Assignment 2 - Kakuro

Due date:

4th November at 11:55pm

Submission details:

This is a small group project. While the best group size is 4, I will allow groups of 2 or 4, though a group of 4 will be expected to do more to get a high mark. You should attempt to form the groups yourself – please do so by the end of Week 9 of semester and let me know your group members. I will, over the weekend before Monday of Week 10, organise those not in a group into groups, and send emails out to the different group members.

Eventual assignment submission will be by groups. Ensure only **one person** does the submission of the assignment per group. The organisation of the marking is difficult enough with groups without having multiple submissions per group to look out for.

You should ensure all group member's names and email addresses are part the code files and document files that you submit.

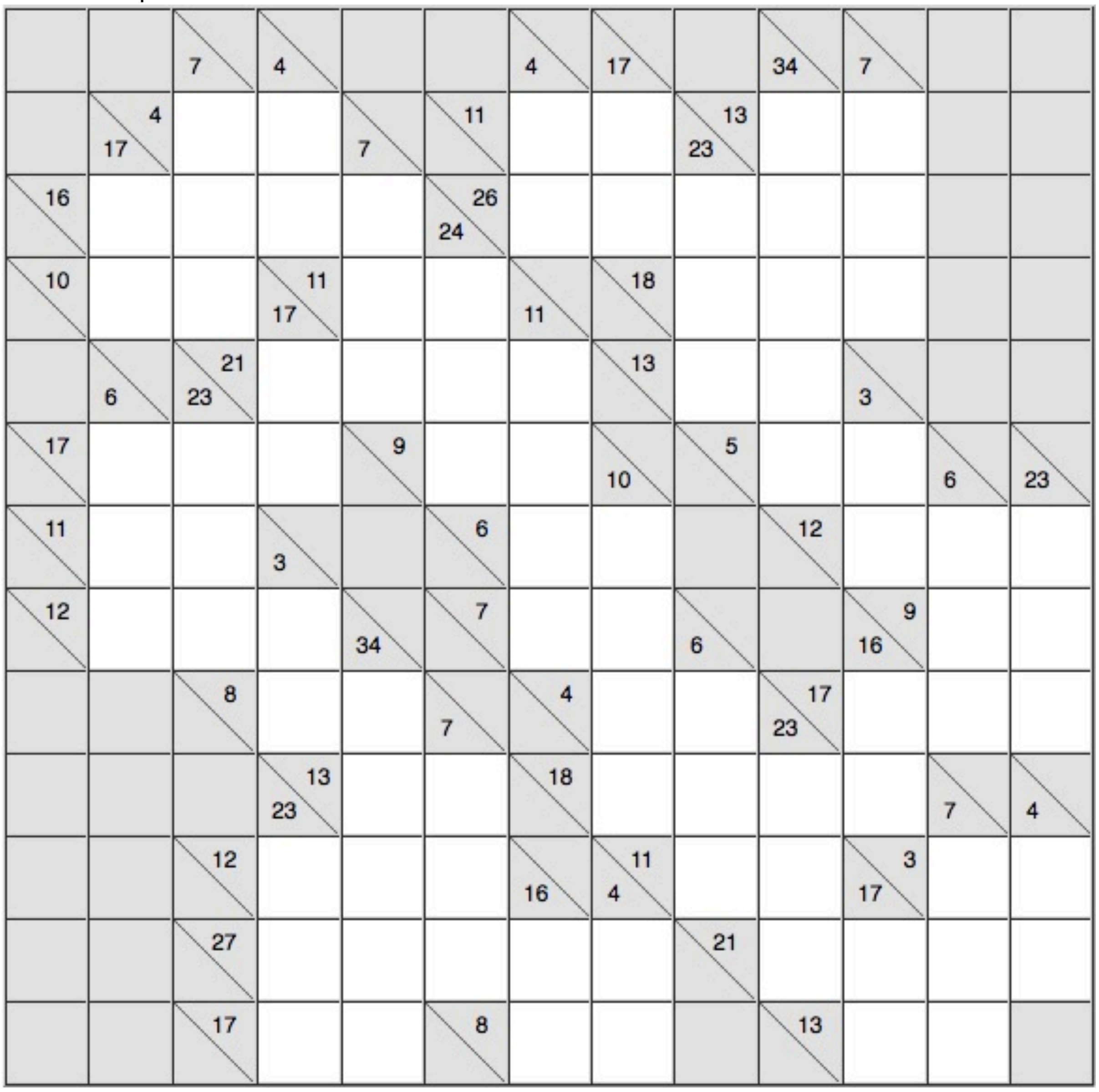
You should submit:

1. The Qt project and all associated files. This must include the .pro project file, any .ui form files, and all the .h and .cpp files. Zip these up together.
2. A document stating clearly the features that you have implemented, along with a few screen shots to illustrate these features. If a feature is partially implemented or have problems, please indicate this.
3. Any example Kakuro input files that your program can read.
4. A short document (I don't want to use the term 'user manual' as some of you will interpret this as a formal document) that describes how to use the program. Don't make this a big part of what you do – enough so that I can easily start up and know how to play with your application. The screen shots you used in the 'features implemented' document above can likely be re-used here.
5. For a bonus of up to 5% of the assignment marks (but can't take your mark above 100%), a video (produced by your phone for example) where you demonstrate the features of your working assignment.

