Recruitment: Qt Tasks

Task descriptions

General instructions

1. In Milo Solutions we pay attention to code quality - make sure your code looks nice and clean and is easy to read by others.
2. Good comments in source code are very important in our company. Don’t comment obvious things but make sure that everything is well commented. Use doxygen/ Qt comment style.
3. If you received more than two tasks, choose which two you will do. Think which of them will better represent your skills. If you solve harder task this may give you advantage over other candidates. We expect you to do at least 2 of the tasks, but in some cases doing only one can be enough.
4. We expect your final solution in a form of source code that can be easily built using Qt tools and fully working according to requirements. We assume that Qt Framework will be used.
5. Solution should be cross-platform. Exception is possible for platforms not fully supported by Qt framework. We most often test on Linux.
6. Using modern C++ is a plus.
7. Nice-looking UI is not required, but counts as a plus, too.

Task 1. Morse Code translator (QtWidgets or QtQuick)

Implementation of Morse Code translator using Qt Widgets or QML.

Requirements:

- it should be possible to both, load the text (either normal text or already coded one) from selected file and add it manually
- the program should also allow to save the coded / decoded text into a new file
- input and output texts should be visible in application window
- Morse-coded text should consist only of '.', '-' and ' ' (space) characters
- single space character is used to separate characters in a word, two space characters are used to separate words
- there should be no difference between upper- and lower-case letters
- characters that don't exist in Morse alphabet should be skipped

Task 2. Noughts and Crosses (QtQuick)

Implementation of "Noughts and Crosses" in QML without AI (only two players mode). User interface in QML must be intuitive and friendly. Use of graphical effects is a plus.

The game should indicate whose turn it is, what is the current player score, and it should show who has won the current round (or whether it is a draw).

Requirements:

- Game engine should be written in C++
- When a player wins, his winning line should be crossed (~~xxx~~ or ~~ooo~~)

Nice to have:

- If you can add an AI (computer) player, it will be a big plus

Task 3. Anagram derivation (C++)

An anagram derivation is a N-letter word derived from a N-1 letter word by adding a letter and rearranging. For example, here is a derivation of "*aliens*":

ail + s = sail + n = nails + e = aliens

Write a program that will find the longest such derivation from a specific 3-letter word (one word provided by user) in a list of words where every derived word also exists in the list of words.
words = {ail, tennis, nails, desk, aliens, table, engine, sail, etc....}

Requirements:

- fully working application - GUI or command line. Qt is preferred, but pure STL implementation is acceptable, too
- documentation of used algorithm (can be in a form of comments in code)
- application should be able to operate on external dictionary - text file where each line is another word. User should be able to choose the dictionary at runtime (or as command line argument)
- application should show derivation chain when results are shown. It can be ail+s=sail+n=... or with arrows: ails->sail->... or any similar way
- if there is more than 1 longest derivation (many words of the same length), the application can show just one result. If it will show all, we'll be extra happy

Note: At Milo, we will test your algorithm using a very big dictionary.

Task 4. Dining philosophers (QtQuick or QtWidgets)

Implementation of dining philosophers problem in Qt: "N philosophers sit at a round table with bowls. Forks are placed between each pair of adjacent philosophers. Each philosopher must alternately think and eat. However, a philosopher can only eat when he has both left and right forks". Find solution that avoids deadlock and will preserve all philosophers from starvation.

Requirements:

- Each philosopher is a subclass of QThread or a worker object running in a separate thread

- Number of philosophers is variable (initially 5), algorithm should start automatically.
- Implement MVC model for philosopher class that will allow to add / remove philosophers after dinner algorithm is started
- Application GUI should represent dining philosophers in MVC based view: QML (QtWidgets are acceptable, too)
- Implement custom MVC delegate for philosopher that will present class as avatar (image), name and state (eating / thinking) and button to remove item from model e.g.: