall -Wextra -pedantic on Linux.

The library will be built in shared configuration (ie. as a .so on Linux and as a .dll on Windows), and should be resilient to differences in ABI between itself and the user's application. Keep this in mind when deciding how to define your interface. Only 64-bit support is required.

The structure of the internal code files for the library is up to you. Any extra files should be added to the project's CMakeLists.txt. Any source files not added will not be evaluated.

External libraries may not be used. Other code snippets found on the internet to solve specific problems may be used, if they are short and appropriately credited.

FACEIT LIMITED – Millbank Tower, 25th Floor, 21-24 Millbank, London, SW1P4QP, United Kingdom Tel: +44 77 8994 3776 / Fax: +44 (0) 20 7100 8144 Reg
VAT Number: GB131331074, the Registrar of Companies for England and Wales

Testing

Your solution will be tested internally by a test suite, to check the level of compliance with the specification.

Additionally, although we want to avoid bringing the complexity of writing exhaustive tests into the requirements of your solution, there is space in the code project to add “sanity” tests to verify your implementation. If you do sanity test your code as you write it, please include this code in your final solution and it will be included in the evaluation.

Submission

Please provide a zip file containing the entire code project, along with your modifications, to your FACEIT contact.

Evaluation Criteria

The exercise is not designed to be a pass/fail task, but rather an opportunity for us to learn more about your development style and experience. Although conformance to the specification is important, your implementation will be evaluated as a whole, based on a number of different aspects. These include, but are not limited to:

- **Correctness:** Does the library output what is expected? Does it appropriately cater for edge cases? Does it compile without warnings?
- **Design:** How is the external interface designed? Is it clear and straightforward to use? Is it portable? Are names and types chosen appropriately?
- **Testing:** Have you verified that your code works as expected?
- **Structure and readability:** Is the library implementation easy for others to understand? Is it structured in a logical way? Is it appropriately commented, and are the comments valuable?
- **Flexibility and maintainability:** How easy is it to add extra features, such as new commands? Is the code robust to human error if modified by other developers?
- **Knowledge:** How comfortable are you with the technologies and requirements presented? How is this reflected in your implementation?

Do not be afraid to submit a solution which is not fully finished, but do be prepared to explain the reasons why, and what improvements you would like to make.

If you do not have access to a Windows or a Linux platform to develop or test your solution in a cross-platform manner, let us know and this will be taken into account when evaluating it.