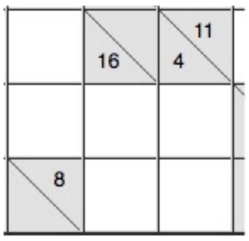
Your assignment will be marked primarily on what works, rather than the details of the code. This is not a User Interface class, so for the advanced features that you implement, while a nice user interface may receive a slightly higher mark (within the grade ranges), you are just starting with using Qt, and using the UI components well is something that develops with experience.

Overview of Kakuro

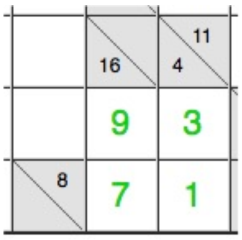
Kakuro puzzles are similar to crosswords, but instead of letters the digits (from 1 to 9) are used. Instead of clues going across and down, you are given the sum of the values across or down. There is the constraint that the digits 1 to 9 can only be used once in any 'word' going across or down. The board's squares need to be filled in with these digits in order to sum up to the specified numbers. Below is an example Kakuro problem with a 13x13 grid (effectively a 12x12 problem as you need an extra row at the top, and a column at the left, in which to put ‘down’ or ‘across’ clues.



At any point you can use the down or across clues to deduce some values in the puzzle, and then use these values to help deduce other values.



For example, in the bottom middle there is the section shown to the right. For the column with sum of 4, the two values have to be 1 and 3 (as 2 and 2 not allowed). The 3 has to be at the top, as if it is at the bottom, then as the bottom ‘across’ adds to 8, the left value will be 5. But then the ‘down’ that sums to 16 does not have a solution, as you can’t have 5+ (value from 1 to 9) that sums to 16. This means that you have to have 3, 1 in the ‘down’ that sums to 4, and 9, 7 in the column that adds to 16: as shown on the left.



You can try the puzzle online at www.kakuros.com, which provides an interactive web page to help you play Kakuro.

The Assignment

In this assignment I would like you to use Qt to develop a program with a graphical user interface to help the user solve Kakuro problems.

A minimal program that will give you at least a P1 grade should:

1. Display the grid for a Kakuro problem – you program should at least display a 13x13 grid, but you can allow the program to display a range of sizes – smaller and possibly larger.
2. Allow the user to initialise the Kakuro puzzle. This could be done, for example, by:
    a. Reading the information from a text file. Possible formats for the text file will be discussed in a lecture.
    b. Selecting a cell with the mouse and setting up the problem interactively.
    c. Generating a random problem (this is one of the possible extensions for a higher grade)
3. Allow the user to select a grid location and ask that the possible values be determined and displayed in the grid location. Note that if there is only one possible value, you should **not** make this the grid location's final value – the user needs to do this (but see extensions below).
4. Allow the user to select a value from the possible values displayed in a grid location as the final value. When this occurs, the other grid locations where you have asked for the possible values to be displayed should all be updated to use this additional information.

For particularly good implementations of 3 and 4 you can receive a small number of additional marks.

For a higher grade you need to do more. Below are some possible extensions you can do to your 'P1' level program, along with some examples of sensible combinations of extensions that will get you a HD grade.

Possible Extensions to obtain a higher grade:

1. Highlight cells where there is only one possible value. How you highlight the cell is up to you.
2. Provide an option to calculate and display the possible values for every grid location where the value is not yet specified.
3. Determine if a solution can't be found using the current values set in the grid – for example by conflicting requirements for down or across values or sums. For example, a 'down' clue may require a particular grid location to have a value of (say) 2, but the 'across' set of values already has a 2 in it. This means that a wrong choice was made by the user previously.
4. Keep track of changes to the grid values, and allow the user to 'back-track' if they have made an error.
5. Keep track of the operations requested by the user to the grid locations, and allow the user to 'replay' the way the solution was found (for example, using time delay between 'steps', or to use a key or button to 'step' forward in the replay. The way

you do this is up to you. You will get additional marks for an easy to follow replay of the steps taken.

6. If you do option 5 above, try to automate the finding of a solution. This is not trivial as sometimes you need to consider the possible values of multiple grid positions to determine the correct value of a grid position. You may be able to do a partial automation – do what you can and then let the user set a grid position's value before continuing on. You should be able to step through the automated solution. This is a hard extension - you can get a good mark for this part for a reasonable attempt.
7. Save a puzzle at any point, along with the information about the steps taken, so that you can reload the puzzle and either backtrack or continue from the current set of values.
8. Generate random Kakuro puzzles - generate a random (but reasonable) puzzle 'boards', then generate a random 'solution' (fill the board consistently with digits 1 to 9), and then generate the appropriate down and across clues.

You can obtain a HD by implementing:

- Extensions 1, 2, 3, 4, or:
- Extensions 4, 5, 6, or:
- Extensions 4, 5, 7, or:
- Extension 1 and 8, or:
- Suggesting an alternative set of extensions to the course coordinator and obtaining approval for this being the goal for achieving a HD.

Generally, you will need 3 or 4 extensions to get a HD, depending on how hard the extensions are. For reasonably sized extensions, each working extension takes you up a grade (C, D, HD). Some easy extensions (such as 1 and 2 above) are too small to take you up a whole grade, so in the first suggested set of extensions above, 1 and 2 together take you to a Credit grade. Some extensions are difficult, and a reasonable attempt will get you the additional grade. An example of this is extension 6.

Getting Started

1. Install Qt

Download and install the latest version of the open source distribution for Qt – this was version 5.7.0 when I wrote up this assignment. Go to the page http://www.qt.io/download/ and follow the links to get the open source version for your computer platform.

You may find the following pages useful:

1. QMake variable reference - http://doc.qt.io/qt-5/qmake-variable-reference.html#makefile
   I set the c++14 compiler flag by adding the line:
   ```
   CONFIG += c++14
   ```
   to the projects ".pro" file.
2. Specifically for the Mac, I also added to the project file (see https://forum.qt.io/topic/58926/solved-xcode-7-and-qt-error) the line:
   ```
   QMAKE_MAC_SDK=macosx10.11
   ```

2. Learn how to use Qt

Once installed, start the QtCreator application, and then work through some of the on-line tutorials and the materials on the course web pages to learn the basics of using Qt for GUI development.

Some useful starting pages:

1. Qt for Beginners - http://wiki.qt.io/Qt_for_Beginners
2. Qt Documentation - http://doc.qt.io/qt-5/index.html
3. A very good example to start with – Creating a Qt Widget Based Application http://doc.qt.io/qtcreator/creator-writing-program.html
4. Qt Creator documentation - http://doc.qt.io/qtcreator/creator-getting-started.html (and see links on right hand side of this page for rest of Qt Creator manual.
5. All Qt C++ Classes - http://doc.qt.io/qt-5/classes.html
   Note: You **must** use classes such as QVector, QString etc rather than the standard STL classes – these Qt classes are all safe to use in a user interface where an operation on the data structure may be interrupted by a user initiated event. Some classes I have used include: QVector, QSet, QString, QFont, QPushButton, QMenu, QGridLayout, QMouseEvent, ….
6. You will need to understand Signals and Slots – have a look at a quick read on this at: http://www.bogotobogo.com/Qt/Qt5_SignalsSlotsGui.php (this site has other tutorial examples on using Qt), and then look at the Qt documentation on Signals and Slots at: http://doc.qt.io/qt-5/signalsandslots.html

3. Use the web for information.

Use the web to search for solutions to problems when you have them – there are sites such as stackoverflow.com that have information on how to solve very specific issues when using Qt. Some examples:

1. There is a nice page with examples on using the grid layout at http://www.notmart.org/index.php/software/fighting_against_qt_designer_gri
2. You will likely need to use set operations – if so, there always Wikipedia: https://en.wikipedia.org/wiki/Set_(mathematics)

4. Design before you code

For those who have done the user interface design course, do a design of your user interface by hand first, and then implement using Qt Creator. You can add the logic needed to support the user interface within Qt Creator – this becomes your development environment.

For everyone, work out how you are going to represent the Kakuro problem as a data structure, and how you will manipulate it to do the processing you need.
I **strongly suggest** you work out your data structures and algorithms before you code up the problem. You can step through the algorithms by hand on a small problem (3x3 or 4x4) to check that they work before you try to code them. This also gives you the opportunity to **learn more about Qt before you try to code up your assignment**.

5. Some Hints

1. I found Qt Creator occasionally crashed when editing code. This may be specific to the Mac version – but probably not - so you may find it happens to you too. Qt Creator does do checkpoint saves at time intervals, but I tend to save files frequently.
2. Use size constraints for widgets and layouts. These are very useful to get the UI to behave as you want.
3. You can add widgets (such as buttons) to a layout to see how it will work and how the GUI design tool is creating the objects in the layout (look in the temporary files created in the target build directory), and once you see how the different widget properties are being set, you can then remove the widgets from your layout (but using layout size constraints to keep the layout as you want), and add the widgets back using code (and so allow for different size Kakuro problems). I did this last year for the 81 buttons that I used for the Sudoku grid for last year's project. In the layout editor I added and configured a few QPushButton objects, but I replaced these by buttons of a class I derived from QPushButton. This also allowed me to add the connections between signals and slots for the buttons via code rather than the user interface design tool (which would be tedious for 81 buttons)
4. Qt Creator (at least on the Mac) has an "Application Output" window, so that you can write information to standard output while the application is running to help with debugging your code.